

Optimization Wiki

Contents

| | |
|---|----------|
| 1 Mathematical optimization | 1 |
| 1.1 Optimization problems | 1 |
| 1.2 Notation | 2 |
| 1.2.1 Minimum and maximum value of a function | 2 |
| 1.2.2 Optimal input arguments | 2 |
| 1.3 History | 2 |
| 1.4 Major subfields | 2 |
| 1.4.1 Multi-objective optimization | 3 |
| 1.4.2 Multi-modal optimization | 4 |
| 1.5 Classification of critical points and extrema | 4 |
| 1.5.1 Feasibility problem | 4 |
| 1.5.2 Existence | 4 |
| 1.5.3 Necessary conditions for optimality | 4 |
| 1.5.4 Sufficient conditions for optimality | 4 |
| 1.5.5 Sensitivity and continuity of optima | 5 |
| 1.5.6 Calculus of optimization | 5 |
| 1.6 Computational optimization techniques | 5 |
| 1.6.1 Optimization algorithms | 5 |
| 1.6.2 Iterative methods | 5 |
| 1.6.3 Global convergence | 6 |
| 1.6.4 Heuristics | 6 |
| 1.7 Applications | 7 |
| 1.7.1 Mechanics and engineering | 7 |
| 1.7.2 Economics | 7 |
| 1.7.3 Operations research | 7 |
| 1.7.4 Control engineering | 7 |
| 1.7.5 Petroleum engineering | 7 |
| 1.7.6 Molecular modeling | 7 |
| 1.8 Solvers | 7 |
| 1.9 See also | 7 |
| 1.10 Notes | 7 |
| 1.11 Further reading | 8 |

| | | |
|----------|--|-----------|
| 1.11.1 | Comprehensive | 8 |
| 1.11.2 | Continuous optimization | 9 |
| 1.11.3 | Combinatorial optimization | 9 |
| 1.11.4 | Relaxation (extension method) | 9 |
| 1.12 | Journals | 10 |
| 1.13 | External links | 10 |
| 2 | Golden section search | 11 |
| 2.1 | Basic idea | 11 |
| 2.2 | Probe point selection | 11 |
| 2.3 | Termination condition | 12 |
| 2.4 | Algorithm | 12 |
| 2.4.1 | Iterative algorithm | 12 |
| 2.4.2 | Recursive algorithm | 12 |
| 2.5 | Fibonacci search | 12 |
| 2.6 | See also | 13 |
| 2.7 | References | 13 |
| 3 | Powell's method | 14 |
| 3.1 | References | 14 |
| 3.2 | External links | 14 |
| 4 | Line search | 15 |
| 4.1 | Example use | 15 |
| 4.2 | Algorithms | 15 |
| 4.2.1 | Direct search methods | 15 |
| 4.3 | See also | 15 |
| 4.4 | References | 15 |
| 5 | Nelder–Mead method | 16 |
| 5.1 | Overview | 16 |
| 5.2 | One possible variation of the NM algorithm | 16 |
| 5.3 | See also | 17 |
| 5.4 | References | 17 |
| 5.4.1 | Further reading | 18 |
| 5.5 | External links | 18 |
| 6 | Successive parabolic interpolation | 19 |
| 6.1 | Advantages | 19 |
| 6.2 | Disadvantages | 19 |
| 6.3 | Improvements | 19 |
| 6.4 | See also | 19 |
| 6.5 | References | 19 |

| | |
|---|-----------|
| 7 Trust region | 20 |
| 7.1 Example | 20 |
| 7.2 References | 20 |
| 7.3 External links | 20 |
| 8 Wolfe conditions | 21 |
| 8.1 Armijo rule and curvature | 21 |
| 8.2 Strong Wolfe condition on curvature | 21 |
| 8.3 Rationale | 21 |
| 8.4 References | 22 |
| 9 Broyden–Fletcher–Goldfarb–Shanno algorithm | 23 |
| 9.1 Rationale | 23 |
| 9.2 Algorithm | 23 |
| 9.3 Implementations | 24 |
| 9.4 See also | 24 |
| 9.5 Notes | 24 |
| 9.6 Bibliography | 24 |
| 9.7 External links | 25 |
| 10 Limited-memory BFGS | 26 |
| 10.1 Algorithm | 26 |
| 10.2 Applications | 26 |
| 10.3 Variants | 26 |
| 10.3.1 L-BFGS-B | 27 |
| 10.3.2 OWL-QN | 27 |
| 10.3.3 O-LBFGS | 27 |
| 10.4 Implementations | 27 |
| 10.4.1 Implementations of variants | 27 |
| 10.5 Works cited | 27 |
| 10.6 Further reading | 28 |
| 11 Davidon–Fletcher–Powell formula | 29 |
| 11.1 See also | 29 |
| 11.2 References | 29 |
| 12 Symmetric rank-one | 30 |
| 12.1 See also | 30 |
| 12.2 Notes | 30 |
| 12.3 References | 31 |
| 13 Gauss–Newton algorithm | 32 |
| 13.1 Description | 32 |
| 13.2 Notes | 32 |

| | |
|--|-----------|
| 13.3 Example | 33 |
| 13.4 Convergence properties | 33 |
| 13.5 Derivation from Newton's method | 33 |
| 13.6 Improved versions | 34 |
| 13.7 Other applications | 34 |
| 13.8 Related algorithms | 34 |
| 13.9 Notes | 35 |
| 13.10 References | 35 |
| 14 Gradient descent | 36 |
| 14.1 Description | 36 |
| 14.1.1 Examples | 36 |
| 14.1.2 Limitations | 37 |
| 14.2 Solution of a linear system | 37 |
| 14.3 Solution of a non-linear system | 38 |
| 14.4 Comments | 39 |
| 14.5 A computational example | 39 |
| 14.6 Extensions | 39 |
| 14.6.1 Fast proximal gradient method | 39 |
| 14.6.2 The momentum method | 40 |
| 14.7 See also | 40 |
| 14.8 References | 40 |
| 14.9 External links | 40 |
| 15 Levenberg–Marquardt algorithm | 41 |
| 15.1 The problem | 41 |
| 15.2 The solution | 41 |
| 15.2.1 Choice of damping parameter | 42 |
| 15.3 Example | 42 |
| 15.4 See also | 43 |
| 15.5 Notes | 43 |
| 15.6 References | 43 |
| 15.7 External links | 43 |
| 15.7.1 Descriptions | 43 |
| 15.7.2 Implementations | 43 |
| 16 Nonlinear conjugate gradient method | 45 |
| 16.1 External links | 46 |
| 17 Newton's method in optimization | 47 |
| 17.1 Method | 47 |
| 17.2 Geometric interpretation | 47 |
| 17.3 Higher dimensions | 47 |

| | |
|--|-----------|
| 17.4 See also | 48 |
| 17.5 Notes | 48 |
| 17.6 References | 48 |
| 18 Barrier function | 49 |
| 18.1 Logarithmic barrier function | 49 |
| 18.1.1 Higher dimensions | 49 |
| 18.1.2 Formal definition | 49 |
| 18.2 References | 49 |
| 19 Penalty method | 50 |
| 19.1 Example | 50 |
| 19.2 Practical application | 50 |
| 19.3 Barrier methods | 50 |
| 19.4 See also | 50 |
| 19.5 References | 50 |
| 20 Augmented Lagrangian method | 51 |
| 20.1 General method | 51 |
| 20.2 Comparison with penalty methods | 52 |
| 20.3 Alternating direction method of multipliers | 52 |
| 20.4 Software | 52 |
| 20.5 See also | 52 |
| 20.6 References | 52 |
| 20.7 Bibliography | 52 |
| 21 Sequential quadratic programming | 53 |
| 21.1 Algorithm basics | 53 |
| 21.2 See also | 53 |
| 21.3 References | 53 |
| 21.4 External links | 53 |
| 22 Successive linear programming | 54 |
| 22.1 References | 54 |
| 22.2 External links | 54 |
| 23 Cutting-plane method | 55 |
| 23.1 Gomory's cut | 55 |
| 23.2 Convex optimization | 56 |
| 23.3 See also | 56 |
| 23.4 References | 56 |
| 23.5 External links | 56 |
| 24 Frank–Wolfe algorithm | 57 |

| | |
|---|-----------|
| 24.1 Problem statement | 57 |
| 24.2 Algorithm | 57 |
| 24.3 Properties | 57 |
| 24.4 Lower bounds on the solution value, and primal-dual analysis | 58 |
| 24.5 Notes | 58 |
| 24.6 Bibliography | 58 |
| 24.7 External links | 58 |
| 24.8 See also | 58 |
| 25 Subgradient method | 59 |
| 25.1 Classical subgradient rules | 59 |
| 25.1.1 Step size rules | 59 |
| 25.1.2 Convergence results | 60 |
| 25.2 Subgradient-projection & bundle methods | 60 |
| 25.3 Constrained optimization | 60 |
| 25.3.1 Projected subgradient | 60 |
| 25.3.2 General constraints | 60 |
| 25.4 References | 60 |
| 25.5 Further reading | 61 |
| 25.6 External links | 61 |
| 26 Ellipsoid method | 62 |
| 26.1 History | 62 |
| 26.2 Description | 62 |
| 26.3 Unconstrained minimization | 62 |
| 26.4 Inequality-constrained minimization | 63 |
| 26.4.1 Application to linear programming | 63 |
| 26.5 Performance | 63 |
| 26.6 Notes | 63 |
| 26.7 Further reading | 63 |
| 26.8 External links | 64 |
| 27 Karmarkar's algorithm | 65 |
| 27.1 The Algorithm | 65 |
| 27.2 Example | 65 |
| 27.3 Patent controversy - <i>Can Mathematics be patented?</i> | 65 |
| 27.4 References | 66 |
| 28 Simplex algorithm | 68 |
| 28.1 Overview | 68 |
| 28.2 Standard form | 69 |
| 28.3 Simplex tableaux | 70 |
| 28.4 Pivot operations | 70 |

| | |
|--|-----------|
| 28.5 Algorithm | 70 |
| 28.5.1 Entering variable selection | 70 |
| 28.5.2 Leaving variable selection | 70 |
| 28.5.3 Example | 71 |
| 28.6 Finding an initial canonical tableau | 71 |
| 28.6.1 Example | 72 |
| 28.7 Advanced topics | 72 |
| 28.7.1 Implementation | 72 |
| 28.7.2 Degeneracy: Stalling and cycling | 72 |
| 28.7.3 Efficiency | 73 |
| 28.8 Other algorithms | 73 |
| 28.9 Linear-fractional programming | 73 |
| 28.10 See also | 73 |
| 28.11 Notes | 73 |
| 28.12 References | 75 |
| 28.13 Further reading | 75 |
| 28.14 External links | 75 |
| 29 Revised simplex method | 76 |
| 29.1 Problem formulation | 76 |
| 29.2 Algorithmic description | 76 |
| 29.2.1 Optimality conditions | 76 |
| 29.2.2 Pivot operation | 77 |
| 29.3 Numerical example | 77 |
| 29.4 Practical issues | 77 |
| 29.4.1 Degeneracy | 77 |
| 29.4.2 Basis representation | 77 |
| 29.5 Notes and references | 78 |
| 29.5.1 Notes | 78 |
| 29.5.2 References | 78 |
| 29.5.3 Bibliography | 78 |
| 30 Criss-cross algorithm | 79 |
| 30.1 History | 79 |
| 30.2 Comparison with the simplex algorithm for linear optimization | 79 |
| 30.3 Description | 80 |
| 30.4 Computational complexity: Worst and average cases | 80 |
| 30.5 Variants | 80 |
| 30.5.1 Other optimization problems with linear constraints | 81 |
| 30.5.2 Vertex enumeration | 81 |
| 30.5.3 Oriented matroids | 81 |
| 30.6 Summary | 81 |

| | |
|---|------------|
| 30.7 See also | 81 |
| 30.8 Notes | 81 |
| 30.9 References | 82 |
| 30.10 External links | 83 |
| 31 Lemke's algorithm | 84 |
| 31.1 References | 84 |
| 31.2 External links | 84 |
| 32 Approximation algorithm | 85 |
| 32.1 Performance guarantees | 85 |
| 32.2 Algorithm design techniques | 86 |
| 32.3 Epsilon terms | 86 |
| 32.4 See also | 86 |
| 32.5 Citations | 87 |
| 32.6 References | 87 |
| 32.7 External links | 87 |
| 33 Dynamic programming | 88 |
| 33.1 Overview | 88 |
| 33.1.1 Dynamic programming in mathematical optimization | 89 |
| 33.1.2 Dynamic programming in bioinformatics | 89 |
| 33.1.3 Dynamic programming in computer programming | 89 |
| 33.2 Example: Mathematical optimization | 90 |
| 33.2.1 Optimal consumption and saving | 90 |
| 33.3 Examples: Computer algorithms | 91 |
| 33.3.1 Dijkstra's algorithm for the shortest path problem | 91 |
| 33.3.2 Fibonacci sequence | 91 |
| 33.3.3 A type of balanced 0–1 matrix | 91 |
| 33.3.4 Checkerboard | 92 |
| 33.3.5 Sequence alignment | 93 |
| 33.3.6 Tower of Hanoi puzzle | 93 |
| 33.3.7 Egg dropping puzzle | 94 |
| 33.3.8 Matrix chain multiplication | 95 |
| 33.4 History | 96 |
| 33.5 Algorithms that use dynamic programming | 97 |
| 33.6 See also | 97 |
| 33.7 References | 98 |
| 33.8 Further reading | 98 |
| 33.9 External links | 98 |
| 34 Greedy algorithm | 100 |
| 34.1 Specifics | 100 |

| | |
|--|------------|
| 34.1.1 Cases of failure | 100 |
| 34.2 Types | 101 |
| 34.3 Applications | 101 |
| 34.4 Examples | 101 |
| 34.5 See also | 102 |
| 34.6 Notes | 102 |
| 34.7 References | 102 |
| 34.8 External links | 102 |
| 35 Integer programming | 103 |
| 35.1 Canonical and standard form for ILPs | 103 |
| 35.2 Example | 103 |
| 35.3 Variants | 104 |
| 35.4 Example problems that can be formulated as ILPs | 104 |
| 35.5 Applications | 104 |
| 35.5.1 Production planning | 104 |
| 35.5.2 Scheduling | 104 |
| 35.5.3 Telecommunications networks | 104 |
| 35.5.4 Cellular networks | 104 |
| 35.6 Algorithms | 104 |
| 35.6.1 Using total unimodularity | 105 |
| 35.6.2 Exact algorithms | 105 |
| 35.6.3 Heuristic methods | 105 |
| 35.7 References | 105 |
| 35.8 Further reading | 106 |
| 35.9 External links | 106 |
| 36 Branch and bound | 107 |
| 36.1 Overview | 107 |
| 36.1.1 Generic version | 107 |
| 36.2 Applications | 108 |
| 36.3 See also | 108 |
| 36.4 References | 108 |
| 37 Branch and cut | 109 |
| 37.1 Description of the Algorithm | 109 |
| 37.2 Branching Strategies | 109 |
| 37.3 External links | 110 |
| 37.4 References | 110 |
| 38 Bellman–Ford algorithm | 111 |
| 38.1 Algorithm | 111 |
| 38.2 Proof of correctness | 112 |

| | |
|--|------------|
| 38.3 Finding negative cycles | 112 |
| 38.4 Applications in routing | 112 |
| 38.5 Improvements | 113 |
| 38.6 Notes | 113 |
| 38.7 References | 113 |
| 38.7.1 Original sources | 113 |
| 38.7.2 Secondary sources | 113 |
| 38.8 External links | 114 |
| 39 Borůvka's algorithm | 115 |
| 39.1 Pseudocode | 115 |
| 39.2 Complexity | 115 |
| 39.3 Example | 115 |
| 39.4 Other algorithms | 115 |
| 39.5 Notes | 116 |
| 40 Dijkstra's algorithm | 117 |
| 40.1 History | 117 |
| 40.2 Algorithm | 117 |
| 40.3 Description | 118 |
| 40.4 Pseudocode | 119 |
| 40.4.1 Using a priority queue | 119 |
| 40.5 Running time | 119 |
| 40.5.1 Practical optimizations and infinite graphs | 120 |
| 40.5.2 Specialized variants | 120 |
| 40.6 Related problems and algorithms | 120 |
| 40.6.1 Dynamic programming perspective | 121 |
| 40.7 See also | 121 |
| 40.8 Notes | 121 |
| 40.9 References | 122 |
| 40.10 External links | 122 |
| 41 Floyd–Warshall algorithm | 123 |
| 41.1 Algorithm | 123 |
| 41.2 Example | 123 |
| 41.3 Behavior with negative cycles | 124 |
| 41.4 Path reconstruction | 124 |
| 41.5 Analysis | 124 |
| 41.6 Applications and generalizations | 124 |
| 41.7 Implementations | 125 |
| 41.8 See also | 125 |
| 41.9 References | 125 |

| | |
|--------------------------------------|------------|
| 41.10 External links | 126 |
| 42 Johnson's algorithm | 127 |
| 42.1 Algorithm description | 127 |
| 42.2 Example | 127 |
| 42.3 Correctness | 127 |
| 42.4 Analysis | 128 |
| 42.5 References | 128 |
| 42.6 External links | 128 |
| 43 Kruskal's algorithm | 129 |
| 43.1 Description | 129 |
| 43.2 Pseudocode | 129 |
| 43.3 Complexity | 129 |
| 43.4 Example | 130 |
| 43.5 Proof of correctness | 130 |
| 43.5.1 Spanning tree | 130 |
| 43.5.2 Minimality | 130 |
| 43.6 See also | 130 |
| 43.7 References | 130 |
| 43.8 External links | 130 |
| 44 Dinic's algorithm | 131 |
| 44.1 History | 131 |
| 44.2 Definition | 131 |
| 44.3 Algorithm | 131 |
| 44.4 Analysis | 132 |
| 44.4.1 Special cases | 132 |
| 44.5 Example | 132 |
| 44.6 See also | 132 |
| 44.7 Notes | 132 |
| 44.8 References | 132 |
| 45 Edmonds–Karp algorithm | 133 |
| 45.1 Algorithm | 133 |
| 45.2 Pseudocode | 133 |
| 45.3 Example | 133 |
| 45.4 Notes | 134 |
| 45.5 References | 134 |
| 46 Ford–Fulkerson algorithm | 135 |
| 46.1 Algorithm | 135 |
| 46.2 Complexity | 135 |

| | |
|--|------------|
| 46.3 Integral example | 136 |
| 46.4 Non-terminating example | 136 |
| 46.5 Python implementation | 136 |
| 46.5.1 Usage example | 136 |
| 46.6 Notes | 137 |
| 46.7 References | 137 |
| 46.8 See also | 137 |
| 46.9 External links | 137 |
| 47 Push–relabel maximum flow algorithm | 138 |
| 47.1 History | 138 |
| 47.2 Concepts | 138 |
| 47.2.1 Definitions and notations | 138 |
| 47.2.2 Operations | 139 |
| 47.3 The generic push–relabel algorithm | 139 |
| 47.3.1 Description | 139 |
| 47.3.2 Correctness | 140 |
| 47.3.3 Time complexity | 140 |
| 47.3.4 Example | 140 |
| 47.4 Practical implementations | 141 |
| 47.4.1 “Current-edge” data structure and discharge operation | 141 |
| 47.4.2 Active vertex selection rules | 141 |
| 47.4.3 Implementation techniques | 141 |
| 47.5 Sample implementations | 142 |
| 47.6 References | 142 |
| 48 Evolutionary algorithm | 144 |
| 48.1 Implementation of biological processes | 144 |
| 48.2 Evolutionary algorithm types | 144 |
| 48.3 Related techniques | 145 |
| 48.4 Other population-based metaheuristic method | 145 |
| 48.5 See also | 145 |
| 48.6 Gallery ^[5] | 146 |
| 48.7 References | 146 |
| 48.8 Bibliography | 146 |
| 49 Hill climbing | 147 |
| 49.1 Mathematical description | 147 |
| 49.2 Variants | 147 |
| 49.3 Problems | 148 |
| 49.3.1 Local maxima | 148 |
| 49.3.2 Ridges and alleys | 148 |

| | |
|--|------------|
| 49.3.3 Plateau | 149 |
| 49.4 Pseudocode | 149 |
| 49.5 See also | 149 |
| 49.6 References | 149 |
| 49.7 External links | 149 |
| 50 Local search (optimization) | 150 |
| 50.1 Examples | 150 |
| 50.2 Description | 150 |
| 50.3 See also | 151 |
| 50.3.1 Real-valued search-spaces | 151 |
| 50.4 Bibliography | 151 |
| 51 Simulated annealing | 152 |
| 51.1 Overview | 152 |
| 51.1.1 The basic iteration | 152 |
| 51.1.2 The neighbours of a state | 152 |
| 51.1.3 Acceptance probabilities | 153 |
| 51.1.4 The annealing schedule | 153 |
| 51.2 Pseudocode | 153 |
| 51.3 Selecting the parameters | 153 |
| 51.3.1 Diameter of the search graph | 154 |
| 51.3.2 Transition probabilities | 154 |
| 51.3.3 Acceptance probabilities | 154 |
| 51.3.4 Efficient candidate generation | 154 |
| 51.3.5 Barrier avoidance | 154 |
| 51.3.6 Cooling schedule | 155 |
| 51.4 Restarts | 155 |
| 51.5 Related methods | 155 |
| 51.6 See also | 156 |
| 51.7 References | 156 |
| 51.8 Further reading | 156 |
| 51.9 External links | 156 |
| 52 Tabu search | 157 |
| 52.1 Background | 157 |
| 52.2 Basic Description | 157 |
| 52.3 Types of Memory | 158 |
| 52.4 Pseudocode | 158 |
| 52.5 Example: Traveling salesman problem | 159 |
| 52.6 References | 159 |
| 52.7 External links | 159 |

| | |
|---|------------|
| 53 Active set method | 160 |
| 53.1 Active set methods | 160 |
| 53.2 References | 160 |
| 54 Adaptive coordinate descent | 161 |
| 54.1 Relevant approaches | 161 |
| 54.2 See also | 161 |
| 54.3 References | 161 |
| 54.4 External links | 162 |
| 55 Alpha–beta pruning | 163 |
| 55.1 History | 163 |
| 55.2 Improvements over naive minimax | 163 |
| 55.3 Pseudocode | 164 |
| 55.4 Heuristic improvements | 164 |
| 55.5 Other algorithms | 165 |
| 55.6 See also | 165 |
| 55.7 References | 165 |
| 55.8 External links | 165 |
| 56 Artificial bee colony algorithm | 167 |
| 56.1 Algorithm | 167 |
| 56.2 Application to real-world problems | 167 |
| 56.3 See also | 168 |
| 56.4 References | 168 |
| 56.5 External links | 168 |
| 57 Auction algorithm | 169 |
| 57.1 Comparisons | 169 |
| 57.2 See also | 170 |
| 57.3 References | 170 |
| 57.4 External links | 170 |
| 58 Automatic label placement | 171 |
| 58.1 Rule-based algorithms | 171 |
| 58.2 Local optimization algorithms | 171 |
| 58.3 Divide-and-conquer algorithms | 172 |
| 58.4 2-satisfiability algorithms | 172 |
| 58.5 Other algorithms | 172 |
| 58.6 Notes | 172 |
| 58.7 References | 172 |
| 58.8 External links | 172 |
| 59 Bees algorithm | 173 |

| | |
|--|------------|
| 59.1 Honey bees foraging strategy in nature | 173 |
| 59.2 The Bees Algorithm | 173 |
| 59.3 Applications | 174 |
| 59.4 See also | 174 |
| 59.5 References | 174 |
| 59.6 External links | 175 |
| 60 Benson's algorithm | 176 |
| 60.1 Idea of algorithm | 176 |
| 60.2 Implementations | 176 |
| 60.2.1 Bensolve - a free VLP solver (C programming language) | 176 |
| 60.3 References | 176 |
| 61 Berndt–Hall–Hall–Hausman algorithm | 177 |
| 61.1 Usage | 177 |
| 61.2 See also | 177 |
| 61.3 Further reading | 177 |
| 62 Big M method | 178 |
| 62.1 Algorithm | 178 |
| 62.2 Other usage | 178 |
| 62.3 See also | 178 |
| 62.4 References and external links | 178 |
| 63 Bin packing problem | 180 |
| 63.1 Formal statement | 180 |
| 63.2 First-fit algorithm | 180 |
| 63.3 Analysis of approximate algorithms | 181 |
| 63.4 Exact algorithm | 181 |
| 63.5 Software | 181 |
| 63.6 See also | 181 |
| 63.7 Notes | 181 |
| 63.8 References | 181 |
| 63.9 External links | 182 |
| 64 Bland's rule | 183 |
| 64.1 Algorithm | 183 |
| 64.2 Extensions to oriented matroids | 183 |
| 64.3 Notes | 183 |
| 64.4 Further reading | 183 |
| 65 BOBYQA | 185 |
| 65.1 See also | 185 |
| 65.2 References | 185 |

| | |
|--|------------|
| 65.3 External links | 185 |
| 66 Branch and price | 186 |
| 66.1 Description of Algorithm | 186 |
| 66.2 Applications of Branch and Price | 186 |
| 66.3 See also | 187 |
| 66.4 External References | 187 |
| 66.5 References | 187 |
| 67 CMA-ES | 188 |
| 67.1 Principles | 188 |
| 67.2 Algorithm | 189 |
| 67.3 Example code in MATLAB/Octave | 190 |
| 67.4 Theoretical Foundations | 191 |
| 67.4.1 Variable Metric | 191 |
| 67.4.2 Maximum-Likelihood Updates | 191 |
| 67.4.3 Natural Gradient Descent in the Space of Sample Distributions | 192 |
| 67.4.4 Stationarity or Unbiasedness | 193 |
| 67.4.5 Invariance | 193 |
| 67.4.6 Convergence | 193 |
| 67.4.7 Interpretation as Coordinate System Transformation | 193 |
| 67.5 Performance in Practice | 194 |
| 67.6 Variations and Extensions | 194 |
| 67.7 See also | 194 |
| 67.8 References | 194 |
| 67.9 Bibliography | 195 |
| 67.10 External links | 195 |
| 68 COBYLA | 196 |
| 68.1 References | 196 |
| 68.2 See also | 196 |
| 68.3 External links | 196 |
| 69 Coffman–Graham algorithm | 197 |
| 69.1 Problem statement and applications | 197 |
| 69.2 The algorithm | 197 |
| 69.3 Analysis | 198 |
| 69.4 References | 198 |
| 70 Column generation | 200 |
| 71 Constructive cooperative coevolution | 201 |
| 71.1 Algorithm | 201 |
| 71.2 Applications | 201 |

| | |
|--|------------|
| 71.3 See also | 201 |
| 71.4 References | 202 |
| 72 Crew scheduling | 203 |
| 72.1 Complex | 203 |
| 72.2 4 Parts | 203 |
| 72.3 Disruptions | 204 |
| 72.4 Related topics (systems) | 204 |
| 72.5 Related topics (algorithms and software) | 204 |
| 72.6 References | 204 |
| 73 Cross-entropy method | 205 |
| 73.1 Estimation via importance sampling | 205 |
| 73.2 Generic CE algorithm | 205 |
| 73.3 Continuous optimization—example | 205 |
| 73.3.1 Pseudo-code | 206 |
| 73.4 Related methods | 206 |
| 73.5 See also | 206 |
| 73.6 References | 206 |
| 73.7 External links | 206 |
| 74 Cuckoo search | 207 |
| 74.1 Random walks and the step size | 207 |
| 74.2 Implementation | 208 |
| 74.3 Modified cuckoo search | 208 |
| 74.4 Multiobjective cuckoo search (MOCS) | 208 |
| 74.5 Hybridization | 208 |
| 74.6 Applications | 208 |
| 74.7 References | 209 |
| 75 Derivation of the conjugate gradient method | 212 |
| 75.1 Derivation from the conjugate direction method | 212 |
| 75.1.1 The conjugate direction method | 212 |
| 75.2 Derivation from the Arnoldi/Lanczos iteration | 212 |
| 75.2.1 The general Arnoldi method | 212 |
| 75.2.2 The direct Lanczos method | 213 |
| 75.2.3 The conjugate gradient method from imposing orthogonality and conjugacy | 213 |
| 75.3 References | 214 |
| 76 Derivative-free optimization | 215 |
| 76.1 Introduction | 215 |
| 76.2 Examples of derivative-free optimization problems | 215 |
| 76.3 Derivative-free optimization algorithms | 215 |

| | |
|---|------------|
| 76.4 Software | 215 |
| 76.5 See also | 215 |
| 76.6 References | 215 |
| 77 Destination dispatch | 216 |
| 77.1 Algorithms | 216 |
| 77.2 Limitations | 216 |
| 77.3 Manufacturers | 216 |
| 77.4 References | 216 |
| 77.5 External links | 216 |
| 78 Differential evolution | 217 |
| 78.1 Algorithm | 217 |
| 78.2 Parameter selection | 218 |
| 78.3 Variants | 218 |
| 78.4 Sample code | 218 |
| 78.5 See also | 219 |
| 78.6 References | 219 |
| 78.7 External links | 219 |
| 79 Divide and conquer algorithms | 220 |
| 79.1 Decrease and conquer | 220 |
| 79.2 Early historical examples | 220 |
| 79.3 Advantages | 221 |
| 79.3.1 Solving difficult problems | 221 |
| 79.3.2 Algorithm efficiency | 221 |
| 79.3.3 Parallelism | 221 |
| 79.3.4 Memory access | 221 |
| 79.3.5 Roundoff control | 221 |
| 79.4 Implementation issues | 221 |
| 79.4.1 Recursion | 221 |
| 79.4.2 Explicit stack | 222 |
| 79.4.3 Stack size | 222 |
| 79.4.4 Choosing the base cases | 222 |
| 79.4.5 Sharing repeated subproblems | 222 |
| 79.5 See also | 222 |
| 79.6 References | 223 |
| 79.7 External links | 223 |
| 80 Dykstra's projection algorithm | 224 |
| 80.1 Algorithm | 224 |
| 80.2 References | 224 |

| | |
|---|------------|
| 81 Eagle strategy | 226 |
| 81.1 References | 226 |
| 82 Evolutionary programming | 227 |
| 82.1 See also | 227 |
| 82.2 References | 227 |
| 82.3 External links | 227 |
| 83 Expectation–maximization algorithm | 228 |
| 83.1 History | 228 |
| 83.2 Introduction | 228 |
| 83.3 Description | 229 |
| 83.4 Properties | 230 |
| 83.5 Proof of correctness | 230 |
| 83.6 Alternative description | 230 |
| 83.7 Applications | 231 |
| 83.8 Filtering and smoothing EM algorithms | 231 |
| 83.9 Variants | 231 |
| 83.9.1 α -EM algorithm | 231 |
| 83.10 Relation to variational Bayes methods | 232 |
| 83.11 Geometric interpretation | 232 |
| 83.12 Examples | 232 |
| 83.12.1 Gaussian mixture | 232 |
| 83.12.2 Truncated and censored regression | 233 |
| 83.13 See also | 234 |
| 83.14 Further reading | 234 |
| 83.15 References | 234 |
| 83.16 External links | 235 |
| 84 Extremal optimization | 236 |
| 84.1 Relation to self-organized criticality | 236 |
| 84.2 Relation to computational complexity | 236 |
| 84.3 The technique | 236 |
| 84.4 Variations on the theme and applications | 237 |
| 84.5 References | 237 |
| 84.6 Web resources | 237 |
| 84.7 See also | 238 |
| 85 Fernandez’s method | 239 |
| 85.1 Further reading | 239 |
| 86 Firefly algorithm | 240 |
| 86.1 Algorithm description | 240 |

| | |
|--|------------|
| 86.2 Implementation Guides | 240 |
| 86.3 Variants of Firefly Algorithm | 241 |
| 86.3.1 Discrete Firefly Algorithm (DFA) | 241 |
| 86.3.2 Multiobjective FA | 241 |
| 86.3.3 Lagrangian FA | 241 |
| 86.3.4 Chaotic FA | 241 |
| 86.3.5 Hybrid Algorithms | 241 |
| 86.3.6 Firefly Algorithm Based Memetic Algorithm | 241 |
| 86.3.7 Parallel Firefly Algorithm with Predation (pFAP) | 241 |
| 86.3.8 Modified Firefly Algorithm | 241 |
| 86.4 Applications | 241 |
| 86.4.1 Digital Image Compression and Image Processing | 241 |
| 86.4.2 Eigenvalue optimization | 241 |
| 86.4.3 Nanoelectronic Integrated Circuit and System Design | 242 |
| 86.4.4 Feature selection and fault detection | 242 |
| 86.4.5 Antenna Design | 242 |
| 86.4.6 Structural Design | 242 |
| 86.4.7 Scheduling and TSP | 242 |
| 86.4.8 Semantic Web Composition | 242 |
| 86.4.9 Chemical Phase equilibrium | 242 |
| 86.4.10 Clustering | 242 |
| 86.4.11 Dynamic Problems | 242 |
| 86.4.12 Rigid Image Registration Problems | 242 |
| 86.4.13 Protein Structure Prediction | 242 |
| 86.4.14 Parameter Optimization of SVM | 242 |
| 86.4.15 IK-FA, Solving Inverse Kinematics using FA | 243 |
| 86.5 See also | 243 |
| 86.6 References | 243 |
| 86.7 External links | 245 |
| 87 Fourier–Motzkin elimination | 246 |
| 87.1 Elimination | 246 |
| 87.2 Complexity | 246 |
| 87.3 See also | 246 |
| 87.4 References | 246 |
| 87.5 External links | 247 |
| 88 Generalized iterative scaling | 248 |
| 88.1 See also | 248 |
| 88.2 References | 248 |
| 89 Genetic algorithm | 249 |

| | |
|--|------------|
| 89.1 Methodology | 249 |
| 89.1.1 Initialization | 250 |
| 89.1.2 Selection | 250 |
| 89.1.3 Genetic operators | 250 |
| 89.1.4 Termination | 250 |
| 89.2 The building block hypothesis | 251 |
| 89.3 Limitations | 251 |
| 89.4 Variants | 252 |
| 89.4.1 Chromosome representation | 252 |
| 89.4.2 Elitism | 253 |
| 89.4.3 Parallel implementations | 253 |
| 89.4.4 Adaptive GAs | 253 |
| 89.5 Problem domains | 253 |
| 89.6 History | 254 |
| 89.7 Related techniques | 254 |
| 89.7.1 Parent fields | 254 |
| 89.7.2 Related fields | 254 |
| 89.8 See also | 256 |
| 89.9 References | 256 |
| 89.10 Bibliography | 258 |
| 89.11 External links | 259 |
| 89.11.1 Resources | 259 |
| 89.11.2 Tutorials | 259 |
| 90 Genetic algorithms in economics | 260 |
| 90.1 Genetic algorithm in the cobweb model | 260 |
| 90.2 References | 260 |
| 90.3 External links | 260 |
| 91 Glowworm swarm optimization | 261 |
| 91.1 See also | 261 |
| 91.2 References | 261 |
| 91.3 External links | 261 |
| 92 Gradient method | 262 |
| 92.1 See also | 262 |
| 93 Graduated optimization | 263 |
| 93.1 Technique description | 263 |
| 93.2 Some examples | 263 |
| 93.3 Related optimization techniques | 264 |
| 93.4 See also | 264 |
| 93.5 References | 264 |

| | |
|---|------------|
| 94 Great Deluge algorithm | 265 |
| 94.1 References | 265 |
| 94.2 See also | 265 |
| 95 Guided Local Search | 266 |
| 95.1 Overview | 266 |
| 95.1.1 Solution features | 266 |
| 95.1.2 Selective penalty modifications | 266 |
| 95.1.3 Searching through an augmented cost function | 266 |
| 95.1.4 Extensions of Guided Local Search | 267 |
| 95.2 Related work | 267 |
| 95.3 Bibliography | 267 |
| 95.4 External links | 268 |
| 96 Harmony search | 269 |
| 96.1 Basic harmony search algorithm | 269 |
| 96.2 Other related algorithms | 270 |
| 96.3 Criticism | 270 |
| 96.4 Notes | 270 |
| 96.5 References | 270 |
| 96.5.1 General information | 271 |
| 96.5.2 Theory of harmony search | 271 |
| 96.5.3 Applications in computer science | 271 |
| 96.5.4 Applications in engineering | 272 |
| 96.5.5 Applications in economics | 273 |
| 96.6 Source codes | 273 |
| 97 Imperialist competitive algorithm | 274 |
| 97.1 Algorithm | 274 |
| 97.2 Pseudocode | 274 |
| 97.3 Variants | 274 |
| 97.4 Applications | 275 |
| 97.5 References | 275 |
| 98 Intelligent Water Drops algorithm | 277 |
| 98.1 Introduction | 277 |
| 98.2 Pseudo-code | 277 |
| 98.3 Applications | 277 |
| 98.4 See also | 278 |
| 98.5 References | 278 |
| 98.6 External links | 278 |
| 99 Interior point method | 279 |

| | |
|---|------------|
| 99.1 Primal-dual interior point method for nonlinear optimization | 279 |
| 99.2 See also | 280 |
| 99.3 References | 280 |
| 99.4 Bibliography | 280 |
| 100 Interval contractor | 282 |
| 100.1 Properties of contractors | 282 |
| 100.2 Illustration | 282 |
| 100.3 Contractor algebra | 282 |
| 100.4 Building contractors | 283 |
| 100.5 References | 283 |
| 101 IOSO | 284 |
| 101.1 IOSO approach | 284 |
| 101.2 History | 284 |
| 101.3 Products | 284 |
| 101.4 Purpose | 284 |
| 101.4.1 Performance improvement and design optimisation | 284 |
| 101.4.2 Search for optimal system management laws | 284 |
| 101.4.3 Identification of mathematical models | 285 |
| 101.5 Robust design optimization and robust optimal control | 285 |
| 101.5.1 Introduction | 285 |
| 101.5.2 IOSO robust design optimization concept | 285 |
| 101.6 References | 285 |
| 101.7 External links | 285 |
| 102 IPOPT | 287 |
| 102.1 External links | 287 |
| 103 Iterated local search | 288 |
| 103.1 References | 288 |
| 104 Job shop scheduling | 289 |
| 104.1 Problem variations | 289 |
| 104.2 NP-hardness | 289 |
| 104.3 Problem representation | 289 |
| 104.4 The problem of infinite cost | 290 |
| 104.5 Major results | 290 |
| 104.6 Offline makespan minimization | 290 |
| 104.6.1 Atomic jobs | 290 |
| 104.6.2 Jobs consisting of multiple operations | 290 |
| 104.7 Example | 291 |
| 104.8 See also | 291 |

| | |
|--|------------|
| 104.9References | 291 |
| 104.1External links | 292 |
| 105Kantorovich theorem | 293 |
| 105.1Assumptions | 293 |
| 105.2Statement | 293 |
| 105.3Notes | 294 |
| 105.4References | 294 |
| 105.5Literature | 294 |
| 106Killer heuristic | 295 |
| 106.1External links | 295 |
| 107LINCOA | 296 |
| 107.1See also | 296 |
| 107.2References | 296 |
| 107.3External links | 296 |
| 108Local convergence | 297 |
| 109Luus–Jaakola | 298 |
| 109.1Motivation | 298 |
| 109.2Heuristic | 298 |
| 109.3Convergence | 299 |
| 109.4See also | 299 |
| 109.5References | 299 |
| 110Matrix chain multiplication | 300 |
| 110.1A Dynamic Programming Algorithm | 300 |
| 110.2More Efficient Algorithms | 301 |
| 110.2.1 Hu & Shing (1981) | 301 |
| 110.3Generalizations | 302 |
| 110.4References | 302 |
| 111Maximum subarray problem | 303 |
| 111.1Kadane’s algorithm | 303 |
| 111.2Generalizations | 303 |
| 111.3See also | 303 |
| 111.4References | 303 |
| 111.5External links | 303 |
| 112MCS algorithm | 305 |
| 112.1External links | 305 |
| 113Mehrotra predictor–corrector method | 306 |

| | |
|--|------------|
| 113.1 References | 306 |
| 114 Meta-optimization | 307 |
| 114.1 Motivation | 307 |
| 114.2 Methods | 307 |
| 114.3 See also | 308 |
| 114.4 References | 308 |
| 115 Minimax | 309 |
| 115.1 Game theory | 309 |
| 115.1.1 Minimax theorem | 309 |
| 115.1.2 Example | 309 |
| 115.1.3 Maximin | 310 |
| 115.2 Combinatorial game theory | 310 |
| 115.2.1 Minimax algorithm with alternate moves | 310 |
| 115.2.2 Pseudocode | 310 |
| 115.2.3 Example | 311 |
| 115.3 Minimax for individual decisions | 311 |
| 115.3.1 Minimax in the face of uncertainty | 311 |
| 115.3.2 Minimax criterion in statistical decision theory | 311 |
| 115.3.3 Non-probabilistic decision theory | 312 |
| 115.4 Maximin in philosophy | 312 |
| 115.5 See also | 312 |
| 115.6 Notes | 312 |
| 115.7 External links | 312 |
| 116 MM algorithm | 314 |
| 116.1 History | 314 |
| 116.2 How it works | 314 |
| 116.3 Ways to construct surrogate functions | 314 |
| 116.4 References | 315 |
| 117 Multi-swarm optimization | 316 |
| 117.1 Description | 316 |
| 117.2 Current Work | 316 |
| 117.3 See also | 316 |
| 117.4 References | 316 |
| 118 Natural evolution strategy | 318 |
| 118.1 Method | 318 |
| 118.1.1 Search gradients | 318 |
| 118.1.2 Natural gradient ascent | 319 |
| 118.1.3 Fitness shaping | 319 |

| | |
|--|------------|
| 118.1.4 Pseudocode | 319 |
| 118.2 See also | 319 |
| 118.3 Bibliography | 319 |
| 118.4 External links | 319 |
| 119 Negamax | 320 |
| 119.1 Negamax Base Algorithm | 320 |
| 119.2 Negamax with Alpha Beta Pruning | 321 |
| 119.3 Negamax with Alpha Beta Pruning and Transposition Tables | 321 |
| 119.4 References | 322 |
| 119.5 External links | 322 |
| 120 Newton's method | 323 |
| 120.1 Description | 323 |
| 120.2 History | 324 |
| 120.3 Practical considerations | 324 |
| 120.3.1 Difficulty in calculating derivative of a function | 324 |
| 120.3.2 Failure of the method to converge to the root | 324 |
| 120.3.3 Slow convergence for roots of multiplicity > 1 | 325 |
| 120.4 Analysis | 325 |
| 120.4.1 Proof of quadratic convergence for Newton's iterative method | 325 |
| 120.4.2 Basins of attraction | 326 |
| 120.5 Failure analysis | 326 |
| 120.5.1 Bad starting points | 326 |
| 120.5.2 Derivative issues | 327 |
| 120.5.3 Non-quadratic convergence | 328 |
| 120.6 Generalizations | 328 |
| 120.6.1 Complex functions | 328 |
| 120.6.2 Nonlinear systems of equations | 329 |
| 120.6.3 Nonlinear equations in a Banach space | 329 |
| 120.6.4 Nonlinear equations over p -adic numbers | 329 |
| 120.6.5 Newton-Fourier method | 329 |
| 120.6.6 Quasi-Newton methods | 329 |
| 120.7 Applications | 329 |
| 120.7.1 Minimization and maximization problems | 329 |
| 120.7.2 Multiplicative inverses of numbers and power series | 330 |
| 120.7.3 Solving transcendental equations | 330 |
| 120.8 Examples | 330 |
| 120.8.1 Square root of a number | 330 |
| 120.8.2 Solution of $\cos(x) = x^3$ | 330 |
| 120.9 Pseudocode | 331 |
| 120.10 See also | 331 |

| | |
|---|------------|
| 120.1 References | 331 |
| 120.1 External links | 332 |
| 121 NEWUOA | 333 |
| 121.1 See also | 333 |
| 121.2 References | 333 |
| 121.3 External links | 334 |
| 122 Nonlinear programming | 335 |
| 122.1 Applicability | 335 |
| 122.2 The general non-linear optimization problem (NLP) | 335 |
| 122.3 Possible solutions | 335 |
| 122.4 Methods for solving the problem | 335 |
| 122.5 Examples | 336 |
| 122.5.1 2-dimensional example | 336 |
| 122.5.2 3-dimensional example | 336 |
| 122.6 Applications | 336 |
| 122.7 See also | 337 |
| 122.8 References | 337 |
| 122.9 Further reading | 337 |
| 122.10 External links | 337 |
| 123 Ordered subset expectation maximization | 338 |
| 123.1 References | 338 |
| 123.2 External links | 338 |
| 124 Parallel metaheuristic | 339 |
| 124.1 Background | 339 |
| 124.2 Parallel trajectory-based metaheuristics | 340 |
| 124.3 Parallel population-based metaheuristics | 340 |
| 124.4 See also | 341 |
| 124.5 References | 341 |
| 124.6 External links | 341 |
| 125 Particle swarm optimization | 342 |
| 125.1 Algorithm | 342 |
| 125.2 Parameter selection | 343 |
| 125.3 Neighborhoods and Topologies | 343 |
| 125.4 Inner workings | 343 |
| 125.4.1 Convergence | 343 |
| 125.4.2 Biases | 344 |
| 125.5 Variants | 344 |
| 125.5.1 Simplifications | 344 |

| | |
|---|------------|
| 125.5.2 Multi-objective optimization | 344 |
| 125.5.3 Binary, Discrete, and Combinatorial PSO | 345 |
| 125.6 See also | 345 |
| 125.7 References | 345 |
| 125.8 External links | 347 |
| 126 Pattern search (optimization) | 348 |
| 126.1 Convergence | 348 |
| 126.2 See also | 348 |
| 126.3 References | 349 |
| 127 pSeven | 350 |
| 127.1 History | 350 |
| 127.2 Functionality | 350 |
| 127.2.1 Process integration | 350 |
| 127.2.2 Design Space Exploration | 351 |
| 127.2.3 Visualization and postprocessing | 353 |
| 127.3 Application areas and customers | 353 |
| 127.4 Modules and Packs | 354 |
| 127.5 References | 354 |
| 127.6 External links | 354 |
| 128 Quantum annealing | 355 |
| 128.1 Comparison to simulated annealing | 355 |
| 128.2 Quantum mechanics: Analogy & advantage | 355 |
| 128.3 Implementations | 356 |
| 128.4 References | 356 |
| 128.5 General review articles and books | 357 |
| 129 Quasi-Newton method | 358 |
| 129.1 Description of the method | 358 |
| 129.1.1 Search for zeroes | 358 |
| 129.1.2 Search for extrema | 358 |
| 129.2 Implementations | 359 |
| 129.3 See also | 360 |
| 129.4 References | 360 |
| 129.5 Further reading | 360 |
| 130 Random optimization | 361 |
| 130.1 Algorithm | 361 |
| 130.2 Convergence and variants | 361 |
| 130.3 See also | 361 |
| 130.4 References | 361 |

| | |
|---|------------|
| 131 Random search | 363 |
| 131.1 Algorithm | 363 |
| 131.2 Variants | 363 |
| 131.3 See also | 363 |
| 131.4 References | 363 |
| 132 Rosenbrock methods | 365 |
| 132.1 Numerical solution of differential equations | 365 |
| 132.2 Search method | 365 |
| 132.3 See also | 365 |
| 132.4 References | 365 |
| 132.5 External links | 365 |
| 133 Search-based software engineering | 366 |
| 133.1 Definition | 366 |
| 133.2 Brief history | 366 |
| 133.3 Application areas | 366 |
| 133.3.1 Requirements engineering | 366 |
| 133.3.2 Debugging and maintenance | 366 |
| 133.3.3 Testing | 367 |
| 133.3.4 Optimizing software | 367 |
| 133.3.5 Project management | 367 |
| 133.4 Tools | 367 |
| 133.5 Methods and techniques | 367 |
| 133.6 Industry acceptance | 367 |
| 133.7 See also | 367 |
| 133.8 References | 367 |
| 133.9 External links | 368 |
| 134 Sequence-dependent setup | 370 |
| 134.1 See also | 370 |
| 135 Sequential minimal optimization | 371 |
| 135.1 Optimization problem | 371 |
| 135.2 Algorithm | 371 |
| 135.3 See also | 371 |
| 135.4 References | 371 |
| 136 Shuffled frog leaping algorithm | 373 |
| 136.1 References | 373 |
| 137 Simultaneous perturbation stochastic approximation | 374 |
| 137.1 Convergence lemma | 374 |
| 137.2 Sketch of the proof | 374 |

| | |
|---|------------|
| 137.3 References | 375 |
| 138 Social cognitive optimization | 376 |
| 138.1 Algorithm | 376 |
| 138.2 References | 377 |
| 139 Space allocation problem | 378 |
| 140 Space mapping | 379 |
| 140.1 Concept | 379 |
| 140.2 Development | 379 |
| 140.3 Category | 379 |
| 140.4 Terminology | 379 |
| 140.5 Methodology | 379 |
| 140.6 Applications | 380 |
| 140.7 Simulators | 380 |
| 140.8 Nonlinear Device Modeling | 380 |
| 140.9 Conferences | 380 |
| 140.10 See also | 380 |
| 140.11 References | 380 |
| 141 Special ordered set | 382 |
| 141.1 Context of Applications | 382 |
| 141.2 History | 382 |
| 141.3 Types of SOS | 382 |
| 141.4 Further Example | 382 |
| 141.5 Notes | 382 |
| 141.6 References | 383 |
| 142 Stochastic hill climbing | 384 |
| 142.1 See also | 384 |
| 142.2 References | 384 |
| 143 Swarm intelligence | 385 |
| 143.1 Example algorithms | 385 |
| 143.1.1 Particle swarm optimization | 385 |
| 143.1.2 Ant colony optimization | 385 |
| 143.1.3 Ant Lion Optimizer | 385 |
| 143.1.4 Artificial bee colony algorithm | 385 |
| 143.1.5 Bacterial colony optimization | 386 |
| 143.1.6 Differential evolution | 386 |
| 143.1.7 The bees algorithm | 386 |
| 143.1.8 Artificial immune systems | 386 |
| 143.1.9 Grey wolf optimizer | 386 |

| | |
|--|------------|
| 143.1.1 Bat algorithm | 386 |
| 143.1.1 Multi-Verse Optimizer | 386 |
| 143.1.1 Gravitational search algorithm | 386 |
| 143.1.1 Altruism algorithm | 387 |
| 143.1.1 Glowworm swarm optimization | 387 |
| 143.1.1 River Formation Dynamics | 387 |
| 143.1.1 Self-propelled particles | 387 |
| 143.1.1 Stochastic diffusion search | 387 |
| 143.1.1 Multi-swarm optimization | 387 |
| 143.2 Applications | 388 |
| 143.2.1 Ant-based routing | 388 |
| 143.2.2 Crowd simulation | 388 |
| 143.2.3 Swarmic art | 388 |
| 143.3 In popular culture | 389 |
| 143.4 Notable researchers | 390 |
| 143.5 See also | 390 |
| 143.6 References | 390 |
| 143.7 External links | 392 |
| 143.8 Further reading | 392 |
| 144 TOLMIN (optimization software) | 394 |
| 144.1 See also | 394 |
| 144.2 References | 394 |
| 144.3 External links | 394 |
| 145 Tree rearrangement | 395 |
| 145.1 Basic tree rearrangements | 395 |
| 145.2 Tree fusion | 395 |
| 145.3 Sectorial search | 395 |
| 145.4 Tree drifting | 395 |
| 145.5 Tree fusing | 395 |
| 145.6 References | 395 |
| 146 UOBYQA | 397 |
| 146.1 See also | 397 |
| 146.2 References | 397 |
| 146.3 External links | 397 |
| 147 Very large-scale neighborhood search | 398 |
| 147.1 References | 398 |
| 148 Zions–Wallenius method | 399 |
| 148.1 Detail | 399 |

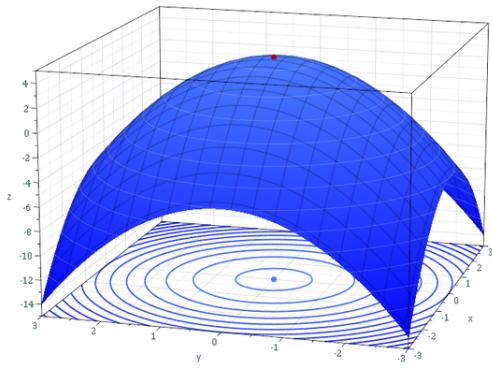
| | |
|--|-----|
| 148.2 References | 399 |
| 148.3 Text and image sources, contributors, and licenses | 400 |
| 148.3.1 Text | 400 |
| 148.3.2 Images | 412 |
| 148.3.3 Content license | 418 |

Chapter 1

Mathematical optimization

“Optimization” and “Optimum” redirect here. For other uses, see Optimization (disambiguation) and Optimum (disambiguation).

In mathematics, computer science, economics, or



Graph of a paraboloid given by $f(x, y) = -(x^2 + y^2) + 4$. The global maximum at $(0, 0, 4)$ is indicated by a red dot.

management science, **mathematical optimization** (alternatively, **optimization** or **mathematical programming**) is the selection of a best element (with regard to some criteria) from some set of available alternatives.^[1]

In the simplest case, an optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function. The generalization of optimization theory and techniques to other formulations comprises a large area of applied mathematics. More generally, optimization includes finding “best available” values of some objective function given a defined domain (or a set of constraints), including a variety of different types of objective functions and different types of domains.

1.1 Optimization problems

Main article: Optimization problem

An optimization problem can be represented in the fol-

lowing way:

Given: a function $f : A \rightarrow \mathbf{R}$ from some set A to the real numbers

Sought: an element x_0 in A such that $f(x_0) \leq f(x)$ for all x in A (“minimization”) or such that $f(x_0) \geq f(x)$ for all x in A (“maximization”).

Such a formulation is called an **optimization problem** or a **mathematical programming problem** (a term not directly related to computer programming, but still in use for example in linear programming – see History below). Many real-world and theoretical problems may be modeled in this general framework. Problems formulated using this technique in the fields of physics and computer vision may refer to the technique as **energy minimization**, speaking of the value of the function f as representing the energy of the system being modeled.

Typically, A is some subset of the Euclidean space \mathbf{R}^n , often specified by a set of **constraints**, equalities or inequalities that the members of A have to satisfy. The domain A of f is called the **search space** or the **choice set**, while the elements of A are called **candidate solutions** or **feasible solutions**.

The function f is called, variously, an **objective function**, a **loss function** or **cost function** (minimization),^[2] a **utility function** or **fitness function** (maximization), or, in certain fields, an **energy function** or **energy functional**. A feasible solution that minimizes (or maximizes, if that is the goal) the objective function is called an *optimal solution*.

In mathematics, by convention optimization problems are usually stated in terms of minimization. Generally, unless both the objective function and the feasible region are **convex** in a minimization problem, there may be several local minima, where a *local minimum* x^* is defined as a point for which there exists some $\delta > 0$ so that for all x such that

$$\|\mathbf{x} - \mathbf{x}^*\| \leq \delta;$$

the expression

$$f(\mathbf{x}^*) \leq f(\mathbf{x})$$

holds; that is to say, on some region around \mathbf{x}^* all of the function values are greater than or equal to the value at that point. Local maxima are defined similarly.

A large number of algorithms proposed for solving non-convex problems – including the majority of commercially available solvers – are not capable of making a distinction between local optimal solutions and rigorous optimal solutions, and will treat the former as actual solutions to the original problem. The branch of applied mathematics and numerical analysis that is concerned with the development of deterministic algorithms that are capable of guaranteeing convergence in finite time to the actual optimal solution of a non-convex problem is called global optimization.

1.2 Notation

Optimization problems are often expressed with special notation. Here are some examples.

1.2.1 Minimum and maximum value of a function

Consider the following notation:

$$\min_{x \in \mathbb{R}} (x^2 + 1)$$

This denotes the minimum value of the objective function $x^2 + 1$, when choosing x from the set of real numbers \mathbb{R} . The minimum value in this case is 1, occurring at $x = 0$.

Similarly, the notation

$$\max_{x \in \mathbb{R}} 2x$$

asks for the maximum value of the objective function $2x$, where x may be any real number. In this case, there is no such maximum as the objective function is unbounded, so the answer is "infinity" or "undefined".

1.2.2 Optimal input arguments

Main article: Arg max

Consider the following notation:

$$\arg \min_{x \in (-\infty, -1]} x^2 + 1,$$

or equivalently

$$\arg \min_x x^2 + 1, \text{ to: subject } x \in (-\infty, -1].$$

This represents the value (or values) of the argument x in the interval $(-\infty, -1]$ that minimizes (or minimize) the objective function $x^2 + 1$ (the actual minimum value of that function is not what the problem asks for). In this case, the answer is $x = -1$, since $x = 0$ is infeasible, i.e. does not belong to the feasible set.

Similarly,

$$\arg \max_{x \in [-5, 5], y \in \mathbb{R}} x \cos(y),$$

or equivalently

$$\arg \max_{x, y} x \cos(y), \text{ to: subject } x \in [-5, 5], y \in \mathbb{R},$$

represents the (x, y) pair (or pairs) that maximizes (or maximize) the value of the objective function $x \cos(y)$, with the added constraint that x lie in the interval $[-5, 5]$ (again, the actual maximum value of the expression does not matter). In this case, the solutions are the pairs of the form $(5, 2k\pi)$ and $(-5, (2k+1)\pi)$, where k ranges over all integers.

arg min and **arg max** are sometimes also written **argmin** and **argmax**, and stand for **argument of the minimum** and **argument of the maximum**.

1.3 History

Fermat and Lagrange found calculus-based formulas for identifying optima, while Newton and Gauss proposed iterative methods for moving towards an optimum.

The term "linear programming" for certain optimization cases was due to George B. Dantzig, although much of the theory had been introduced by Leonid Kantorovich in 1939. (Programming in this context does not refer to computer programming, but from the use of *program* by the United States military to refer to proposed training and logistics schedules, which were the problems Dantzig studied at that time.) Dantzig published the Simplex algorithm in 1947, and John von Neumann developed the theory of duality in the same year.

Other major researchers in mathematical optimization include the following:

1.4 Major subfields

- Convex programming studies the case when the objective function is convex (minimization) or concave

(maximization) and the constraint set is convex. This can be viewed as a particular case of nonlinear programming or as generalization of linear or convex quadratic programming.

- Linear programming (LP), a type of convex programming, studies the case in which the objective function f is linear and the set of constraints is specified using only linear equalities and inequalities. Such a set is called a polyhedron or a polytope if it is bounded.
- Second order cone programming (SOCP) is a convex program, and includes certain types of quadratic programs.
- Semidefinite programming (SDP) is a subfield of convex optimization where the underlying variables are semidefinite matrices. It is generalization of linear and convex quadratic programming.
- Conic programming is a general form of convex programming. LP, SOCP and SDP can all be viewed as conic programs with the appropriate type of cone.
- Geometric programming is a technique whereby objective and inequality constraints expressed as posynomials and equality constraints as monomials can be transformed into a convex program.
- Integer programming studies linear programs in which some or all variables are constrained to take on integer values. This is not convex, and in general much more difficult than regular linear programming.
- Quadratic programming allows the objective function to have quadratic terms, while the feasible set must be specified with linear equalities and inequalities. For specific forms of the quadratic term, this is a type of convex programming.
- Fractional programming studies optimization of ratios of two nonlinear functions. The special class of concave fractional programs can be transformed to a convex optimization problem.
- Nonlinear programming studies the general case in which the objective function or the constraints or both contain nonlinear parts. This may or may not be a convex program. In general, whether the program is convex affects the difficulty of solving it.
- Stochastic programming studies the case in which some of the constraints or parameters depend on random variables.
- Robust programming is, like stochastic programming, an attempt to capture uncertainty in the data underlying the optimization problem. Robust optimization targets to find solutions that are valid under all possible realizations of the uncertainties.
- Combinatorial optimization is concerned with problems where the set of feasible solutions is discrete or can be reduced to a discrete one.
- Stochastic optimization for use with random (noisy) function measurements or random inputs in the search process.
- Infinite-dimensional optimization studies the case when the set of feasible solutions is a subset of an infinite-dimensional space, such as a space of functions.
- Heuristics and metaheuristics make few or no assumptions about the problem being optimized. Usually, heuristics do not guarantee that any optimal solution need be found. On the other hand, heuristics are used to find approximate solutions for many complicated optimization problems.
- Constraint satisfaction studies the case in which the objective function f is constant (this is used in artificial intelligence, particularly in automated reasoning).
 - Constraint programming.
- Disjunctive programming is used where at least one constraint must be satisfied but not all. It is of particular use in scheduling.

In a number of subfields, the techniques are designed primarily for optimization in dynamic contexts (that is, decision making over time):

- Calculus of variations seeks to optimize an objective defined over many points in time, by considering how the objective function changes if there is a small change in the choice path.
- Optimal control theory is a generalization of the calculus of variations.
- Dynamic programming studies the case in which the optimization strategy is based on splitting the problem into smaller subproblems. The equation that describes the relationship between these subproblems is called the Bellman equation.
- Mathematical programming with equilibrium constraints is where the constraints include variational inequalities or complementarities.

1.4.1 Multi-objective optimization

Main article: [Multi-objective optimization](#)

Adding more than one objective to an optimization problem adds complexity. For example, to optimize a structural design, one would want a design that is both light

and rigid. Because these two objectives conflict, a trade-off exists. There will be one lightest design, one stiffest design, and an infinite number of designs that are some compromise of weight and stiffness. The set of trade-off designs that cannot be improved upon according to one criterion without hurting another criterion is known as the Pareto set. The curve created plotting weight against stiffness of the best designs is known as the Pareto frontier.

A design is judged to be “Pareto optimal” (equivalently, “Pareto efficient” or in the Pareto set) if it is not dominated by any other design: If it is worse than another design in some respects and no better in any respect, then it is dominated and is not Pareto optimal.

The choice among “Pareto optimal” solutions to determine the “favorite solution” is delegated to the decision maker. In other words, defining the problem as multi-objective optimization signals that some information is missing: desirable objectives are given but not their detailed combination. In some cases, the missing information can be derived by interactive sessions with the decision maker.

Multi-objective optimization problems have been generalized further to **vector optimization** problems where the (partial) ordering is no longer given by the Pareto ordering.

1.4.2 Multi-modal optimization

Optimization problems are often multi-modal; that is, they possess multiple good solutions. They could all be globally good (same cost function value) or there could be a mix of globally good and locally good solutions. Obtaining all (or at least some of) the multiple solutions is the goal of a multi-modal optimizer.

Classical optimization techniques due to their iterative approach do not perform satisfactorily when they are used to obtain multiple solutions, since it is not guaranteed that different solutions will be obtained even with different starting points in multiple runs of the algorithm. Evolutionary algorithms are however a very popular approach to obtain multiple solutions in a multi-modal optimization task.

1.5 Classification of critical points and extrema

1.5.1 Feasibility problem

The **satisfiability problem**, also called the **feasibility problem**, is just the problem of finding any **feasible solution** at all without regard to objective value. This can be regarded as the special case of mathematical optimization

where the objective value is the same for every solution, and thus any solution is optimal.

Many optimization algorithms need to start from a feasible point. One way to obtain such a point is to **relax** the feasibility conditions using a **slack variable**; with enough slack, any starting point is feasible. Then, minimize that slack variable until slack is null or negative.

1.5.2 Existence

The **extreme value theorem** of Karl Weierstrass states that a continuous real-valued function on a compact set attains its maximum and minimum value. More generally, a lower semi-continuous function on a compact set attains its minimum; an upper semi-continuous function on a compact set attains its maximum.

1.5.3 Necessary conditions for optimality

One of Fermat's theorems states that optima of unconstrained problems are found at stationary points, where the first derivative or the gradient of the objective function is zero (see **first derivative test**). More generally, they may be found at **critical points**, where the first derivative or gradient of the objective function is zero or is undefined, or on the boundary of the choice set. An equation (or set of equations) stating that the first derivative(s) equal(s) zero at an interior optimum is called a 'first-order condition' or a set of first-order conditions.

Optima of equality-constrained problems can be found by the **Lagrange multiplier** method. The optima of problems with equality and/or inequality constraints can be found using the '**Karush–Kuhn–Tucker conditions**'.

1.5.4 Sufficient conditions for optimality

While the first derivative test identifies points that might be extrema, this test does not distinguish a point that is a minimum from one that is a maximum or one that is neither. When the objective function is twice differentiable, these cases can be distinguished by checking the second derivative or the matrix of second derivatives (called the **Hessian matrix**) in unconstrained problems, or the matrix of second derivatives of the objective function and the constraints called the **bordered Hessian** in constrained problems. The conditions that distinguish maxima, or minima, from other stationary points are called 'second-order conditions' (see '**Second derivative test**'). If a candidate solution satisfies the first-order conditions, then satisfaction of the second-order conditions as well is sufficient to establish at least local optimality.

1.5.5 Sensitivity and continuity of optima

The envelope theorem describes how the value of an optimal solution changes when an underlying parameter changes. The process of computing this change is called comparative statics.

The maximum theorem of Claude Berge (1963) describes the continuity of an optimal solution as a function of underlying parameters.

1.5.6 Calculus of optimization

Main article: Karush–Kuhn–Tucker conditions

See also: Critical point (mathematics), Differential calculus, Gradient, Hessian matrix, Positive definite matrix, Lipschitz continuity, Rademacher's theorem, Convex function and Convex analysis

For unconstrained problems with twice-differentiable functions, some critical points can be found by finding the points where the gradient of the objective function is zero (that is, the stationary points). More generally, a zero subgradient certifies that a local minimum has been found for minimization problems with convex functions and other locally Lipschitz functions.

Further, critical points can be classified using the definiteness of the Hessian matrix: If the Hessian is *positive* definite at a critical point, then the point is a local minimum; if the Hessian matrix is negative definite, then the point is a local maximum; finally, if indefinite, then the point is some kind of saddle point.

Constrained problems can often be transformed into unconstrained problems with the help of Lagrange multipliers. Lagrangian relaxation can also provide approximate solutions to difficult constrained problems.

When the objective function is convex, then any local minimum will also be a global minimum. There exist efficient numerical techniques for minimizing convex functions, such as interior-point methods.

1.6 Computational optimization techniques

To solve problems, researchers may use algorithms that terminate in a finite number of steps, or iterative methods that converge to a solution (on some specified class of problems), or heuristics that may provide approximate solutions to some problems (although their iterates need not converge).

1.6.1 Optimization algorithms

- Simplex algorithm of George Dantzig, designed for linear programming.
- Extensions of the simplex algorithm, designed for quadratic programming and for linear-fractional programming.
- Variants of the simplex algorithm that are especially suited for network optimization.
- Combinatorial algorithms

1.6.2 Iterative methods

Main article: Iterative method

See also: Newton's method, Quasi-Newton method, Finite difference, Approximation theory and Numerical analysis

The iterative methods used to solve problems of nonlinear programming differ according to whether they evaluate Hessians, gradients, or only function values. While evaluating Hessians (H) and gradients (G) improves the rate of convergence, for functions for which these quantities exist and vary sufficiently smoothly, such evaluations increase the computational complexity (or computational cost) of each iteration. In some cases, the computational complexity may be excessively high.

One major criterion for optimizers is just the number of required function evaluations as this often is already a large computational effort, usually much more effort than within the optimizer itself, which mainly has to operate over the N variables. The derivatives provide detailed information for such optimizers, but are even harder to calculate, e.g. approximating the gradient takes at least $N+1$ function evaluations. For approximations of the 2nd derivatives (collected in the Hessian matrix) the number of function evaluations is in the order of N^2 . Newton's method requires the 2nd order derivates, so for each iteration the number of function calls is in the order of N^2 , but for a simpler pure gradient optimizer it is only N . However, gradient optimizers need usually more iterations than Newton's algorithm. Which one is best with respect to the number of function calls depends on the problem itself.

- Methods that evaluate Hessians (or approximate Hessians, using finite differences):
 - Newton's method
 - Sequential quadratic programming: A Newton-based method for small-medium scale *constrained* problems. Some versions can handle large-dimensional problems.

- Methods that evaluate gradients or approximate gradients using finite differences (or even subgradients):
 - Quasi-Newton methods: Iterative methods for medium-large problems (e.g. $N < 1000$).
 - Conjugate gradient methods: Iterative methods for large problems. (In theory, these methods terminate in a finite number of steps with quadratic objective functions, but this finite termination is not observed in practice on finite-precision computers.)
 - Interior point methods: This is a large class of methods for constrained optimization. Some interior-point methods use only (sub)gradient information, and others of which require the evaluation of Hessians.
 - Gradient descent (alternatively, “steepest descent” or “steepest ascent”): A (slow) method of historical and theoretical interest, which has had renewed interest for finding approximate solutions of enormous problems.
 - Subgradient methods - An iterative method for large locally Lipschitz functions using generalized gradients. Following Boris T. Polyak, subgradient-projection methods are similar to conjugate-gradient methods.
 - Bundle method of descent: An iterative method for small-medium-sized problems with locally Lipschitz functions, particularly for convex minimization problems. (Similar to conjugate gradient methods)
 - Ellipsoid method: An iterative method for small problems with quasiconvex objective functions and of great theoretical interest, particularly in establishing the polynomial time complexity of some combinatorial optimization problems. It has similarities with Quasi-Newton methods.
 - Reduced gradient method (Frank-Wolfe) for approximate minimization of specially structured problems with linear constraints, especially with traffic networks. For general unconstrained problems, this method reduces to the gradient method, which is regarded as obsolete (for almost all problems).
 - Simultaneous perturbation stochastic approximation (SPSA) method for stochastic optimization; uses random (efficient) gradient approximation.
 - Methods that evaluate only function values: If a problem is continuously differentiable, then gradients can be approximated using finite differences, in which case a gradient-based method can be used.
 - Interpolation methods
 - Pattern search methods, which have better convergence properties than the Nelder-Mead heuristic (with simplices), which is listed below.

1.6.3 Global convergence

More generally, if the objective function is not a quadratic function, then many optimization methods use other methods to ensure that some subsequence of iterations converges to an optimal solution. The first and still popular method for ensuring convergence relies on line searches, which optimize a function along one dimension. A second and increasingly popular method for ensuring convergence uses trust regions. Both line searches and trust regions are used in modern methods of non-differentiable optimization. Usually a global optimizer is much slower than advanced local optimizers (such as BFGS), so often an efficient global optimizer can be constructed by starting the local optimizer from different starting points.

1.6.4 Heuristics

Main article: [Heuristic algorithm](#)

Besides (finitely terminating) algorithms and (convergent) iterative methods, there are heuristics that can provide approximate solutions to some optimization problems:

- Memetic algorithm
- Differential evolution
- Evolutionary algorithms
- Dynamic relaxation
- Genetic algorithms
- Hill climbing with random restart
- Nelder-Mead simplicial heuristic: A popular heuristic for approximate minimization (without calling gradients)
- Particle swarm optimization
- Artificial bee colony optimization
- Simulated annealing
- Tabu search
- Reactive Search Optimization (RSO)^[3] implemented in LIONsolver

1.7 Applications

1.7.1 Mechanics and engineering

Problems in rigid body dynamics (in particular articulated rigid body dynamics) often require mathematical programming techniques, since you can view rigid body dynamics as attempting to solve an ordinary differential equation on a constraint manifold; the constraints are various nonlinear geometric constraints such as “these two points must always coincide”, “this surface must not penetrate any other”, or “this point must always lie somewhere on this curve”. Also, the problem of computing contact forces can be done by solving a linear complementarity problem, which can also be viewed as a QP (quadratic programming) problem.

Many design problems can also be expressed as optimization programs. This application is called design optimization. One subset is the engineering optimization, and another recent and growing subset of this field is multidisciplinary design optimization, which, while useful in many problems, has in particular been applied to aerospace engineering problems.

1.7.2 Economics

Economics is closely enough linked to optimization of agents that an influential definition relatedly describes economics *qua* science as the “study of human behavior as a relationship between ends and scarce means” with alternative uses.^[4] Modern optimization theory includes traditional optimization theory but also overlaps with game theory and the study of economic equilibria. The *Journal of Economic Literature* codes classify mathematical programming, optimization techniques, and related topics under JEL:C61-C63.

In microeconomics, the utility maximization problem and its dual problem, the expenditure minimization problem, are economic optimization problems. Insofar as they behave consistently, consumers are assumed to maximize their utility, while firms are usually assumed to maximize their profit. Also, agents are often modeled as being risk-averse, thereby preferring to avoid risk. Asset prices are also modeled using optimization theory, though the underlying mathematics relies on optimizing stochastic processes rather than on static optimization. Trade theory also uses optimization to explain trade patterns between nations. The optimization of market portfolios is an example of multi-objective optimization in economics.

Since the 1970s, economists have modeled dynamic decisions over time using control theory. For example, microeconomists use dynamic search models to study labor-market behavior.^[5] A crucial distinction is between deterministic and stochastic models.^[6] Macroeconomists build dynamic stochastic general equilibrium (DSGE) models that describe the dynamics of the whole economy as

the result of the interdependent optimizing decisions of workers, consumers, investors, and governments.^{[7][8]}

1.7.3 Operations research

Another field that uses optimization techniques extensively is operations research.^[9] Operations research also uses stochastic modeling and simulation to support improved decision-making. Increasingly, operations research uses stochastic programming to model dynamic decisions that adapt to events; such problems can be solved with large-scale optimization and stochastic optimization methods.

1.7.4 Control engineering

Mathematical optimization is used in much modern controller design. High-level controllers such as Model predictive control (MPC) or Real-Time Optimization (RTO) employ mathematical optimization. These algorithms run online and repeatedly determine values for decision variables, such as choke openings in a process plant, by iteratively solving a mathematical optimization problem including constraints and a model of the system to be controlled.

1.7.5 Petroleum engineering

Nonlinear optimization methods are used to construct computational models of oil reservoirs.^[10]

1.7.6 Molecular modeling

Main article: [Molecular modeling](#)

Nonlinear optimization methods are widely used in conformational analysis.

1.8 Solvers

Main article: [List of optimization software](#)

1.9 See also

1.10 Notes

[1] "The Nature of Mathematical Programming," *Mathematical Programming Glossary*, INFORMS Computing Society.

- [2] W. Erwin Diewert (2008). "cost functions," *The New Palgrave Dictionary of Economics*, 2nd Edition Contents.
- [3] Battiti, Roberto; Mauro Brunato; Franco Mascia (2008). *Reactive Search and Intelligent Optimization*. Springer Verlag. ISBN 978-0-387-09623-0.
- [4] Lionel Robbins (1935, 2nd ed.) *An Essay on the Nature and Significance of Economic Science*, Macmillan, p. 16.
- [5] A. K. Dixit ([1976] 1990). *Optimization in Economic Theory*, 2nd ed., Oxford. Description and contents preview.
- [6] A.G. Malliaris (2008). "stochastic optimal control," *The New Palgrave Dictionary of Economics*, 2nd Edition. Abstract.
- [7] Julio Rotemberg and Michael Woodford (1997), "An Optimization-based Econometric Framework for the Evaluation of Monetary Policy.*NBER Macroeconomics Annual*, 12, pp. 297-346.
- [8] From *The New Palgrave Dictionary of Economics* (2008), 2nd Edition with Abstract links:
 - "numerical optimization methods in economics" by Karl Schmedders
 - "convex programming" by Lawrence E. Blume
 - "Arrow-Debreu model of general equilibrium" by John Geanakoplos.
- [9] "New force on the political scene: the Seophonisten". <http://www.seophonist-wahl.de>. Retrieved 14 September 2013.
- [10] History matching production data and uncertainty assessment with an efficient TSVD parameterization algorithm, Journal of Petroleum Science and Engineering, <http://www.sciencedirect.com/science/article/pii/S0920410513003227>

1.11 Further reading

1.11.1 Comprehensive

Undergraduate level

- Bradley, S.; Hax, A.; Magnanti, T. (1977). *Applied mathematical programming*. Addison Wesley.
- Rardin, Ronald L. (1997). *Optimization in operations research*. Prentice Hall. p. 919. ISBN 0-02-398415-5. copyright: 1998
- Strang, Gilbert (1986). *Introduction to applied mathematics*. Wellesley, MA: Wellesley-Cambridge Press (Strang's publishing company). pp. xii+758. ISBN 0-9614088-0-4. MR 870634.

Graduate level

- Magnanti, Thomas L. (1989). "Twenty years of mathematical programming". In Cornet, Bernard; Tulkens, Henry. *Contributions to Operations Research and Economics: The twentieth anniversary of CORE (Papers from the symposium held in Louvain-la-Neuve, January 1987)*. Cambridge, MA: MIT Press. pp. 163–227. ISBN 0-262-03149-3. MR 1104662.
- Minoux, M. (1986). *Mathematical programming: Theory and algorithms*. Egon Balas foreword) (Translated by Steven Vajda from the (1983 Paris: Dunod) French ed.). Chichester: A Wiley-Interscience Publication. John Wiley & Sons, Ltd. pp. xxviii+489. ISBN 0-471-90170-9. MR 2571910. (2008 Second ed., in French: *Programmation mathématique: Théorie et algorithmes*. Editions Tec & Doc, Paris, 2008. xxx+711 pp. ISBN 978-2-7430-1000-3).
- Nemhauser, G. L.; Rinnooy Kan, A. H. G.; Todd, M. J., eds. (1989). *Optimization*. Handbooks in Operations Research and Management Science 1. Amsterdam: North-Holland Publishing Co. pp. xiv+709. ISBN 0-444-87284-1. MR 1105099.
 - J. E. Dennis, Jr. and Robert B. Schnabel, A view of unconstrained optimization (pp. 1–72);
 - Donald Goldfarb and Michael J. Todd, Linear programming (pp. 73–170);
 - Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright, Constrained nonlinear programming (pp. 171–210);
 - Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin, Network flows (pp. 211–369);
 - W. R. Pulleyblank, Polyhedral combinatorics (pp. 371–446);
 - George L. Nemhauser and Laurence A. Wolsey, Integer programming (pp. 447–527);
 - Claude Lemaréchal, Nondifferentiable optimization (pp. 529–572);
 - Roger J-B Wets, Stochastic programming (pp. 573–629);
 - A. H. G. Rinnooy Kan and G. T. Timmer, Global optimization (pp. 631–662);
 - P. L. Yu, Multiple criteria decision making: five basic concepts (pp. 663–699).
- Shapiro, Jeremy F. (1979). *Mathematical programming: Structures and algorithms*. New York: Wiley-Interscience [John Wiley & Sons]. pp. xvi+388. ISBN 0-471-77886-9. MR 544669.

- Spall, J. C. (2003), *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*, Wiley, Hoboken, NJ.

1.11.2 Continuous optimization

- Roger Fletcher (2000). *Practical methods of optimization*. Wiley. ISBN 978-0-471-49463-8.
- Mordecai Avriel (2003). *Nonlinear Programming: Analysis and Methods*. Dover Publishing. ISBN 0-486-43227-0.
- P. E. Gill, W. Murray and M. H. Wright (1982). *Practical Optimization*. Emerald Publishing. ISBN 978-0122839528.
- Xin-She Yang (2010). *Engineering Optimization: An Introduction with Metaheuristic Applications*. Wiley. ISBN 978-0470582466.
- Bonnans, J. Frédéric; Gilbert, J. Charles; Lemaréchal, Claude; Sagastizábal, Claudia A. (2006). *Numerical optimization: Theoretical and practical aspects*. Universitext (Second revised ed. of translation of 1997 French ed.). Berlin: Springer-Verlag. pp. xiv+490. doi:10.1007/978-3-540-35447-5. ISBN 3-540-35445-X. MR 2265882.
- Bonnans, J. Frédéric; Shapiro, Alexander (2000). *Perturbation analysis of optimization problems*. Springer Series in Operations Research. New York: Springer-Verlag. pp. xviii+601. ISBN 0-387-98705-3. MR 1756264.
- Boyd, Stephen P.; Vandenberghe, Lieven (2004). *Convex Optimization* (pdf). Cambridge University Press. ISBN 978-0-521-83378-3. Retrieved October 15, 2011.
- Jorge Nocedal and Stephen J. Wright (2006). *Numerical Optimization*. Springer. ISBN 0-387-30303-0.
- Ruszczyński, Andrzej (2006). *Nonlinear Optimization*. Princeton, NJ: Princeton University Press. pp. xii+454. ISBN 978-0691119151. MR 2199043.
- Robert J. Vanderbei (2013). *Linear Programming: Foundations and Extensions, 4th Edition*. Springer. ISBN 978-1461476290.

1.11.3 Combinatorial optimization

- R. K. Ahuja, Thomas L. Magnanti, and James B. Orlin (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc. ISBN 0-13-617549-X.

- William J. Cook, William H. Cunningham, William R. Pulleyblank, Alexander Schrijver; *Combinatorial Optimization*; John Wiley & Sons; 1 edition (November 12, 1997); ISBN 0-471-55894-X.
- Gondran, Michel; Minoux, Michel (1984). *Graphs and algorithms*. Wiley-Interscience Series in Discrete Mathematics (Translated by Steven Vajda from the second (*Collection de la Direction des Études et Recherches d'Électricité de France* [Collection of the Department of Studies and Research of Électricité de France], v. 37. Paris: Éditions Eyrolles 1985. xxviii+545 pp. MR 868083) French ed.). Chichester: John Wiley & Sons, Ltd. pp. xix+650. ISBN 978-2-7430-1035-5. MR 2552933. (Fourth ed. Collection EDF R&D. Paris: Editions Tec & Doc 2009. xxxii+784 pp.
- Eugene Lawler (2001). *Combinatorial Optimization: Networks and Matroids*. Dover. ISBN 0-486-41453-1.
- Lawler, E. L.; Lenstra, J. K.; Rinnooy Kan, A. H. G.; Shmoys, D. B. (1985), *The traveling salesman problem: A guided tour of combinatorial optimization*, John Wiley & Sons, ISBN 0-471-90413-9.
- Jon Lee; *A First Course in Combinatorial Optimization*; Cambridge University Press; 2004; ISBN 0-521-01012-8.
- Christos H. Papadimitriou and Kenneth Steiglitz *Combinatorial Optimization : Algorithms and Complexity*; Dover Pubns; (paperback, Unabridged edition, July 1998) ISBN 0-486-40258-4.

1.11.4 Relaxation (extension method)

Methods to obtain suitable (in some sense) natural extensions of optimization problems that otherwise lack of existence or stability of solutions to obtain problems with guaranteed existence of solutions and their stability in some sense (typically under various perturbation of data) are in general called relaxation. Solutions of such extended (=relaxed) problems in some sense characterizes (at least certain features) of the original problems, e.g. as far as their optimizing sequences concerns. Relaxed problems may also possesses their own natural linear structure that may yield specific optimality conditions different from optimality conditions for the original problems.

- H. O. Fattorini: Infinite Dimensional Optimization and Control Theory. Cambridge Univ. Press, 1999.
- P. Pedregal: Parametrized Measures and Variational Principles. Birkhäuser, Basel, 1997

- T. Roubicek: “Relaxation in Optimization Theory and Variational Calculus”. W. de Gruyter, Berlin, 1997. ISBN 3-11-014542-1.
- J. Warga: Optimal control of differential and functional equations. Academic Press, 1972.

1.12 Journals

- *Computational Optimization and Applications*
- Journal of Computational *Optimization in Economics and Finance*
- *Journal of Economic Dynamics and Control*
- *SIAM Journal on Optimization* (SIOPT) and Editorial Policy
- *SIAM Journal on Control and Optimization* (SICON) and Editorial Policy

1.13 External links

- COIN-OR—Computational Infrastructure for Operations Research
- Decision Tree for Optimization Software Links to optimization source codes
- Global optimization
- Mathematical Programming Glossary
- Mathematical Programming Society
- NEOS Guide currently being replaced by the NEOS Wiki
- Optimization Online A repository for optimization e-prints
- Optimization Related Links
- Convex Optimization I EE364a: Course from Stanford University
- Convex Optimization – Boyd and Vandenberghe Book on Convex Optimization
- Book and Course on Optimization Methods for Engineering Design

Chapter 2

Golden section search

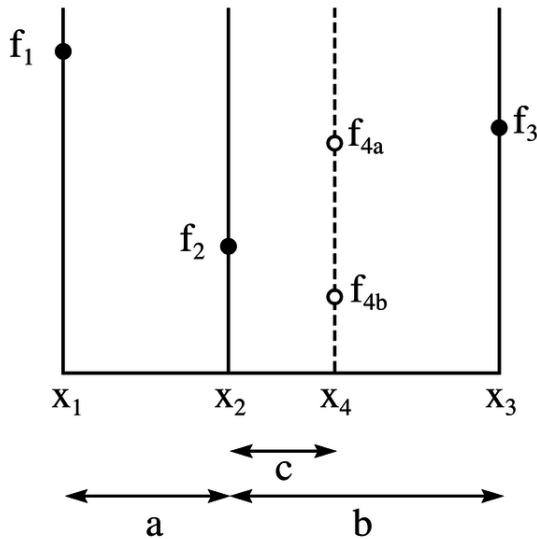


Diagram of a golden section search

The **golden section search** is a technique for finding the extremum (minimum or maximum) of a strictly unimodal function by successively narrowing the range of values inside which the extremum is known to exist. The technique derives its name from the fact that the algorithm maintains the function values for triples of points whose distances form a **golden ratio**. The algorithm is the limit of Fibonacci search (also described below) for a large number of function evaluations. Fibonacci search and Golden section search were discovered by Kiefer (1953). (see also Avriel and Wilde (1966)).

2.1 Basic idea

The diagram above illustrates a single step in the technique for finding a minimum. The functional values of $f(x)$ are on the vertical axis, and the horizontal axis is the x parameter. The value of $f(x)$ has already been evaluated at the three points: x_1 , x_2 , and x_3 . Since f_2 is smaller than either f_1 or f_3 , it is clear that a minimum lies inside the interval from x_1 to x_3 (since f is unimodal).

The next step in the minimization process is to “probe” the function by evaluating it at a new value of x , namely

x_4 . It is most efficient to choose x_4 somewhere inside the largest interval, i.e. between x_2 and x_3 . From the diagram, it is clear that if the function yields f_{4a} then a minimum lies between x_1 and x_4 and the new triplet of points will be x_1 , x_2 , and x_4 . However if the function yields the value f_{4b} then a minimum lies between x_2 and x_3 , and the new triplet of points will be x_2 , x_4 , and x_3 . Thus, in either case, we can construct a new narrower search interval that is guaranteed to contain the function’s minimum.

2.2 Probe point selection

From the diagram above, it is seen that the new search interval will be either between x_1 and x_4 with a length of $a+c$, or between x_2 and x_3 with a length of b . The golden section search requires that these intervals be equal. If they are not, a run of “bad luck” could lead to the wider interval being used many times, thus slowing down the rate of convergence. To ensure that $b = a+c$, the algorithm should choose $x_4 = x_1 + (x_3 - x_2)$.

However there still remains the question of where x_2 should be placed in relation to x_1 and x_3 . The golden section search chooses the spacing between these points in such a way that these points have the same proportion of spacing as the subsequent triple x_1, x_2, x_4 or x_2, x_4, x_3 . By maintaining the same proportion of spacing throughout the algorithm, we avoid a situation in which x_2 is very close to x_1 or x_3 , and guarantee that the interval width shrinks by the same constant proportion in each step.

Mathematically, to ensure that the spacing after evaluating $f(x_4)$ is proportional to the spacing prior to that evaluation, if $f(x_4)$ is f_{4a} and our new triplet of points is x_1 , x_2 , and x_4 then we want:

$$\frac{c}{a} = \frac{a}{b}.$$

However, if $f(x_4)$ is f_{4b} and our new triplet of points is x_2 , x_4 , and x_3 then we want:

$$\frac{c}{(b-c)} = \frac{a}{b}.$$

Eliminating c from these two simultaneous equations yields:

$$\left(\frac{b}{a}\right)^2 = \frac{b}{a} + 1$$

or

$$\frac{b}{a} = \varphi$$

where φ is the golden ratio:

$$\varphi = \frac{1 + \sqrt{5}}{2} = 1.618033988\dots$$

The appearance of the golden ratio in the proportional spacing of the evaluation points is how this search algorithm gets its name.

2.3 Termination condition

In addition to a routine for reducing the size of the bracketing of the solution, a complete algorithm must have a termination condition. The one provided in the book *Numerical Recipes in C* is based on testing the gaps among x_1, x_2, x_3 and x_4 , terminating when within the relative accuracy bounds:

$$|x_3 - x_1| < \tau(|x_2| + |x_4|)$$

where τ is a tolerance parameter of the algorithm and $|x|$ is the absolute value of x . The check is based on the bracket size relative to its central value, because that relative error in x is approximately proportional to the squared absolute error in $f(x)$ in typical cases. For that same reason, the Numerical Recipes text recommends that $\tau = \sqrt{\epsilon}$ where ϵ is the required absolute precision of $f(x)$.

2.4 Algorithm

2.4.1 Iterative algorithm

- Let $[a, b]$ be interval of current bracket. $f(a), f(b)$ would already have been computed earlier. $\varphi = (-1 + \sqrt{5})/2$.
- Let $c = b + \varphi(a - b)$, $d = a + \varphi(b - a)$. If $f(c), f(d)$ not available, compute them.
- If $f(c) < f(d)$ (this is to find min, to find max, just reverse it) then move the data: $(b, f(b)) \leftarrow (d, f(d))$, $(d, f(d)) \leftarrow (c, f(c))$ and update $c = b + \varphi(a - b)$ and $f(c)$;

- otherwise, move the data: $(a, f(a)) \leftarrow (c, f(c))$, $(c, f(c)) \leftarrow (d, f(d))$ and update $d = a + \varphi(b - a)$ and $f(d)$.
- At the end of the iteration, $[a, c, d, b]$ bracket the minimum point.

```
# python program for golden section search gr=0.618
def gss(f,a,b,tol=1e-5): """golden section search to find
the minimum of f on [a,b] f: a strictly unimodal function
on [a,b] example: >>> f=lambda x:(x-2)**2 >>>
x=gss(f,1,5) >>> x 1.9999905526669604 """ x=b-gr*(b-a)
y=a+gr*(b-a) while abs(x-y)>tol: fx=f(x);fy=f(y) if
fx<fy: b=y y=x #fy=fx;fx=f(x) x=b-gr*(b-a) else: a=x
x=y #fx=fy;fy=f(y) y=a+gr*(b-a) return (b+a)/2
```

2.4.2 Recursive algorithm

```
double phi = (1 + Math.sqrt(5)) / 2; double resphi = 2
- phi; // a and c are the current bounds; the minimum
is between them. // b is a center point // f(x) is some
mathematical function elsewhere defined // a corre-
sponds to x1; b corresponds to x2; c corresponds to x3
// x corresponds to x4 // tau is a tolerance parameter;
see above public double goldenSectionSearch(double a,
double b, double c, double tau) { double x; if (c - b > b
- a) x = b + resphi * (c - b); else x = b - resphi * (b - a);
if (Math.abs(c - a) < tau * (Math.abs(b) + Math.abs(x)))
return (c + a) / 2; assert(f(x) != f(b)); if (f(x) < f(b)) {
if (c - b > b - a) return goldenSectionSearch(b, x, c, tau);
else return goldenSectionSearch(a, x, b, tau); } else { if
(c - b > b - a) return goldenSectionSearch(a, b, x, tau);
else return goldenSectionSearch(x, b, c, tau); } }
```

To realise the advantage of golden section search, the function $f(x)$ would be implemented with caching, so that in all invocations of `goldenSectionSearch(..)` above, except the first, $f(x_2)$ would have already been evaluated previously — the result of the calculation will be re-used, bypassing the (perhaps expensive) explicit evaluation of the function. Together with a slightly smaller number of recursions, this 50% saving in the number of calls to $f(x)$ is the main algorithmic advantage over *Ternary search*.

2.5 Fibonacci search

A very similar algorithm can also be used to find the **extremum** (minimum or maximum) of a sequence of values that has a single local minimum or local maximum. In order to approximate the probe positions of golden section search while probing only integer sequence indices, the variant of the algorithm for this case typically maintains a bracketing of the solution in which the length of the bracketed interval is a **Fibonacci number**. For this

reason, the sequence variant of golden section search is often called *Fibonacci search*.

Fibonacci search was first devised by Kiefer (1953) as a minimax search for the maximum (minimum) of a unimodal function in an interval.

2.6 See also

- Fibonacci search technique
- Brent's method
- Binary search

2.7 References

- Kiefer, J. (1953), “Sequential minimax search for a maximum”, *Proceedings of the American Mathematical Society* **4** (3): 502–506, doi:10.2307/2032161, JSTOR 2032161, MR 0055639
- Avriel, Mordecai; Wilde, Douglass J. (1966), “Optimality proof for the symmetric Fibonacci search technique”, *Fibonacci Quarterly* **4**: 265–269, MR 0208812
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007), “Section 10.2. Golden Section Search in One Dimension”, *Numerical Recipes: The Art of Scientific Computing* (3rd ed.), New York: Cambridge University Press, ISBN 978-0-521-88068-8

Chapter 3

Powell's method

Powell's method, strictly **Powell's conjugate direction method**, is an algorithm proposed by Michael J. D. Powell for finding a local minimum of a function. The function need not be differentiable, and no derivatives are taken.

The function must be a real-valued function of a fixed number of real-valued inputs. The caller passes in the initial point. The caller also passes in a set of initial search vectors. Typically N search vectors are passed in which are simply the normals aligned to each axis.

The method minimises the function by a bi-directional search along each search vector, in turn. The new position can then be expressed as a linear combination of the search vectors. The new displacement vector becomes a new search vector, and is added to the end of the search vector list. Meanwhile the search vector which contributed most to the new direction, i.e. the one which was most successful, is deleted from the search vector list. The algorithm iterates an arbitrary number of times until no significant improvement is made.

The method is useful for calculating the local minimum of a continuous but complex function, especially one without an underlying mathematical definition, because it is not necessary to take derivatives. The basic algorithm is simple; the complexity is in the linear searches along the search vectors, which can be achieved via Brent's method.

3.1 References

- Powell, M. J. D. (1964). “An efficient method for finding the minimum of a function of several variables without calculating derivatives”. *Computer Journal* 7 (2): 155–162. doi:10.1093/comjnl/7.2.155.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). “Section 10.7. Direction Set (Powell's) Methods in Multidimensions”. *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.
- Brent, Richard P. (1973). “Section 7.3: Powell's algorithm”. *Algorithms for minimization without derivatives*. Englewood Cliffs, N.J.: Prentice-Hall. ISBN 0-486-41998-3.

3.2 External links

- Explanation of Powell's Method

Chapter 4

Line search

In optimization, the **line search** strategy is one of two basic iterative approaches to find a local minimum \mathbf{x}^* of an objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. The other approach is trust region.

The line search approach first finds a descent direction along which the objective function f will be reduced and then computes a step size that determines how far \mathbf{x} should move along that direction. The descent direction can be computed by various methods, such as gradient descent, Newton's method and Quasi-Newton method. The step size can be determined either exactly or inexactly.

4.1 Example use

Here is an example gradient method that uses a line search in step 4.

1. Set iteration counter $k = 0$, and make an initial guess, \mathbf{x}_0 for the minimum
2. Repeat:
3. Compute a descent direction \mathbf{p}_k
4. Choose α_k to 'loosely' minimize $h(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{p}_k)$ over $\alpha \in \mathbb{R}_+$
5. Update $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$, and $k = k + 1$
6. Until $\|\nabla f(\mathbf{x}_k)\| < \text{tolerance}$

At the line search step (4) the algorithm might either *exactly* minimize h , by solving $h'(\alpha_k) = 0$, or *loosely*, by asking for a sufficient decrease in h . One example of the former is **conjugate gradient method**. The latter is called inexact line search and may be performed in a number of ways, such as a **backtracking line search** or using the **Wolfe conditions**.

Like other optimization methods, line search may be combined with simulated annealing to allow it to jump over some local minima.

4.2 Algorithms

4.2.1 Direct search methods

In this method, the minimum must first be bracketed, so the algorithm must identify points x_1 and x_2 such that the sought minimum lies between them. The interval is then divided by computing $f(x)$ at two internal points, x_3 and x_4 , and rejecting whichever of the two outer points is not adjacent to that of x_3 and x_4 which has the lowest function value. In subsequent steps, only one extra internal point needs to be calculated. Of the various methods of dividing the interval,^[1] golden section search is particularly simple and effective, as the interval proportions are preserved regardless of how the search proceeds:

$$\frac{1}{\phi}(x_2 - x_1) = x_4 - x_1 = x_2 - x_3 = \phi(x_2 - x_4) = \phi(x_3 - x_1) = \phi^2(x_4 - x_3) \text{ where } \phi = \frac{1}{2}(1 + \sqrt{5}) \approx 1.618$$

4.3 See also

- Backtracking line search
- Secant method
- Newton–Raphson method
- Pattern search (optimization)
- Nelder–Mead method
- Golden section search

4.4 References

[1] Box, M. J.; Davies, D.; Swann, W. H. (1969). *Non-Linear optimisation Techniques*. Oliver & Boyd.

Chapter 5

Nelder–Mead method

See simplex algorithm for Dantzig's algorithm for the problem of linear optimization.

The **Nelder–Mead method** or **downhill simplex method** or **amoeba method** is a commonly used non-linear optimization technique, which is a well-defined numerical method for problems for which derivatives may not be known. However, the Nelder–Mead technique is a heuristic search method that can converge to non-stationary points^[1] on problems that can be solved by alternative methods.^[2]

The Nelder–Mead technique was proposed by John Nelder & Roger Mead (1965)^[3] and is a technique for minimizing an objective function in a many-dimensional space.

5.1 Overview

The method uses the concept of a **simplex**, which is a special **polytope** of $N + 1$ vertices in N dimensions. Examples of simplices include a line segment on a line, a triangle on a plane, a tetrahedron in three-dimensional space and so forth.

The method approximates a local optimum of a problem with N variables when the objective function varies smoothly and is **unimodal**.

For example, a suspension bridge engineer has to choose how thick each strut, cable, and pier must be. These elements are interdependent, but it is not easy to visualize the impact of changing any specific element. Simulation of such complicated structures is often extremely computationally expensive to run, possibly taking upwards of hours per execution. An engineer may therefore prefer the Nelder–Mead method as it requires fewer evaluations per iteration than other optimization methods.

Nelder–Mead generates a new test position by extrapolating the behavior of the objective function measured at each test point arranged as a simplex. The algorithm then chooses to replace one of these test points with the new test point and so the technique progresses. The simplest step is to replace the worst point with a point reflected through the **centroid** of the remaining N points. If this

point is better than the best current point, then we can try stretching exponentially out along this line. On the other hand, if this new point isn't much better than the previous value, then we are stepping across a valley, so we shrink the simplex towards a better point.

Unlike modern optimization methods, the Nelder–Mead heuristic can converge to a non-stationary point unless the problem satisfies stronger conditions than are necessary for modern methods.^[1] Modern improvements over the Nelder–Mead heuristic have been known since 1979.^[2]

Many variations exist depending on the actual nature of the problem being solved. A common variant uses a constant-size, small simplex that roughly follows the gradient direction (which gives steepest descent). Visualize a small triangle on an elevation map flip-flopping its way down a valley to a local bottom. This method is also known as the Flexible Polyhedron Method. This, however, tends to perform poorly against the method described in this article because it makes small, unnecessary steps in areas of little interest.

5.2 One possible variation of the NM algorithm

- 1. Order according to the values at the vertices:

$$f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \leq \cdots \leq f(\mathbf{x}_{n+1})$$

- 2. Calculate \mathbf{x}_o , the **centroid** of all points except \mathbf{x}_{n+1} .

- 3. **Reflection**

Compute reflected point $\mathbf{x}_r = \mathbf{x}_o + \alpha(\mathbf{x}_o - \mathbf{x}_{n+1})$

If the reflected point is better than the second worst, but not better than the best, i.e.: $f(\mathbf{x}_1) \leq f(\mathbf{x}_r) < f(\mathbf{x}_n)$,

then obtain a new simplex by replacing the worst point \mathbf{x}_{n+1} with the reflected point \mathbf{x}_r , and go to step 1.

- 4. Expansion

If the reflected point is the best point so far, $f(\mathbf{x}_r) < f(\mathbf{x}_1)$, then compute the expanded point $\mathbf{x}_e = \mathbf{x}_o + \gamma(\mathbf{x}_o - \mathbf{x}_{n+1})$

If the expanded point is better than the reflected point, $f(\mathbf{x}_e) < f(\mathbf{x}_r)$ then obtain a new simplex by replacing the worst point \mathbf{x}_{n+1} with the expanded point \mathbf{x}_e , and go to step 1.

Else obtain a new simplex by replacing the worst point \mathbf{x}_{n+1} with the reflected point \mathbf{x}_r , and go to step 1.

Else (i.e. reflected point is not better than second worst) continue at step 5.

- 5. Contraction

Here, it is certain that $f(\mathbf{x}_r) \geq f(\mathbf{x}_n)$

Compute contracted point $\mathbf{x}_c = \mathbf{x}_o + \rho(\mathbf{x}_o - \mathbf{x}_{n+1})$

If the contracted point is better than the worst point, i.e. $f(\mathbf{x}_c) < f(\mathbf{x}_{n+1})$

then obtain a new simplex by replacing the worst point \mathbf{x}_{n+1} with the contracted point \mathbf{x}_c , and go to step 1.

Else go to step 6.

- 6. Reduction

For all but the best point, replace the point with

$\mathbf{x}_i = \mathbf{x}_1 + \sigma(\mathbf{x}_i - \mathbf{x}_1)$ i all for $\in \{2, \dots, n+1\}$. go to step 1.

Note: α , γ , ρ and σ are respectively the reflection, the expansion, the contraction and the shrink coefficient. Standard values are $\alpha = 1$, $\gamma = 2$, $\rho = -1/2$ and $\sigma = 1/2$.

For the **reflection**, since \mathbf{x}_{n+1} is the vertex with the higher associated value among the vertices, we can expect to find a lower value at the reflection of \mathbf{x}_{n+1} in the opposite face formed by all vertices point \mathbf{x}_i except \mathbf{x}_{n+1} .

For the **expansion**, if the reflection point \mathbf{x}_r is the new minimum along the vertices we can expect to find interesting values along the direction from \mathbf{x}_o to \mathbf{x}_r .

Concerning the **contraction**: If $f(\mathbf{x}_r) > f(\mathbf{x}_n)$ we can expect that a better value will be inside the simplex formed by all the vertices \mathbf{x}_i .

Finally, the **reduction** handles the rare case that contracting away from the largest point increases f , something that cannot happen sufficiently close to a non-singular minimum. In that case we contract towards the lowest point in the expectation of finding a simpler landscape.

The initial simplex is important, indeed, a too small initial simplex can lead to a local search, consequently the NM can get more easily stuck. So this simplex should depend on the nature of the problem.

5.3 See also

- Derivative-free optimization
- COBYLA
- NEWUOA
- LINCOA
- Conjugate gradient method
- Levenberg–Marquardt algorithm
- Broyden–Fletcher–Goldfarb–Shanno or BFGS method
- Differential evolution
- Pattern search (optimization)
- CMA-ES

5.4 References

- [1] • Powell, Michael J. D. (1973). “On Search Directions for Minimization Algorithms”. *Mathematical Programming* **4**: 193–201. doi:10.1007/bf01584660.
- [2] • McKinnon, K.I.M. (1999). “Convergence of the Nelder–Mead simplex method to a non-stationary point”. *SIAM J Optimization* **9**: 148–158. doi:10.1137/S1052623496303482. (algorithm summary online).
- Yu, Wen Ci. 1979. “Positive basis and a class of direct search techniques”. *Scientia Sinica [Zhongguo Kexue]*: 53—68.

- Yu, Wen Ci. 1979. “The convergent property of the simplex evolutionary technique”. *Scientia Sinica [Zhongguo Kexue]*: 69–77.
 - Kolda, Tamara G.; Lewis, Robert Michael; Torczon, Virginia (2003). “Optimization by direct search: new perspectives on some classical and modern methods”. *SIAM Rev.* **45**: 385–482. doi:10.1137/S003614450242889.
 - Lewis, Robert Michael; Shepherd, Anne; Torczon, Virginia (2007). “Implementing generating set search methods for linearly constrained minimization”. *SIAM J. Sci. Comput.* **29**: 2507–2530. doi:10.1137/050635432.
- [3] Nelder, John A.; R. Mead (1965). “A simplex method for function minimization”. *Computer Journal* **7**: 308–313. doi:10.1093/comjnl/7.4.308.

5.4.1 Further reading

- Avriel, Mordecai (2003). *Nonlinear Programming: Analysis and Methods*. Dover Publishing. ISBN 0-486-43227-0.
- Coope, I. D.; C.J. Price, 2002. “Positive bases in numerical optimization”, *Computational Optimization & Applications*, Vol. 21, No. 2, pp. 169–176, 2002.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). “Section 10.5. Downhill Simplex Method in Multidimensions”. *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.

5.5 External links

- Nelder–Mead (Simplex) Method
- Nelder–Mead (Downhill Simplex) explanation and visualization with the Rosenbrock banana function
- Nelder–Mead Search for a Minimum
- John Burkardt: Nelder–Mead code in Matlab - note that a variation of the Nelder–Mead method is also implemented by the Matlab function fminsearch.
- Nelder–Mead online for the calibration of the SABR model - Application in Finance.
- SOVA 1.0 (freeware) - Simplex Optimization for Various Applications
- - HillStormer, a practical tool for nonlinear, multivariate and constrained Simplex Optimization by Nelder Mead.

Chapter 6

Successive parabolic interpolation

Successive parabolic interpolation is a technique for finding the **extremum** (minimum or maximum) of a continuous **unimodal** function by successively fitting **parabolas** (polynomials of degree two) to the function at three unique points, and at each iteration replacing the “oldest” point with the extremum of the fitted parabola.

6.1 Advantages

Only function values are used, and when this method converges to an extremum, it does so with an **order of convergence** of approximately 1.325. The superlinear rate of convergence is superior to that of other methods with only linear convergence (such as **line search**). Moreover, not requiring the computation or approximation of function **derivatives** makes successive parabolic interpolation a popular alternative to other methods that do require them (such as **gradient descent** and **Newton’s method**).

6.2 Disadvantages

On the other hand, convergence (even to a local extremum) is not guaranteed when using this method in isolation. For example, if the three points are **collinear**, the resulting parabola is **degenerate** and thus does not provide a new candidate point. Furthermore, if function derivatives are available, **Newton’s method** is applicable and exhibits quadratic convergence.

6.3 Improvements

Alternating the parabolic iterations with a more robust method (**golden section search** is a popular choice) to choose candidates can greatly increase the probability of convergence without hampering the convergence rate.

6.4 See also

- Inverse quadratic interpolation is a related method that uses parabolas to find roots rather than extrema.
- Simpson’s rule uses parabolas to approximate definite integrals.

6.5 References

Michael Heath (2002). *Scientific Computing: An Introductory Survey* (2nd ed.). New York: McGraw-Hill. ISBN 0-07-239910-4.

Chapter 7

Trust region

Trust region is a term used in mathematical optimization to denote the subset of the region of the objective function that is approximated using a model function (often a quadratic). If an adequate model of the objective function is found within the trust region then the region is expanded; conversely, if the approximation is poor then the region is contracted. Trust region methods are also known as **restricted step methods**.

The fit is evaluated by comparing the ratio of expected improvement from the model approximation with the actual improvement observed in the objective function. Simple thresholding of the ratio is used as the criterion for expansion and contraction—a model function is “trusted” only in the region where it provides a reasonable approximation.

Trust region methods are in some sense dual to line search methods: trust region methods first choose a step size (the size of the trust region) and then a step direction while line search methods first choose a step direction and then a step size.

The earliest use of the term seems to be by Sorensen (1982).

7.1 Example

Conceptually, in the Levenberg–Marquardt algorithm, the objective function is iteratively approximated by a quadratic surface, then using a linear solve, the estimate is updated. This alone may not converge nicely if the initial guess is too far from the optimum. For this reason, the algorithm instead restricts each step, preventing it from stepping “too far”. It operationalizes “too far” as follows. Rather than solving $A\Delta x = b$ for Δx , it solves $(A + \lambda \text{diag}(A))\Delta x = b$ where $\text{diag}(A)$ is the diagonal matrix with the same diagonal as A and λ is a parameter that controls the trust-region size. Geometrically, this adds a paraboloid centered at $\Delta x = 0$ to the quadratic form, resulting in a smaller step.

The trick is to change the trust-region size (λ). At each iteration, the damped quadratic fit predicts a certain reduction in the cost function, Δf_{pred} , which we would expect to be a smaller reduction than the true reduction. Given

Δx we can evaluate

$$\Delta f_{\text{actual}} = f(x + \Delta x) - f(x)$$

By looking at the ratio $\Delta f_{\text{pred}}/\Delta f_{\text{actual}}$ we can adjust the trust-region size. In general, we expect Δf_{pred} to be a bit less than Δf_{actual} and so the ratio would be between, say, 0.25 and 0.5. If the ratio is more than 0.5, then we aren’t damping the step much, so expand the trust region (decrease λ), and iterate. If the ratio is smaller than 0.25, then the true function is diverging “too much” from the trust-region approximation, so shrink the trust region (increase λ) and try again.

7.2 References

- Andrew R. Conn, Nicholas I. M. Gould, Philippe L. Toint "Trust-Region Methods (MPS-SIAM Series on Optimization)".
- Byrd, R. H., R. B. Schnabel, and G. A. Schultz. "A trust region algorithm for nonlinearly constrained optimization", SIAM J. Numer. Anal., 24 (1987), pp. 1152–1170.
- Sorensen, D. C.: "Newton’s Method with a Model Trust Region Modification", SIAM J. Numer. Anal., 19(2), 409–426 (1982).
- Yuan, Y. "A review of trust region algorithms for optimization" in ICIAM 99: Proceedings of the Fourth International Congress on Industrial & Applied Mathematics, Edinburgh, 2000 Oxford University Press, USA.

7.3 External links

- Kranf site: Trust Region Algorithms

Chapter 8

Wolfe conditions

In the unconstrained minimization problem, the **Wolfe conditions** are a set of inequalities for performing **inexact line search**, especially in quasi-Newton methods, first published by Philip Wolfe in 1969.^{[1][2]}

In these methods the idea is to find

$$\min_x f(\mathbf{x})$$

for some smooth $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Each step often involves approximately solving the subproblem

$$\min_{\alpha} f(\mathbf{x}_k + \alpha \mathbf{p}_k)$$

where \mathbf{x}_k is the current best guess, $\mathbf{p}_k \in \mathbb{R}^n$ is a search direction, and $\alpha \in \mathbb{R}$ is the step length.

The inexact line searches provide an efficient way of computing an acceptable step length α that reduces the objective function 'sufficiently', rather than minimizing the objective function over $\alpha \in \mathbb{R}^+$ exactly. A line search algorithm can use Wolfe conditions as a requirement for any guessed α , before finding a new search direction \mathbf{p}_k .

8.1 Armijo rule and curvature

Denote a univariate function ϕ restricted to the direction \mathbf{p}_k as $\phi(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{p}_k)$. A step length α_k is said to satisfy the *Wolfe conditions* if the following two inequalities hold:

$$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \leq f(\mathbf{x}_k) + c_1 \alpha_k \mathbf{p}_k^T \nabla f(\mathbf{x}_k)$$

$$\mathbf{p}_k^T \nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \geq c_2 \mathbf{p}_k^T \nabla f(\mathbf{x}_k)$$

with $0 < c_1 < c_2 < 1$. (In examining condition (ii), recall that to ensure that \mathbf{p}_k is a descent direction, we have $\mathbf{p}_k^T \nabla f(\mathbf{x}_k) < 0$.)

c_1 is usually chosen to be quite small while c_2 is much larger; Nocedal^[3] gives example values of $c_1 = 10^{-4}$ and $c_2 = 0.9$ for Newton or quasi-Newton methods and $c_2 = 0.1$ for the nonlinear conjugate gradient method. Inequality i) is known as the **Armijo rule**^[4] and ii) as the **curvature condition**; i) ensures that the step length α_k decreases f 'sufficiently', and ii) ensures that the slope has been reduced sufficiently.

8.2 Strong Wolfe condition on curvature

The Wolfe conditions, however, can result in a value for the step length that is not close to a minimizer of ϕ . If we modify the curvature condition to the following,

$$|\mathbf{p}_k^T \nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)| \leq c_2 |\mathbf{p}_k^T \nabla f(\mathbf{x}_k)|$$

then i) and ii) together form the so-called **strong Wolfe conditions**, and force α_k to lie close to a critical point of ϕ .

8.3 Rationale

The principal reason for imposing the Wolfe conditions in an optimization algorithm where $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \mathbf{p}_k$ is to ensure convergence of the gradient to zero. In particular, if the cosine of the angle between \mathbf{p}_k and the gradient,

$$\cos \theta_k = \frac{\nabla f(\mathbf{x}_k)^T \mathbf{p}_k}{\|\nabla f(\mathbf{x}_k)\| \|\mathbf{p}_k\|}$$

is bounded away from zero and the i) and ii) conditions hold, then $\nabla f(\mathbf{x}_k) \rightarrow 0$.

An additional motivation, in the case of a quasi-Newton method is that if $\mathbf{p}_k = -B_k^{-1} \nabla f(\mathbf{x}_k)$, where the matrix B_k is updated by the BFGS or DFP formula, then if B_k is positive definite ii) implies B_{k+1} is also positive definite.

8.4 References

- “Line Search Methods”. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. 2006. pp. 30–32. doi:10.1007/978-0-387-40065-5_3. ISBN 978-0-387-30303-1.
 - “Quasi-Newton Methods”. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. 2006. pp. 135–163. doi:10.1007/978-0-387-40065-5_6. ISBN 978-0-387-30303-1.
- [1] Wolfe, P. (1969). “Convergence Conditions for Ascent Methods”. *SIAM Review* **11** (2): 226–000. doi:10.1137/1011036. JSTOR 2028111.
- [2] Wolfe, P. (1971). “Convergence Conditions for Ascent Methods. II: Some Corrections”. *SIAM Review* **13** (2): 185–000. doi:10.1137/1013035.
- [3] Nocedal, Jorge; Wright, Stephen (1999). *Numerical Optimization*.
- [4] Armijo, Larry (1966). “Minimization of functions having Lipschitz continuous first partial derivatives”. *Pacific J. Math.* **16** (1): 1–3.

Chapter 9

Broyden–Fletcher–Goldfarb–Shanno algorithm

In numerical optimization, the **Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm** is an iterative method for solving unconstrained nonlinear optimization problems.

The BFGS method approximates Newton's method, a class of hill-climbing optimization techniques that seeks a stationary point of a (preferably twice continuously differentiable) function. For such problems, a necessary condition for optimality is that the gradient be zero. Newton's method and the BFGS methods are not guaranteed to converge unless the function has a quadratic Taylor expansion near an optimum. These methods use both the first and second derivatives of the function. However, BFGS has proven to have good performance even for non-smooth optimizations.

In quasi-Newton methods, the Hessian matrix of second derivatives doesn't need to be evaluated directly. Instead, the Hessian matrix is approximated using rank-one updates specified by gradient evaluations (or approximate gradient evaluations). Quasi-Newton methods are generalizations of the secant method to find the root of the first derivative for multidimensional problems. In multidimensional problems, the secant equation does not specify a unique solution, and quasi-Newton methods differ in how they constrain the solution. The BFGS method is one of the most popular members of this class.^[1] Also in common use is L-BFGS, which is a limited-memory version of BFGS that is particularly suited to problems with very large numbers of variables (e.g., >1000). The BFGS-B^[2] variant handles simple box constraints.

9.1 Rationale

The search direction \mathbf{p}_k at stage k is given by the solution of the analogue of the Newton equation

$$B_k \mathbf{p}_k = -\nabla f(\mathbf{x}_k)$$

where B_k is an approximation to the Hessian matrix which is updated iteratively at each stage, and $\nabla f(\mathbf{x}_k)$ is

the gradient of the function evaluated at \mathbf{x}_k . A line search in the direction \mathbf{p}_k is then used to find the next point \mathbf{x}_{k+1} . Instead of requiring the full Hessian matrix at the point \mathbf{x}_{k+1} to be computed as B_{k+1} , the approximate Hessian at stage k is updated by the addition of two matrices.

$$B_{k+1} = B_k + U_k + V_k$$

Both U_k and V_k are symmetric rank-one matrices but have different (matrix) bases. The symmetric rank one assumption here means that we may write

$$C = \mathbf{a}\mathbf{b}^T$$

So equivalently, U_k and V_k construct a rank-two update matrix which is robust against the scale problem often suffered in the gradient descent searching (e.g., in Broyden's method).

The quasi-Newton condition imposed on this update is

$$B_{k+1}(\mathbf{x}_{k+1} - \mathbf{x}_k) = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k).$$

9.2 Algorithm

From an initial guess \mathbf{x}_0 and an approximate Hessian matrix B_0 the following steps are repeated as \mathbf{x}_k converges to the solution.

1. Obtain a direction \mathbf{p}_k by solving: $B_k \mathbf{p}_k = -\nabla f(\mathbf{x}_k)$.
2. Perform a line search to find an acceptable stepsize α_k in the direction found in the first step, then update $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$.
3. Set $\mathbf{s}_k = \alpha_k \mathbf{p}_k$.
4. $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$.

$$5. B_{k+1} = B_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{B_k \mathbf{s}_k \mathbf{s}_k^T B_k}{\mathbf{s}_k^T B_k \mathbf{s}_k}.$$

$f(\mathbf{x})$ denotes the objective function to be minimized. Convergence can be checked by observing the norm of the gradient, $|\nabla f(\mathbf{x}_k)|$. Practically, B_0 can be initialized with $B_0 = I$, so that the first step will be equivalent to a gradient descent, but further steps are more and more refined by B_k , the approximation to the Hessian.

The first step of the algorithm is carried out using the inverse of the matrix B_k , which is usually obtained efficiently by applying the Sherman–Morrison formula to the fifth line of the algorithm, giving

$$B_{k+1}^{-1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k} \right) B_k^{-1} \left(I - \frac{y_k s_k^T}{y_k^T s_k} \right) + \frac{s_k s_k^T}{y_k^T s_k}.$$

This can be computed efficiently without temporary matrices, recognizing that B_k^{-1} is symmetric, and that $\mathbf{y}_k^T B_k^{-1} \mathbf{y}_k$ and $\mathbf{s}_k^T \mathbf{y}_k$ are scalar, using an expansion such as

$$B_{k+1}^{-1} = B_k^{-1} + \frac{(\mathbf{s}_k^T \mathbf{y}_k + \mathbf{y}_k^T B_k^{-1} \mathbf{y}_k)(\mathbf{s}_k \mathbf{s}_k^T)}{(\mathbf{s}_k^T \mathbf{y}_k)^2} - \frac{B_k^{-1} \mathbf{y}_k \mathbf{s}_k^T + \mathbf{s}_k \mathbf{y}_k^T B_k^{-1}}{\mathbf{s}_k^T \mathbf{y}_k}.$$

`Eigen_(C++_library)` are available on [github](#) under the [MIT_License](#) here.

BFGS and L-BFGS are also implemented in C as part of the open-source Gnu Regression, Econometrics and Time-series Library ([gretl](#)).

9.4 See also

- Quasi-Newton methods
- Davidon–Fletcher–Powell formula
- L-BFGS
- Gradient descent
- Nelder–Mead method
- Pattern search (optimization)
- BHHH algorithm

9.5 Notes

[1] Nocedal & Wright (2006), page 24

[2] Byrd, Richard H.; Lu, Peihuang; Nocedal, Jorge; Zhu, Ciyou (1995), “A Limited Memory Algorithm for Bound Constrained Optimization”, *SIAM Journal on Scientific Computing* **16** (5): 1190–1208, doi:10.1137/0916069

9.6 Bibliography

- Avriel, Mordca (2003), *Nonlinear Programming: Analysis and Methods*, Dover Publishing, ISBN 0-486-43227-0
- Bonnans, J. Frédéric; Gilbert, J. Charles; Lemaréchal, Claude; Sagastizábal, Claudia A. (2006), *Numerical optimization: Theoretical and practical aspects*, Universitext (Second revised ed. of translation of 1997 French ed.), Berlin: Springer-Verlag, pp. xiv+490, doi:10.1007/978-3-540-35447-5, ISBN 3-540-35445-X, MR 2265882
- Broyden, C. G. (1970), “The convergence of a class of double-rank minimization algorithms”, *Journal of the Institute of Mathematics and Its Applications* **6**: 76–90, doi:10.1093/imamat/6.1.76
- Fletcher, R. (1970), “A New Approach to Variable Metric Algorithms”, *Computer Journal* **13** (3): 317–322, doi:10.1093/comjnl/13.3.317
- Fletcher, Roger (1987), *Practical methods of optimization* (2nd ed.), New York: John Wiley & Sons, ISBN 978-0-471-91547-8

9.3 Implementations

The `GSL` implements BFGS as `gsl_multimin_fdfminimizer_vector_bfgs2`. `Ceres Solver` implements both BFGS and L-BFGS. In `SciPy`, the `scipy.optimize.fmin_bfgs` function implements BFGS. It is also possible to run BFGS using any of the L-BFGS algorithms by setting the parameter `L` to a very large number.

`Octave` uses BFGS with a double-dogleg approximation to the cubic line search.

In the `MATLAB Optimization Toolbox`, the `fminunc` function uses BFGS with cubic line search when the problem size is set to “medium scale.”

A high-precision arithmetic version of BFGS (pBFGS), implemented in C++ and integrated with the high-precision arithmetic package `ARPREC` is robust against numerical instability (e.g. round-off errors).

Another C++ implementation of BFGS, along with L-BFGS, L-BFGS-B, CG, and Newton’s method) using

- Goldfarb, D. (1970), “A Family of Variable Metric Updates Derived by Variational Means”, *Mathematics of Computation* **24** (109): 23–26, doi:10.1090/S0025-5718-1970-0258249-6
- Luenberger, David G.; Ye, Yinyu (2008), *Linear and nonlinear programming*, International Series in Operations Research & Management Science **116** (Third ed.), New York: Springer, pp. xiv+546, ISBN 978-0-387-74502-2, MR 2423726
- Nocedal, Jorge; Wright, Stephen J. (2006), *Numerical Optimization* (2nd ed.), Berlin, New York: Springer-Verlag, ISBN 978-0-387-30303-1
- Shanno, David F. (July 1970), “Conditioning of quasi-Newton methods for function minimization”, *Math. Comput.* **24** (111): 647–656, doi:10.1090/S0025-5718-1970-0274029-X, MR 42:8905
- Shanno, David F.; Kettler, Paul C. (July 1970), “Optimal conditioning of quasi-Newton methods”, *Math. Comput.* **24** (111): 657–664, doi:10.1090/S0025-5718-1970-0274030-6, MR 42:8906

9.7 External links

- Source code of high-precision BFGS A C++ source code of BFGS with high-precision arithmetic

Chapter 10

Limited-memory BFGS

Limited-memory BFGS (L-BFGS or LM-BFGS) is an optimization algorithm in the family of quasi-Newton methods that approximates the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm using a limited amount of computer memory. It is a popular algorithm for parameter estimation in machine learning.^{[1][2]}

Like the original BFGS, L-BFGS uses an approximation to the inverse Hessian matrix to steer its search through variable space, but where BFGS stores a dense $n \times n$ approximation to the inverse Hessian (n being the number of variables in the problem), L-BFGS stores only a few vectors that represent the approximation implicitly. Due to its resulting linear memory requirement, the L-BFGS method is particularly well suited for optimization problems with a large number of variables. Instead of the inverse Hessian \mathbf{H}_k , L-BFGS maintains a history of the past m updates of the position \mathbf{x} and gradient $\nabla f(\mathbf{x})$, where generally the history size m can be small (often $m < 10$). These updates are used to implicitly do operations requiring the \mathbf{H}_k -vector product.

10.1 Algorithm

L-BFGS shares many features with other quasi-Newton algorithms, but is very different in how the matrix-vector multiplication for finding the search direction is carried out $d_k = -\mathbf{H}_k g_k$. There are multiple published approaches to using a history of updates to form this direction vector. Here, we give a common approach, the so-called “two loop recursion.”^{[3][4]}

We'll take as given x_k , the position at the k -th iteration, and $g_k \equiv \nabla f(x_k)$ where f is the function being minimized, and all vectors are column vectors. We also assume that we have stored the last m updates of the form $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$. We'll define $\rho_k = \frac{1}{y_k^T s_k}$, and H_k^0 will be the ‘initial’ approximate of the inverse Hessian that our estimate at iteration k begins with. Then we can compute the (uphill) direction as follows:

$$q = g_k$$

For $i = k-1, k-2, \dots, k-m$

$$\alpha_i = \rho_i s_i^T q$$

$$q = q - \alpha_i y_i$$

$$H_k = y_{k-1}^T s_{k-1} / y_{k-1}^T y_{k-1}$$

$$z = H_k q$$

For $i = k-m, k-m+1, \dots, k-1$

$$\beta_i = \rho_i y_i^T z$$

$$z = z + s_i (\alpha_i - \beta_i)$$

Stop with $H_k g_k = z$

This formulation is valid whether we are minimizing or maximizing. Note that if we are minimizing, the search direction would be the negative of z (since z is “uphill”), and if we are maximizing, H_k^0 should be negative definite rather than positive definite. We would typically do a backtracking line search in the search direction (any line search would be valid, but L-BFGS does not require exact line searches in order to converge).

Commonly, the inverse Hessian H_k^0 is represented as a diagonal matrix, so that initially setting z requires only an element-by-element multiplication.

This two loop update only works for the inverse Hessian. Approaches to implementing L-BFGS using the direct approximate Hessian B_k have also been developed, as have other means of approximating the inverse Hessian.^[5]

10.2 Applications

L-BFGS has been called “the algorithm of choice” for fitting log-linear (MaxEnt) models and conditional random fields with ℓ_2 -regularization.^[2]

10.3 Variants

Since BFGS (and hence L-BFGS) is designed to minimize smooth functions without constraints, the L-BFGS

algorithm must be modified to handle functions that include non-differentiable components or constraints. A popular class of modifications are called active-set methods, based on the concept of the active set. The idea is that when restricted to a small neighborhood of the current iterate, the function and constraints can be simplified.

10.3.1 L-BFGS-B

The **L-BFGS-B** algorithm extends L-BFGS to handle simple box constraints (aka bound constraints) on variables; that is, constraints of the form $l_i \leq x_i \leq u_i$ where l_i and u_i are per-variable constant lower and upper bounds, respectively (for each x_i , either or both bounds may be omitted).^{[6][7]} The method works by identifying fixed and free variables at every step (using a simple gradient method), and then using the L-BFGS method on the free variables only to get higher accuracy, and then repeating the process.

10.3.2 OWL-QN

Orthant-wise limited-memory quasi-Newton (OWL-QN) is an L-BFGS variant for fitting ℓ_1 -regularized models, exploiting the inherent sparsity of such models.^[2] It minimizes functions of the form

$$f(\vec{x}) = g(\vec{x}) + C\|\vec{x}\|_1$$

where g is a differentiable convex loss function. The method is an active-set type method: at each iterate, it estimates the sign of each component of the variable, and restricts the subsequent step to have the same sign. Once the sign is fixed, the non-differentiable $\|\vec{x}\|_1$ term becomes a smooth linear term which can be handled by L-BFGS. After a L-BFGS step, the method allows some variables to change sign, and repeats the process.

10.3.3 O-LBFGS

Schraudolph *et al.* present an online approximation to both BFGS and L-BFGS.^[8] Similar to stochastic gradient descent, this can be used to reduce the computational complexity by evaluating the error function and gradient on a randomly drawn subset of the overall dataset in each iteration.

10.4 Implementations

An early, open source implementation of L-BFGS in Fortran exists in **Netlib** as a **shar** archive. Multiple other open source implementations have been produced as translations of this Fortran code (e.g. **java**, and

python via **SciPy**). Other implementations exist (e.g. **fmincon** (**Matlab** optimization toolbox), **FMINLBFGS** (for **Matlab**, **BSD license**), **minFunc** (also for **Matlab**), **LBFGS-D** (in the **D** programming language)), frequently as part of generic optimization libraries (e.g. **Mathematica**, **FuncLib C# library**, and **dlib C++ library**). The **libLBFGS** is a **C** implementation.

10.4.1 Implementations of variants

The L-BFGS-B variant also exists as **ACM TOMS** algorithm 778.^[7] In February 2011, some of the authors of the original L-BFGS-B code posted a major update (version 3.0).

A reference implementation^[9] is available in **Fortran 77** (and with a **Fortran 90** interface) at the author's website. This version, as well as older versions, has been converted to many other languages, including a **Java** wrapper for v3.0; **Matlab** interfaces for v3.0, v2.4, and v2.1; a **C++** interface for v2.1; a **Python** interface for v3.0 as part of **scipy.optimize.minimize**; an **OCaml** interface for v2.1 and v3.0; version 2.3 has been converted to **C** by **f2c** and is available at this **website**; and **R**'s **optim** general-purpose optimizer routine includes L-BFGS-B by using method="L-BFGS-B".^[10]

There exists a complete **C++11** rewrite of the L-BFGS-B solver using **Eigen3**. OWL-QN implementations are available in:

- **C++** implementation by its designers, includes the original ICML paper on the algorithm^[2]
- The **CRF** toolkit **Wapiti** includes a **C** implementation

10.5 Works cited

- [1] Malouf, Robert (2002). “A comparison of algorithms for maximum entropy parameter estimation”. *Proc. Sixth Conf. on Natural Language Learning (CoNLL)*. pp. 49–55.
- [2] Andrew, Galen; Gao, Jianfeng (2007). “Scalable training of L_1 -regularized log-linear models”. *Proceedings of the 24th International Conference on Machine Learning*. doi:10.1145/1273496.1273501. ISBN 9781595937933.
- [3] Matthies, H.; Strang, G. (1979). “The solution of non linear finite element equations”. *International Journal for Numerical Methods in Engineering* **14** (11): 1613–1626. doi:10.1002/nme.1620141104.
- [4] Nocedal, J. (1980). “Updating Quasi-Newton Matrices with Limited Storage”. *Mathematics of Computation* **35** (151): 773–782. doi:10.1090/S0025-5718-1980-0572855-7.

- [5] Byrd, R. H.; Nocedal, J.; Schnabel, R. B. (1994). “Representations of Quasi-Newton Matrices and their use in Limited Memory Methods”. *Mathematical Programming* **63** (4): 129–156. doi:10.1007/BF01582063.
- [6] Byrd, R. H.; Lu, P.; Nocedal, J.; Zhu, C. (1995). “A Limited Memory Algorithm for Bound Constrained Optimization”. *SIAM J. Sci. Comput.* **16** (5): 1190–1208. doi:10.1137/0916069.
- [7] Zhu, C.; Byrd, Richard H.; Lu, Peihuang; Nocedal, Jorge (1997). “L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization”. *ACM Transactions on Mathematical Software* **23** (4): 550–560. doi:10.1145/279232.279236.
- [8] N. Schraudolph, J. Yu, and S. Günter (2007). *A stochastic quasi-Newton method for online convex optimization*. AISTATS.
- [9] Morales, J. L.; Nocedal, J. (2011). “Remark on “algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization””. *ACM Transactions on Mathematical Software* **38**: 1. doi:10.1145/2049662.2049669.
- [10] “General-purpose Optimization”. *R documentation*. Comprehensive R Archive Network.

10.6 Further reading

- Liu, D. C.; Nocedal, J. (1989). “On the Limited Memory Method for Large Scale Optimization”. *Mathematical Programming B* **45** (3): 503–528. doi:10.1007/BF01589116.
- Byrd, Richard H.; Lu, Peihuang; Nocedal, Jorge; Zhu, Ciyou (1995). “A Limited Memory Algorithm for Bound Constrained Optimization”. *SIAM Journal on Scientific and Statistical Computing* **16** (5): 1190–1208. doi:10.1137/0916069.

Chapter 11

Davidon–Fletcher–Powell formula

The **Davidon–Fletcher–Powell formula** (or **DFP**, named after William C. Davidon, Roger Fletcher, and Michael J. D. Powell) finds the solution to the secant equation that is closest to the current estimate and satisfies the curvature condition (see below). It was the first quasi-Newton method to generalize the secant method to a multidimensional problem. This update maintains the symmetry and positive definiteness of the Hessian matrix.

Given a function $f(x)$, its gradient (∇f), and positive definite Hessian matrix B , the Taylor series is:

$$f(x_k + s_k) = f(x_k) + \nabla f(x_k)^T s_k + \frac{1}{2} s_k^T B s_k,$$

and the Taylor series of the gradient itself (secant equation):

$$\nabla f(x_k + s_k) = \nabla f(x_k) + B s_k,$$

is used to update B . The DFP formula finds a solution that is symmetric, positive definite and closest to the current approximate value of B_k :

$$B_{k+1} = (I - \gamma_k y_k s_k^T) B_k (I - \gamma_k s_k y_k^T) + \gamma_k y_k y_k^T,$$

where

$$y_k = \nabla f(x_k + s_k) - \nabla f(x_k),$$

$$\gamma_k = \frac{1}{y_k^T s_k}.$$

and B_k is a symmetric and positive definite matrix. The corresponding update to the inverse Hessian approximation $H_k = B_k^{-1}$ is given by:

$$H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + \frac{s_k s_k^T}{y_k^T s_k}.$$

B is assumed to be positive definite, and the vectors s_k^T and y must satisfy the curvature condition:

$$s_k^T y_k = s_k^T B s_k > 0.$$

The DFP formula is quite effective, but it was soon superseded by the **BFGS formula**, which is its dual (interchanging the roles of y and s).

11.1 See also

- Newton's method
- Newton's method in optimization
- Quasi-Newton method
- Broyden–Fletcher–Goldfarb–Shanno (BFGS) method
- L-BFGS method
- SR1 formula
- Nelder–Mead method

11.2 References

- Davidon, W. C. (1991), “Variable metric method for minimization”, *SIAM Journal on Optimization* **1**: 1–17, doi:10.1137/0801001
- Fletcher, Roger (1987), *Practical methods of optimization* (2nd ed.), New York: John Wiley & Sons, ISBN 978-0-471-91547-8.
- Nocedal, Jorge & Wright, Stephen J. (1999), *Numerical Optimization*, Springer-Verlag, ISBN 0-387-98793-2

Chapter 12

Symmetric rank-one

The **Symmetric Rank 1 (SR1)** method is a quasi-Newton method to update the second derivative (Hessian) based on the derivatives (gradients) calculated at two points. It is a generalization to the **secant method** for a multidimensional problem. This update maintains the *symmetry* of the matrix but does *not* guarantee that the update be *positive definite*.

The sequence of Hessian approximations generated by the SR1 method converges to the true Hessian under mild conditions, in theory; in practice, the approximate Hessians generated by the SR1 method show faster progress towards the true Hessian than do popular alternatives (BFGS or DFP), in preliminary numerical experiments.^[1] The SR1 method has computational advantages for sparse or partially separable problems.

A twice continuously differentiable function $x \mapsto f(x)$ has a gradient (∇f) and Hessian matrix B : The function f has an expansion as a Taylor series at x_0 , which can be truncated

$$f(x_0 + \Delta x) = f(x_0) + \nabla f(x_0)^T \Delta x + \frac{1}{2} \Delta x^T B \Delta x$$

its gradient has a Taylor-series approximation also

$$\nabla f(x_0 + \Delta x) = \nabla f(x_0) + B \Delta x$$

which is used to update B . The above secant-equation need not have a unique solution B . The SR1 formula computes (via an update of **rank 1**) the symmetric solution that is closest to the current approximate-value B_k :

$$B_{k+1} = B_k + \frac{(y_k - B_k \Delta x_k)(y_k - B_k \Delta x_k)^T}{(y_k - B_k \Delta x_k)^T \Delta x_k}$$

where

$$y_k = \nabla f(x_k + \Delta x_k) - \nabla f(x_k)$$

The corresponding update to the approximate inverse-Hessian $H_k = B_k^{-1}$ is

$$H_{k+1} = H_k + \frac{(\Delta x_k - H_k y_k)(\Delta x_k - H_k y_k)^T}{(\Delta x_k - H_k y_k)^T y_k}$$

The SR1 formula has been rediscovered a number of times. A drawback is that the denominator can vanish. Some authors have suggested that the update be applied only if

$$|\Delta x_k^T (y_k - B_k \Delta x_k)| \geq r \|\Delta x_k\| \cdot \|y_k - B_k \Delta x_k\|$$

where $r \in (0, 1)$ is a small number, e.g. 10^{-8} .^[2]

12.1 See also

- Quasi-Newton method
- Newton's method in optimization
- Broyden-Fletcher-Goldfarb-Shanno (BFGS) method
- L-BFGS method

12.2 Notes

[1] Conn, Gould & Toint (1991)

[2] Nocedal & Wright (1999)

12.3 References

- Byrd, Richard H. (1996) Analysis of a Symmetric Rank-One Trust Region Method. *SIAM Journal on Optimization* 6(4)
- Conn, A. R.; Gould, N. I. M.; Toint, Ph. L. (March 1991). “Convergence of quasi-Newton matrices generated by the symmetric rank one update”. *Mathematical Programming* (Springer Berlin/ Heidelberg) **50** (1): 177–195. doi:10.1007/BF01594934. ISSN 0025-5610. PDF file at Nick Gould’s website.
- Khalfan, H. Faye (1993) A Theoretical and Experimental Study of the Symmetric Rank-One Update. *SIAM Journal on Optimization* 3(1)
- Nocedal, Jorge & Wright, Stephen J. (1999). *Numerical Optimization*. Springer-Verlag. ISBN 0-387-98793-2.

Chapter 13

Gauss–Newton algorithm

The **Gauss–Newton algorithm** is a method used to solve non-linear least squares problems. It is a modification of **Newton’s method** for finding a **minimum** of a **function**. Unlike Newton’s method, the Gauss–Newton algorithm can only be used to minimize a sum of squared function values, but it has the advantage that second derivatives, which can be challenging to compute, are not required.

Non-linear least squares problems arise for instance in **non-linear regression**, where parameters in a model are sought such that the model is in good agreement with available observations.

The method is named after the mathematicians **Carl Friedrich Gauss** and **Isaac Newton**.

which is a direct generalization of **Newton’s method** in one dimension.

In data fitting, where the goal is to find the parameters β such that a given model function $y = f(x, \beta)$ best fits some data points (x_i, y_i) , the functions r_i are the **residuals**

$$r_i(\beta) = y_i - f(x_i, \beta).$$

Then, the Gauss–Newton method can be expressed in terms of the Jacobian \mathbf{J}_f of the function f as

$$\beta^{(s+1)} = \beta^{(s)} + (\mathbf{J}_f^\top \mathbf{J}_f)^{-1} \mathbf{J}_f^\top \mathbf{r}(\beta^{(s)}).$$

13.1 Description

Given m functions $\mathbf{r} = (r_1, \dots, r_m)$ of n variables $\beta = (\beta_1, \dots, \beta_n)$, with $m \geq n$, the Gauss–Newton algorithm iteratively finds the minimum of the sum of squares^[1]

$$S(\beta) = \sum_{i=1}^m r_i(\beta)^2.$$

Starting with an initial guess $\beta^{(0)}$ for the minimum, the method proceeds by the iterations

$$\beta^{(s+1)} = \beta^{(s)} - (\mathbf{J}_r^\top \mathbf{J}_r)^{-1} \mathbf{J}_r^\top \mathbf{r}(\beta^{(s)})$$

where, if \mathbf{r} and β are column vectors, the entries of the Jacobian matrix are

$$(\mathbf{J}_r)_{ij} = \frac{\partial r_i(\beta^{(s)})}{\partial \beta_j},$$

and the symbol \top denotes the matrix transpose.

If $m = n$, the iteration simplifies to

$$\beta^{(s+1)} = \beta^{(s)} - (\mathbf{J}_r)^{-1} \mathbf{r}(\beta^{(s)})$$

13.2 Notes

The assumption $m \geq n$ in the algorithm statement is necessary, as otherwise the matrix $\mathbf{J}_r^\top \mathbf{J}_r$ is not invertible and the normal equations cannot be solved (at least uniquely).

The Gauss–Newton algorithm can be derived by linearly approximating the vector of functions r_i . Using Taylor’s theorem, we can write at every iteration:

$$\mathbf{r}(\beta) \approx \mathbf{r}(\beta^s) + \mathbf{J}_r(\beta^s) \Delta$$

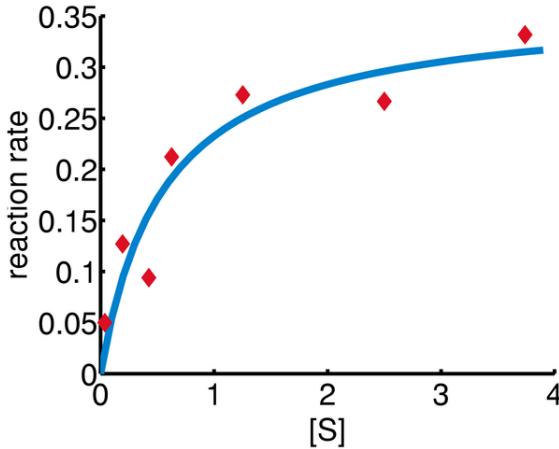
with $\Delta = \beta - \beta^s$. The task of finding Δ minimizing the sum of squares of the right-hand side, i.e.,

$$\min \|\mathbf{r}(\beta^s) + \mathbf{J}_r(\beta^s) \Delta\|_2^2$$

is a **linear least squares** problem, which can be solved explicitly, yielding the normal equations in the algorithm.

The normal equations are m linear simultaneous equations in the unknown increments, Δ . They may be solved in one step, using **Cholesky decomposition**, or, better, the **QR factorization** of \mathbf{J}_r . For large systems, an **iterative method**, such as the **conjugate gradient** method, may be more efficient. If there is a linear dependence between columns of \mathbf{J}_r , the iterations will fail as $\mathbf{J}_r^\top \mathbf{J}_r$ becomes singular.

13.3 Example



Calculated curve obtained with $\hat{\beta}_1 = 0.362$ and $\hat{\beta}_2 = 0.556$ (in blue) versus the observed data (in red).

In this example, the Gauss–Newton algorithm will be used to fit a model to some data by minimizing the sum of squares of errors between the data and model's predictions.

In a biology experiment studying the relation between substrate concentration $[S]$ and reaction rate in an enzyme-mediated reaction, the data in the following table were obtained.

It is desired to find a curve (model function) of the form

$$\text{rate} = \frac{V_{\max}[S]}{K_M + [S]}$$

that fits best the data in the least squares sense, with the parameters V_{\max} and K_M to be determined.

Denote by x_i and y_i the value of $[S]$ and the rate from the table, $i = 1, \dots, 7$. Let $\beta_1 = V_{\max}$ and $\beta_2 = K_M$. We will find β_1 and β_2 such that the sum of squares of the residuals

$$r_i = y_i - \frac{\beta_1 x_i}{\beta_2 + x_i} \quad (i = 1, \dots, 7)$$

is minimized.

The Jacobian \mathbf{J}_r of the vector of residuals r_i in respect to the unknowns β_j is an 7×2 matrix with the i -th row having the entries

$$\frac{\partial r_i}{\partial \beta_1} = -\frac{x_i}{\beta_2 + x_i}, \quad \frac{\partial r_i}{\partial \beta_2} = \frac{\beta_1 x_i}{(\beta_2 + x_i)^2}.$$

Starting with the initial estimates of $\beta_1 = 0.9$ and $\beta_2 = 0.2$, after five iterations of the Gauss–Newton algorithm the

optimal values $\hat{\beta}_1 = 0.362$ and $\hat{\beta}_2 = 0.556$ are obtained. The sum of squares of residuals decreased from the initial value of 1.445 to 0.00784 after the fifth iteration. The plot in the figure on the right shows the curve determined by the model for the optimal parameters versus the observed data.

13.4 Convergence properties

It can be shown^[2] that the increment Δ is a descent direction for S , and, if the algorithm converges, then the limit is a stationary point of S . However, convergence is not guaranteed, not even local convergence as in Newton's method.

The rate of convergence of the Gauss–Newton algorithm can approach quadratic.^[3] The algorithm may converge slowly or not at all if the initial guess is far from the minimum or the matrix $\mathbf{J}_r^T \mathbf{J}_r$ is ill-conditioned. For example, consider the problem with $m = 2$ equations and $n = 1$ variable, given by

$$\begin{aligned} r_1(\beta) &= \beta + 1 \\ r_2(\beta) &= \lambda \beta^2 + \beta - 1. \end{aligned}$$

The optimum is at $\beta = 0$. (Actually the optimum is at $\beta = -1$ for $\lambda = 2$, because $S(0) = 1^2 + (-1)^2 = 2$, but $S(-1) = 0$.) If $\lambda = 0$ then the problem is in fact linear and the method finds the optimum in one iteration. If $|\lambda| < 1$ then the method converges linearly and the error decreases asymptotically with a factor $|\lambda|$ at every iteration. However, if $|\lambda| > 1$, then the method does not even converge locally.^[4]

13.5 Derivation from Newton's method

In what follows, the Gauss–Newton algorithm will be derived from Newton's method for function optimization via an approximation. As a consequence, the rate of convergence of the Gauss–Newton algorithm can be quadratic under certain regularity conditions. In general (under weaker conditions), the convergence rate is linear.^[5]

The recurrence relation for Newton's method for minimizing a function S of parameters, β , is

$$\beta^{(s+1)} = \beta^{(s)} - \mathbf{H}^{-1} \mathbf{g}$$

where \mathbf{g} denotes the gradient vector of S and \mathbf{H} denotes the Hessian matrix of S . Since $S = \sum_{i=1}^m r_i^2$, the gradient is given by

$$g_j = 2 \sum_{i=1}^m r_i \frac{\partial r_i}{\partial \beta_j}.$$

Elements of the Hessian are calculated by differentiating the gradient elements, g_j , with respect to β_k

$$H_{jk} = 2 \sum_{i=1}^m \left(\frac{\partial r_i}{\partial \beta_j} \frac{\partial r_i}{\partial \beta_k} + r_i \frac{\partial^2 r_i}{\partial \beta_j \partial \beta_k} \right).$$

The Gauss–Newton method is obtained by ignoring the second-order derivative terms (the second term in this expression). That is, the Hessian is approximated by

$$H_{jk} \approx 2 \sum_{i=1}^m J_{ij} J_{ik}$$

where $J_{ij} = \frac{\partial r_i}{\partial \beta_j}$ are entries of the Jacobian \mathbf{J}_r . The gradient and the approximate Hessian can be written in matrix notation as

$$\mathbf{g} = 2\mathbf{J}_r^T \mathbf{r}, \quad \mathbf{H} \approx 2\mathbf{J}_r^T \mathbf{J}_r.$$

These expressions are substituted into the recurrence relation above to obtain the operational equations

$$\boldsymbol{\beta}^{(s+1)} = \boldsymbol{\beta}^{(s)} + \Delta; \quad \Delta = -(\mathbf{J}_r^T \mathbf{J}_r)^{-1} \mathbf{J}_r^T \mathbf{r}.$$

Convergence of the Gauss–Newton method is not guaranteed in all instances. The approximation

$$\left| r_i \frac{\partial^2 r_i}{\partial \beta_j \partial \beta_k} \right| \ll \left| \frac{\partial r_i}{\partial \beta_j} \frac{\partial r_i}{\partial \beta_k} \right|$$

that needs to hold to be able to ignore the second-order derivative terms may be valid in two cases, for which convergence is to be expected.^[6]

1. The function values r_i are small in magnitude, at least around the minimum.
2. The functions are only “mildly” non linear, so that $\frac{\partial^2 r_i}{\partial \beta_j \partial \beta_k}$ is relatively small in magnitude.

13.6 Improved versions

With the Gauss–Newton method the sum of squares S may not decrease at every iteration. However, since Δ is a descent direction, unless $S(\boldsymbol{\beta}^s)$ is a stationary point, it holds that $S(\boldsymbol{\beta}^s + \alpha \Delta) < S(\boldsymbol{\beta}^s)$ for all sufficiently small $\alpha > 0$. Thus, if divergence occurs, one solution is to

employ a fraction, α , of the increment vector, Δ in the updating formula

$$\boldsymbol{\beta}^{s+1} = \boldsymbol{\beta}^s + \alpha \Delta$$

In other words, the increment vector is too long, but it points in “downhill”, so going just a part of the way will decrease the objective function S . An optimal value for α can be found by using a line search algorithm, that is, the magnitude of α is determined by finding the value that minimizes S , usually using a direct search method in the interval $0 < \alpha < 1$.

In cases where the direction of the shift vector is such that the optimal fraction, α , is close to zero, an alternative method for handling divergence is the use of the Levenberg–Marquardt algorithm, also known as the “trust region method”.^[1] The normal equations are modified in such a way that the increment vector is rotated towards the direction of steepest descent,

$$(\mathbf{J}^T \mathbf{J} + \lambda \mathbf{D}) \Delta = -\mathbf{J}^T \mathbf{r}$$

where \mathbf{D} is a positive diagonal matrix. Note that when D is the identity matrix and $\lambda \rightarrow +\infty$, then $\Delta/\lambda \rightarrow -\mathbf{J}^T \mathbf{r}$, therefore the direction of Δ approaches the direction of the negative gradient $-\mathbf{J}^T \mathbf{r}$.

The so-called Marquardt parameter, λ , may also be optimized by a line search, but this is inefficient as the shift vector must be re-calculated every time λ is changed. A more efficient strategy is this. When divergence occurs increase the Marquardt parameter until there is a decrease in S . Then, retain the value from one iteration to the next, but decrease it if possible until a cut-off value is reached when the Marquardt parameter can be set to zero; the minimization of S then becomes a standard Gauss–Newton minimization.

13.7 Other applications

The Gauss–Newton algorithm is a popular method for solving nonlinear inverse problems. A particular application is generating computational models of oil and gas reservoirs for consistency with observed production data.^[7]

13.8 Related algorithms

In a quasi-Newton method, such as that due to Davidon, Fletcher and Powell or Broyden–Fletcher–Goldfarb–Shanno (BFGS method) an estimate of the full Hessian, $\frac{\partial^2 S}{\partial \beta_j \partial \beta_k}$, is built up numerically using first derivatives $\frac{\partial r_i}{\partial \beta_j}$ only so that after n refinement cycles the method closely

approximates to Newton's method in performance. Note that quasi-Newton methods can minimize general real-valued functions, whereas Gauss-Newton, Levenberg-Marquardt, etc. fits only to nonlinear least-squares problems.

Another method for solving minimization problems using only first derivatives is gradient descent. However, this method does not take into account the second derivatives even approximately. Consequently, it is highly inefficient for many functions, especially if the parameters have strong interactions.

13.9 Notes

- [1] Björck (1996)
- [2] Björck (1996) p260
- [3] Björck (1996) p341, 342
- [4] Fletcher (1987) p.113
- [5] <http://www.henley.ac.uk/web/FILES/mathematics/09-04.pdf>
- [6] Nocedal (1997)
- [7] History matching production data and uncertainty assessment with an efficient TSVD parameterization algorithm, Journal of Petroleum Science and Engineering, <http://www.sciencedirect.com/science/article/pii/S0920410513003227>

13.10 References

- Björck, A. (1996). *Numerical methods for least squares problems*. SIAM, Philadelphia. ISBN 0-89871-360-9.
- Fletcher, Roger (1987). *Practical methods of optimization* (2nd ed.). New York: John Wiley & Sons. ISBN 978-0-471-91547-8..
- Nocedal, Jorge; Wright, Stephen (1999). *Numerical optimization*. New York: Springer. ISBN 0-387-98793-2.

Chapter 14

Gradient descent

For the analytical method called “steepest descent”, see Method of steepest descent.

Gradient descent is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the *negative* of the gradient (or of the approximate gradient) of the function at the current point. If instead one takes steps proportional to the *positive* of the gradient, one approaches a local maximum of that function; the procedure is then known as **gradient ascent**.

Gradient descent is also known as **steepest descent**, or the **method of steepest descent**. When known as the latter, gradient descent should not be confused with the method of steepest descent for approximating integrals.

14.1 Description

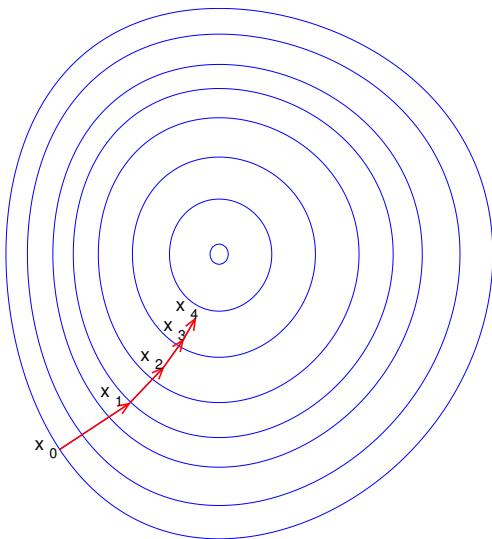


Illustration of gradient descent.

Gradient descent is based on the observation that if the multivariable function $F(\mathbf{x})$ is defined and differentiable in a neighborhood of a point \mathbf{a} , then $F(\mathbf{x})$ decreases

fastest if one goes from \mathbf{a} in the direction of the negative gradient of F at \mathbf{a} , $-\nabla F(\mathbf{a})$. It follows that, if

$$\mathbf{b} = \mathbf{a} - \gamma \nabla F(\mathbf{a})$$

for γ small enough, then $F(\mathbf{a}) \geq F(\mathbf{b})$. With this observation in mind, one starts with a guess \mathbf{x}_0 for a local minimum of F , and considers the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots$ such that

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n), \quad n \geq 0.$$

We have

$$F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \dots,$$

so hopefully the sequence (\mathbf{x}_n) converges to the desired local minimum. Note that the value of the *step size* γ is allowed to change at every iteration. With certain assumptions on the function F (for example, F convex and ∇F Lipschitz) and particular choices of γ (e.g., chosen via a line search that satisfies the **Wolfe conditions**), convergence to a local minimum can be guaranteed. When the function F is convex, all local minima are also global minima, so in this case gradient descent can converge to the global solution.

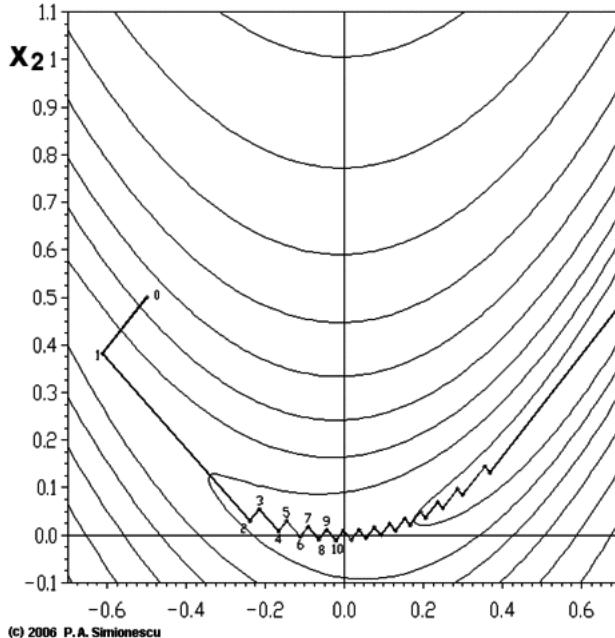
This process is illustrated in the picture to the right. Here F is assumed to be defined on the plane, and that its graph has a bowl shape. The blue curves are the **contour lines**, that is, the regions on which the value of F is constant. A red arrow originating at a point shows the direction of the negative gradient at that point. Note that the (negative) gradient at a point is **orthogonal** to the contour line going through that point. We see that gradient *descent* leads us to the bottom of the bowl, that is, to the point where the value of the function F is minimal.

14.1.1 Examples

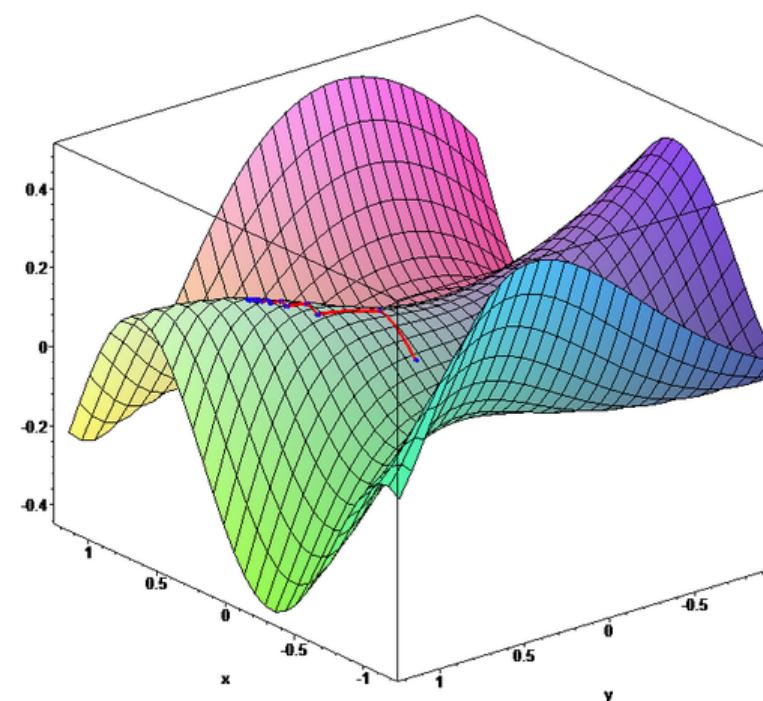
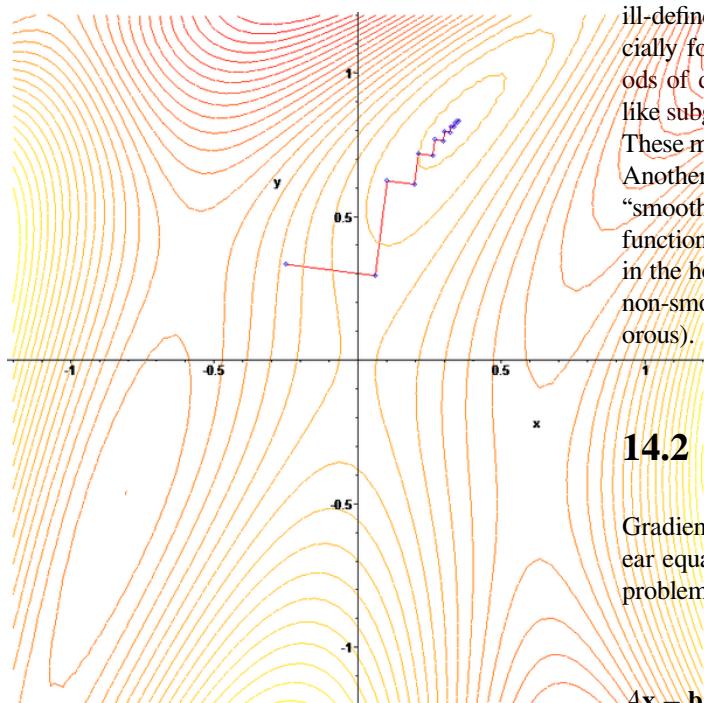
Gradient descent has problems with pathological functions such as the **Rosenbrock function** shown here.

$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2.$$

The Rosenbrock function has a narrow curved valley which contains the minimum. The bottom of the valley is very flat. Because of the curved flat valley the optimization is zig-zagging slowly with small stepsizes towards the minimum.



The “Zig-Zagging” nature of the method is also evident below, where the gradient ascent method is applied to $F(x, y) = \sin(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3) \cos(2x + 1 - e^y)$.



14.1.2 Limitations

For some of the above examples, gradient descent is relatively slow close to the minimum: technically, its asymptotic rate of convergence is inferior to many other methods.^[8] For poorly conditioned convex problems, gradient descent increasingly ‘zigzags’ as the gradients point nearly orthogonally to the shortest direction to a minimum point. For more details, see the comments below.

For non-differentiable functions, gradient methods are ill-defined. For locally Lipschitz problems and especially for convex minimization problems, bundle methods of descent are well-defined. Non-descent methods, like subgradient projection methods, may also be used.^[1] These methods are typically slower than gradient descent. Another alternative for non-differentiable functions is to “smooth” the function, or bound the function by a smooth function. In this approach, the smooth problem is solved in the hope that the answer is close to the answer for the non-smooth problem (occasionally, this can be made rigorous).

14.2 Solution of a linear system

Gradient descent can be used to solve a system of linear equations, reformulated as a quadratic minimization problem, e.g., using linear least squares. Solution of

$$Ax - b = 0$$

in the sense of linear least squares is defined as minimizing the function

$$F(x) = \|Ax - b\|^2.$$

In traditional linear least squares for real A and b the Euclidean norm is used, in which case

$$\nabla F(\mathbf{x}) = 2A^T(A\mathbf{x} - \mathbf{b}).$$

In this case, the line search minimization, finding the locally optimal step size γ on every iteration, can be performed analytically, and explicit formulas for the locally optimal γ are known.^[2]

For solving linear equations, gradient descent is rarely used, with the conjugate gradient method being one of the most popular alternatives. The speed of convergence of gradient descent depends on the maximal and minimal eigenvalues of A , while the speed of convergence of conjugate gradients has a more complex dependence on the eigenvalues, and can benefit from preconditioning. Gradient descent also benefits from preconditioning, but this is not done as commonly.

14.3 Solution of a non-linear system

Gradient descent can also be used to solve a system of nonlinear equations. Below is an example that shows how to use the gradient descent to solve for three unknown variables, x_1 , x_2 , and x_3 . This example shows one iteration of the gradient descent.

Consider a nonlinear system of equations:

$$\begin{cases} 3x_1 - \cos(x_2 x_3) - \frac{3}{2} = 0 \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 = 0 \\ \exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3} = 0 \end{cases}$$

suppose we have the function

$$G(\mathbf{x}) = \begin{bmatrix} 3x_1 - \cos(x_2 x_3) - \frac{3}{2} \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 \\ \exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3} \end{bmatrix}$$

where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

and the objective function

$$\begin{aligned} F(\mathbf{x}) &= \frac{1}{2} G^T(\mathbf{x}) G(\mathbf{x}) \\ &= \frac{1}{2} ((3x_1 - \cos(x_2 x_3) - \frac{3}{2})^2 + (4x_1^2 - 625x_2^2 + 2x_2 - 1)^2 + (\exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3})^2) \end{aligned}$$

With initial guess

$$\mathbf{x}^{(0)} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

We know that

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \gamma_0 \nabla F(\mathbf{x}^{(0)})$$

where

$$\nabla F(\mathbf{x}^{(0)}) = J_G(\mathbf{x}^{(0)})^T G(\mathbf{x}^{(0)})$$

The Jacobian matrix $J_G(\mathbf{x}^{(0)})$

$$J_G = \begin{bmatrix} 3 & \sin(x_2 x_3)x_3 & \sin(x_2 x_3)x_2 \\ 8x_1 & -1250x_2 + 2 & 0 \\ -x_2 \exp(-x_1 x_2) & -x_1 \exp(-x_1 x_2) & 20 \end{bmatrix}$$

Then evaluating these terms at $\mathbf{x}^{(0)}$

$$J_G(\mathbf{x}^{(0)}) = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 20 \end{bmatrix}$$

and

$$G(\mathbf{x}^{(0)}) = \begin{bmatrix} -2.5 \\ -1 \\ 10.472 \end{bmatrix}$$

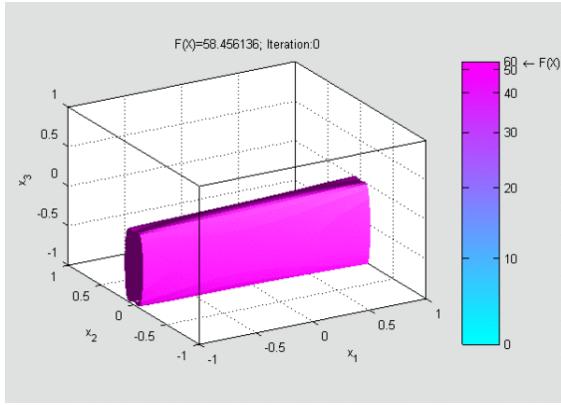
So that

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \gamma_0 \begin{bmatrix} -7.5 \\ -2 \\ 209.44 \end{bmatrix}.$$

and

$$F(\mathbf{x}^{(0)}) = 0.5((-2.5)^2 + (-1)^2 + (10.472)^2) = 58.456$$

Now a suitable γ_0 must be found such that $F(\mathbf{x}^{(1)}) \leq F(\mathbf{x}^{(0)})$. This can be done with any of a variety of line search algorithms. One might also simply guess $\gamma_0 = 0.001$ which gives



An animation showing the first 83 iterations of gradient descent applied to this example. Surfaces are isosurfaces of $F(\mathbf{x}^{(n)})$ at current guess $\mathbf{x}^{(n)}$, and arrows show the direction of descent. Due to a small and constant step size, the convergence is slow.

$$\mathbf{x}^{(1)} = \begin{bmatrix} 0.0075 \\ 0.002 \\ -0.20944 \end{bmatrix}$$

evaluating at this value,

$$F(\mathbf{x}^{(1)}) = 0.5((-2.48)^2 + (-1.00)^2 + (6.28)^2) = 23.306$$

The decrease from $F(\mathbf{x}^{(0)}) = 58.456$ to the next step's value of $F(\mathbf{x}^{(1)}) = 23.306$ is a sizable decrease in the objective function. Further steps would reduce its value until a solution to the system was found.

14.4 Comments

Gradient descent works in spaces of any number of dimensions, even in infinite-dimensional ones. In the latter case the search space is typically a **function space**, and one calculates the **Gâteaux derivative** of the functional to be minimized to determine the descent direction.^[3]

The gradient descent can take many iterations to compute a local minimum with a required **accuracy**, if the **curvature** in different directions is very different for the given function. For such functions, **preconditioning**, which changes the geometry of the space to shape the function level sets like **concentric circles**, cures the slow convergence. Constructing and applying preconditioning can be computationally expensive, however.

The gradient descent can be combined with a **line search**, finding the locally optimal step size γ on every iteration. Performing the line search can be time-consuming. Conversely, using a fixed small γ can yield poor convergence.

Methods based on **Newton's method** and inversion of the **Hessian** using **conjugate gradient** techniques can be better alternatives.^{[4][5]} Generally, such methods converge in

fewer iterations, but the cost of each iteration is higher. An example is the **BFGS method** which consists in calculating on every step a matrix by which the gradient vector is multiplied to go into a “better” direction, combined with a more sophisticated **line search** algorithm, to find the “best” value of γ . For extremely large problems, where the computer memory issues dominate, a limited-memory method such as **L-BFGS** should be used instead of BFGS or the steepest descent.

Gradient descent can be viewed as Euler's method for solving ordinary differential equations $x'(t) = -\nabla f(x(t))$ of a gradient flow.

14.5 A computational example

The gradient descent algorithm is applied to find a local minimum of the function $f(x) = x^4 - 3x^3 + 2$, with derivative $f'(x) = 4x^3 - 9x^2$. Here is an implementation in the Python programming language.

```
# From calculation, we expect that the local minimum occurs at x=9/4
x_old = 0
x_new = 6 # The algorithm starts at x=6
eps = 0.01 # step size precision = 0.00001
def f_derivative(x):
    return 4 * x**3 - 9 * x**2
while abs(x_new - x_old) > precision:
    x_old = x_new
    x_new = x_old - eps * f_derivative(x_old)
print("Local minimum occurs at", x_new)
```

The above piece of code has to be modified with regard to step size according to the system at hand and convergence can be made faster by using an adaptive step size. In the above case the step size is not adaptive. It stays at 0.01 in all the directions which can sometimes cause the method to fail by diverging from the minimum.

14.6 Extensions

Gradient descent can be extended to handle **constraints** by including a **projection** onto the set of constraints. This method is only feasible when the projection is efficiently computable on a computer. Under suitable assumptions, this method converges. This method is a specific case of the **forward-backward algorithm** for monotone inclusions (which includes **convex programming** and **variational inequalities**).^[6]

14.6.1 Fast proximal gradient method

Another extension of gradient descent is due to Yurii Nesterov from 1983,^[7] and has been subsequently generalized. He provides a simple modification of the algorithm that enables faster convergence for convex problems. Specifically, if the function F is convex and ∇F is Lipschitz, and it is not assumed that F is strongly con-

vex, then the error in the objective value generated at each step k by the gradient descent method will be bounded by $\mathcal{O}(1/k)$. Using the Nesterov acceleration technique, the error decreases at $\mathcal{O}(1/k^2)$.^[8]

14.6.2 The momentum method

Yet another extension, that reduces the risk of getting stuck in a local minimum, as well as speeds up the convergence considerably in cases where the process would otherwise zig-zag heavily, is the *momentum method*, which uses a momentum term in analogy to “the mass of Newtonian particles that move through a viscous medium in a conservative force field”.^[9] This method is often used as an extension to the backpropagation algorithms used to train artificial neural networks.^{[10][11]}

14.7 See also

- Conjugate gradient method
- Stochastic gradient descent
- Rprop
- Delta rule
- Wolfe conditions
- Preconditioning
- BFGS method
- Nelder–Mead method
- Extreme Learning Machines

14.8 References

- Mordecai Avriel (2003). *Nonlinear Programming: Analysis and Methods*. Dover Publishing. ISBN 0-486-43227-0.
- Jan A. Snyman (2005). *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Springer Publishing. ISBN 0-387-24348-8
- Cauchy, Augustin (1847). *Méthode générale pour la résolution des systèmes d'équations simultanées*. pp. 536–538.

[1] Kiwiel, Krzysztof C. (2001). “Convergence and efficiency of subgradient methods for quasiconvex minimization”. *Mathematical Programming (Series A)* **90** (1) (Berlin, Heidelberg: Springer). pp. 1–25. doi:10.1007/PL00011414. ISSN 0025-5610. MR 1819784.

- [2] Yuan, Ya-xiang (1999). “Step-sizes for the gradient method”. *AMS/IP Studies in Advanced Mathematics* (Providence, RI: American Mathematical Society) **42** (2): 785.
- [3] G. P. Akilov, L. V. Kantorovich, Functional Analysis, Pergamon Pr; 2 Sub edition,ISBN 0-08-023036-9, 1982
- [4] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical Recipes in C: The Art of Scientific Computing, 2nd Ed., Cambridge University Press, New York, 1992
- [5] T. Strutz: *Data Fitting and Uncertainty (A practical introduction to weighted least squares and beyond)*. Vieweg+Teubner, Wiesbaden 2011, ISBN 978-3-8348-1022-9.
- [6] P. L. Combettes and J.-C. Pesquet, “Proximal splitting methods in signal processing”, in: *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, (H. H. Bauschke, R. S. Burachik, P. L. Combettes, V. Elser, D. R. Luke, and H. Wolkowicz, Editors), pp. 185–212. Springer, New York, 2011.
- [7] Yu. Nesterov, “Introductory Lectures on Convex Optimization. A Basic Course” (Springer, 2004, ISBN 1-4020-7553-7)
- [8] Fast Gradient Methods, lecture notes by Prof. Lieven Vandenberghe for EE236C at UCLA
- [9] Qian, Ning (January 1999). “On the momentum term in gradient descent learning algorithms”. *Neural Networks* **12** (1): 145–151. Retrieved 17 October 2014.
- [10] “Momentum and Learning Rate Adaptation”. Willamette University. Retrieved 17 October 2014.
- [11] Geoffrey Hinton; Nitish Srivastava; Kevin Swersky. “6 - 3 - The momentum method”. *YouTube*. Retrieved 18 October 2014. Part of a lecture series for the Coursera online course Neural Networks for Machine Learning.

14.9 External links

- Interactive examples of gradient descent and some step size selection methods
- Using gradient descent in C++, Boost, Ublas for linear regression

Chapter 15

Levenberg–Marquardt algorithm

In mathematics and computing, the **Levenberg–Marquardt algorithm (LMA)**,^[1] also known as the **damped least-squares (DLS)** method, is used to solve non-linear least squares problems. These minimization problems arise especially in least squares curve fitting.

The LMA interpolates between the Gauss–Newton algorithm (GNA) and the method of gradient descent. The LMA is more robust than the GNA, which means that in many cases it finds a solution even if it starts very far off the final minimum. For well-behaved functions and reasonable starting parameters, the LMA tends to be a bit slower than the GNA. LMA can also be viewed as Gauss–Newton using a trust region approach.

The LMA is a very popular curve-fitting algorithm used in many software applications for solving generic curve-fitting problems. However, as for many fitting algorithms, the LMA finds only a local minimum, which is not necessarily the global minimum.

15.1 The problem

The primary application of the Levenberg–Marquardt algorithm is in the least squares curve fitting problem: given a set of m empirical datum pairs of independent and dependent variables, (x_i, y_i) , optimize the parameters β of the model curve $f(x, \beta)$ so that the sum of the squares of the deviations

$$S(\beta) = \sum_{i=1}^m [y_i - f(x_i, \beta)]^2$$

becomes minimal.

15.2 The solution

Like other numeric minimization algorithms, the Levenberg–Marquardt algorithm is an iterative procedure. To start a minimization, the user has to provide an initial guess for the parameter vector, β . In cases with only one minimum, an uninformed standard guess

like $\beta^T = (1, 1, \dots, 1)$ will work fine; in cases with multiple minima, the algorithm converges to the global minimum only if the initial guess is already somewhat close to the final solution.

In each iteration step, the parameter vector, β , is replaced by a new estimate, $\beta + \delta$. To determine δ , the functions $f(x_i, \beta + \delta)$ are approximated by their linearizations

$$f(x_i, \beta + \delta) \approx f(x_i, \beta) + J_i \delta$$

where

$$J_i = \frac{\partial f(x_i, \beta)}{\partial \beta}$$

is the gradient (row-vector in this case) of f with respect to β .

At the minimum of the sum of squares, $S(\beta)$, the gradient of S with respect to δ will be zero. The above first-order approximation of $f(x_i, \beta + \delta)$ gives

$$S(\beta + \delta) \approx \sum_{i=1}^m (y_i - f(x_i, \beta) - J_i \delta)^2$$

Or in vector notation,

$$S(\beta + \delta) \approx \|y - f(\beta) - J\delta\|^2$$

Taking the derivative with respect to δ and setting the result to zero gives:

$$(J^T J)\delta = J^T[y - f(\beta)]$$

where J is the Jacobian matrix whose i^{th} row equals J_i , and where f and y are vectors with i^{th} component $f(x_i, \beta)$ and y_i , respectively. This is a set of linear equations which can be solved for δ .

Levenberg's contribution is to replace this equation by a “damped version”,

$$(\mathbf{J}^T \mathbf{J} + \lambda \mathbf{I})\boldsymbol{\delta} = \mathbf{J}^T[\mathbf{y} - \mathbf{f}(\boldsymbol{\beta})]$$

where \mathbf{I} is the identity matrix, giving as the increment, $\boldsymbol{\delta}$, to the estimated parameter vector, $\boldsymbol{\beta}$.

The (non-negative) damping factor, λ , is adjusted at each iteration. If reduction of S is rapid, a smaller value can be used, bringing the algorithm closer to the Gauss–Newton algorithm, whereas if an iteration gives insufficient reduction in the residual, λ can be increased, giving a step closer to the gradient descent direction. Note that the gradient of S with respect to $\boldsymbol{\delta}$ equals $-2(\mathbf{J}^T[\mathbf{y} - \mathbf{f}(\boldsymbol{\beta})])^T$. Therefore, for large values of λ , the step will be taken approximately in the direction of the gradient. If either the length of the calculated step, $\boldsymbol{\delta}$, or the reduction of sum of squares from the latest parameter vector, $\boldsymbol{\beta} + \boldsymbol{\delta}$, fall below predefined limits, iteration stops and the last parameter vector, $\boldsymbol{\beta}$, is considered to be the solution.

Levenberg's algorithm has the disadvantage that if the value of damping factor, λ , is large, inverting $\mathbf{J}^T \mathbf{J} + \lambda \mathbf{I}$ is not used at all. Marquardt provided the insight that we can scale each component of the gradient according to the curvature so that there is larger movement along the directions where the gradient is smaller. This avoids slow convergence in the direction of small gradient. Therefore, Marquardt replaced the identity matrix, \mathbf{I} , with the diagonal matrix consisting of the diagonal elements of $\mathbf{J}^T \mathbf{J}$, resulting in the Levenberg–Marquardt algorithm:

$$(\mathbf{J}^T \mathbf{J} + \lambda \text{diag}(\mathbf{J}^T \mathbf{J}))\boldsymbol{\delta} = \mathbf{J}^T[\mathbf{y} - \mathbf{f}(\boldsymbol{\beta})]$$

A similar damping factor appears in Tikhonov regularization, which is used to solve linear ill-posed problems, as well as in ridge regression, an estimation technique in statistics.

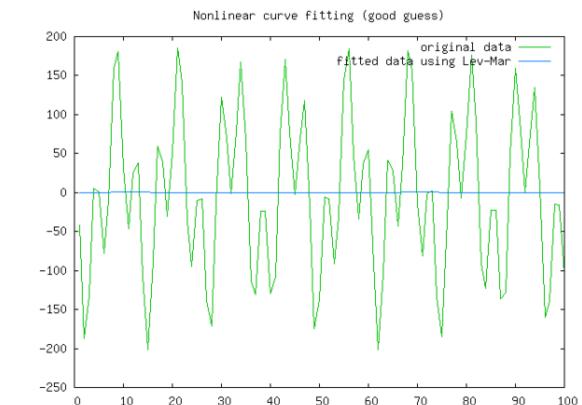
15.2.1 Choice of damping parameter

Various more-or-less heuristic arguments have been put forward for the best choice for the damping parameter λ . Theoretical arguments exist showing why some of these choices guarantee local convergence of the algorithm; however these choices can make the global convergence of the algorithm suffer from the undesirable properties of steepest-descent, in particular very slow convergence close to the optimum.

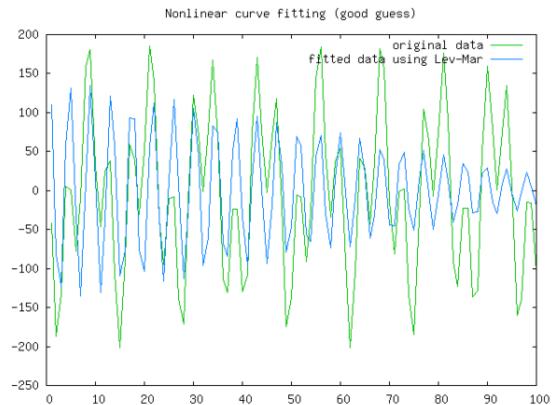
The absolute values of any choice depends on how well-scaled the initial problem is. Marquardt recommended starting with a value λ_0 and a factor $v > 1$. Initially setting $\lambda = \lambda_0$ and computing the residual sum of squares $S(\boldsymbol{\beta})$ after one step from the starting point with the damping factor of $\lambda = \lambda_0$ and secondly with λ_0/v . If both of these are worse than the initial point then the damping is increased by successive multiplication by v until a better point is found with a new damping factor of $\lambda_0 v^k$ for some k .

If use of the damping factor λ/v results in a reduction in squared residual then this is taken as the new value of λ (and the new optimum location is taken as that obtained with this damping factor) and the process continues; if using λ/v resulted in a worse residual, but using λ resulted in a better residual, then λ is left unchanged and the new optimum is taken as the value obtained with λ as damping factor.

15.3 Example

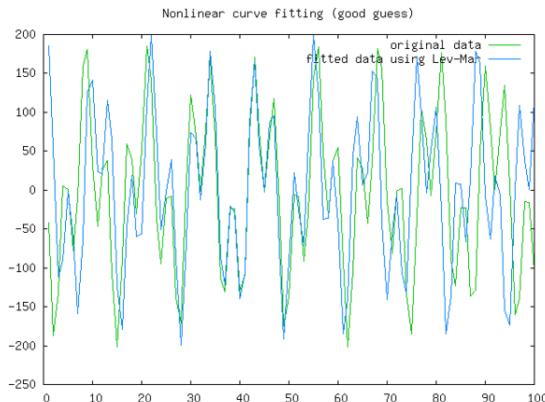


Poor fit



Better fit

In this example we try to fit the function $y = a \cos(bX) + b \sin(aX)$ using the Levenberg–Marquardt algorithm implemented in GNU Octave as the `leasqr` function. The 3 graphs Fig 1,2,3 show progressively better fitting for the parameters $a=100$, $b=102$ used in the initial curve. Only when the parameters in Fig 3 are chosen closest to the original, are the curves fitting exactly. This equation is an example of very sensitive initial conditions for the Levenberg–Marquardt algorithm. One reason for this sensitivity is the existence of multiple minima — the function $\cos(\beta x)$ has minima at parameter value $\hat{\beta}$ and $\hat{\beta} + 2n\pi$.



Best fit

15.4 See also

- Trust region
- Nelder–Mead method (aka simplex)

15.5 Notes

- [1] The algorithm was first published by Kenneth Levenberg, while working at the Frankford Army Arsenal. It was rediscovered by Donald Marquardt who worked as a statistician at DuPont and independently by Girard, Wynn and Morrison.

15.6 References

- Kenneth Levenberg (1944). “A Method for the Solution of Certain Non-Linear Problems in Least Squares”. *Quarterly of Applied Mathematics* **2**: 164–168.
- A. Girard (1958). *Rev. Opt* **37**: 225, 397. Missing or empty |title= (help)
- C.G. Wynn (1959). “Lens Designing by Electronic Digital Computer: I”. *Proc. Phys. Soc. London* **73** (5): 777. doi:10.1088/0370-1328/73/5/310.
- Jorge J. Moré and Daniel C. Sorensen (1983). “Computing a Trust-Region Step”. *SIAM J. Sci. Stat. Comput.* (4): 553–572.
- D.D. Morrison (1960). *Jet Propulsion Laboratory Seminar proceedings*. Missing or empty |title= (help)
- Donald Marquardt (1963). “An Algorithm for Least-Squares Estimation of Nonlinear Parameters”. *SIAM Journal on Applied Mathematics* **11** (2): 431–441. doi:10.1137/0111030.

- Philip E. Gill and Walter Murray (1978). “Algorithms for the solution of the nonlinear least-squares problem”. *SIAM Journal on Numerical Analysis* **15** (5): 977–992. doi:10.1137/0715063.
- Jose Pujol (2007). “The solution of nonlinear inverse problems and the Levenberg–Marquardt method”. *Geophysics (SEG)* **72** (4): W1–W16. doi:10.1190/1.2732552.
- Nocedal, Jorge; Wright, Stephen J. (2006). *Numerical Optimization*, 2nd Edition. Springer. ISBN 0-387-30303-0.

15.7 External links

15.7.1 Descriptions

- Detailed description of the algorithm can be found in *Numerical Recipes in C*, Chapter 15.5: Nonlinear models
- C. T. Kelley, *Iterative Methods for Optimization*, SIAM Frontiers in Applied Mathematics, no 18, 1999, ISBN 0-89871-433-8. Online copy
- History of the algorithm in SIAM news
- A tutorial by Ananth Ranganathan
- *Methods for Non-Linear Least Squares Problems* by K. Madsen, H.B. Nielsen, O. Tingleff is a tutorial discussing non-linear least-squares in general and the Levenberg–Marquardt method in particular
- T. Strutz: *Data Fitting and Uncertainty (A practical introduction to weighted least squares and beyond)*. Vieweg+Teubner, ISBN 978-3-8348-1022-9.
- H. P. Gavin, *The Levenberg–Marquardt method for nonlinear least squares curve-fitting problems* (MATLAB implementation included)

15.7.2 Implementations

- Levenberg–Marquardt is a built-in algorithm in Mathematica , Matlab, NeuroSolutions, GNU Octave, Origin, SciPy, Fityk, IGOR Pro and LabVIEW.
- The oldest implementation still in use is lmdif, from MINPACK, in Fortran, in the public domain. See also:
 - lmfit, a self-contained C implementation of the MINPACK algorithm, with an easy-to-use wrapper for curve fitting, liberal licence (freeBSD).

- **eigen**, a C++ linear algebra library, includes an adaptation of the minpack algorithm in the “NonLinearOptimization” module.
- The **GNU Scientific Library** has a C interface to MINPACK.
- **C/C++ Minpack** includes the Levenberg–Marquardt algorithm.
- Several high-level languages and mathematical packages have wrappers for the **MINPACK** routines, among them:
 - Python library `scipy`, module `scipy.optimize.leastsq`,
 - **IDL**, add-on **MPFIT**.
 - **R** (programming language) has the `minpack.lm` package.
- **levmar** is an implementation in C/C++ with support for constraints, distributed under the **GNU General Public License**.
 - levmar includes a **MEX** file interface for **MATLAB**
 - Perl (**PDL**), python, Haskell and .NET interfaces to levmar are available: see **PDL::Fit::Levmar** or **PDL::Fit::LM**, **PyLevmar**, **HackageDB levmar** and **LevmarSharp**.
- **sparseLM** is a C implementation aimed at minimizing functions with large, arbitrarily sparse Jacobians. Includes a MATLAB MEX interface.
- **SMarquardt.m** is a stand-alone routine for Matlab or Octave.
- **InMin** library contains a C++ implementation of the algorithm based on the **eigen** C++ linear algebra library. It has a pure C-language API as well as a Python binding
- **ceres** is a non-linear minimisation library with an implementation of the Levenberg–Marquardt algorithm. It is written in C++ and uses **eigen**
- **ALGLIB** has implementations of improved LMA in C# / C++ / Delphi / Visual Basic. Improved algorithm takes less time to converge and can use either Jacobian or exact Hessian.
- **NMath** has an implementation for the .NET Framework.
- **gnuplot** uses its own implementation **gnuplot.info**.
- Java programming language implementations: 1) **Javanumerics**, 2) **LMA-package** (a small, user friendly and well documented implementation with examples and support), 3) **Apache Commons Math**
- **OOoConv** implements the L-M algorithm as an OpenOffice.org Calc spreadsheet.
- **SAS**, there are multiple ways to access SAS’s implementation of the Levenberg–Marquardt algorithm: it can be accessed via **NLPLM Call** in **PROC IML** and it can also be accessed through the **LSQ** statement in **PROC NLP**, and the **METHOD=MARQUARDT** option in **PROC NLIN**.
- **LMAM/OLMAM** Matlab toolbox implements Levenberg–Marquardt with adaptive momentum for training feedforward neural networks.
- **GADfit** is a Fortran implementation of global fitting based on a modified Levenberg–Marquardt. Uses automatic differentiation. Allows fitting functions of arbitrary complexity, including integrals.

Chapter 16

Nonlinear conjugate gradient method

In numerical optimization, the **nonlinear conjugate gradient method** generalizes the conjugate gradient method to nonlinear optimization. For a quadratic function $f(x)$:

$$f(x) = \|Ax - b\|^2$$

The minimum of f is obtained when the gradient is 0:

$$\nabla_x f = 2A^\top(Ax - b) = 0$$

Whereas linear conjugate gradient seeks a solution to the linear equation $A^\top Ax = A^\top b$, the nonlinear conjugate gradient method is generally used to find the **local minimum** of a nonlinear function using its **gradient** $\nabla_x f$ alone. It works when the function is approximately quadratic near the minimum, which is the case when the function is twice differentiable at the minimum.

Given a function $f(x)$ of N variables to minimize, its gradient $\nabla_x f$ indicates the direction of maximum increase. One simply starts in the opposite (**steepest descent**) direction:

$$\Delta x_0 = -\nabla_x f(x_0)$$

with an adjustable step length α and performs a **line search** in this direction until it reaches the minimum of f :

$$\alpha_0 := \arg \min_{\alpha} f(x_0 + \alpha \Delta x_0)$$

$$x_1 = x_0 + \alpha_0 \Delta x_0$$

After this first iteration in the steepest direction Δx_0 , the following steps constitute one iteration of moving along a subsequent conjugate direction s_n , where $s_0 = \Delta x_0$:

1. Calculate the steepest direction: $\Delta x_n = -\nabla_x f(x_n)$,
2. Compute β_n according to one of the formulas below,
3. Update the conjugate direction: $s_n = \Delta x_n + \beta_n s_{n-1}$
4. Perform a line search: optimize $\alpha_n = \arg \min_{\alpha} f(x_n + \alpha s_n)$,
5. Update the position: $x_{n+1} = x_n + \alpha_n s_n$,

With a pure quadratic function the minimum is reached within N iterations (excepting roundoff error), but a non-quadratic function will make slower progress. Subsequent search directions lose conjugacy requiring the search direction to be reset to the steepest descent direction at least every N iterations, or sooner if progress stops. However, resetting every iteration turns the method into **steepest descent**. The algorithm stops when it finds the minimum, determined when no progress is made after a direction reset (i.e. in the steepest descent direction), or when some tolerance criterion is reached.

Within a linear approximation, the parameters α and β are the same as in the linear conjugate gradient method but have been obtained with line searches. The conjugate gradient method can follow narrow (**ill-conditioned**) valleys where the **steepest descent** method slows down and follows a criss-cross pattern.

Four of the best known formulas for β_n are named after their developers and are given by the following formulas:

- Fletcher–Reeves:

$$\beta_n^{FR} = \frac{\Delta x_n^\top \Delta x_n}{\Delta x_{n-1}^\top \Delta x_{n-1}}$$

- Polak–Ribi  re:

$$\beta_n^{PR} = \frac{\Delta x_n^\top (\Delta x_n - \Delta x_{n-1})}{\Delta x_{n-1}^\top \Delta x_{n-1}}$$

- Hestenes-Stiefel:

$$\beta_n^{HS} = -\frac{\Delta x_n^\top (\Delta x_n - \Delta x_{n-1})}{s_{n-1}^\top (\Delta x_n - \Delta x_{n-1})}$$

- Dai-Yuan:

$$\beta_n^{DY} = -\frac{\Delta x_n^\top \Delta x_n}{s_{n-1}^\top (\Delta x_n - \Delta x_{n-1})}$$

These formulas are equivalent for a quadratic function, but for nonlinear optimization the preferred formula is a matter of heuristics or taste. A popular choice is $\beta = \max\{0, \beta^{PR}\}$ which provides a direction reset automatically.

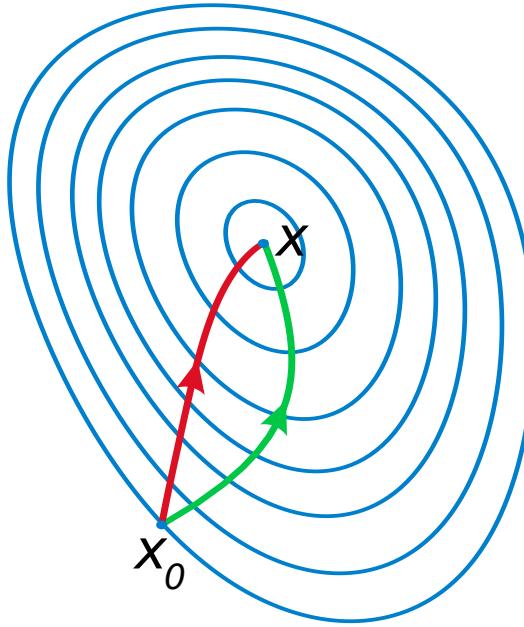
Newton based methods - Newton-Raphson Algorithm, Quasi-Newton methods (e.g., BFGS method) - tend to converge in fewer iterations, although each iteration typically requires more computation than a conjugate gradient iteration as Newton-like methods require computing the Hessian (matrix of second derivatives) in addition to the gradient. Quasi-Newton methods also require more memory to operate (see also the limited memory L-BFGS method).

16.1 External links

- An Introduction to the Conjugate Gradient Method Without the Agonizing Pain by Jonathan Richard Shewchuk.
- A NONLINEAR CONJUGATE GRADIENT METHOD WITH A STRONG GLOBAL CONVERGENCE PROPERTY by Y. H. DAI and Y. YUAN.

Chapter 17

Newton's method in optimization



A comparison of gradient descent (green) and Newton's method (red) for minimizing a function (with small step sizes). Newton's method uses curvature information to take a more direct route.

In calculus, Newton's method is an iterative method for finding the roots of a differentiable function f (i.e. solutions to the equation $f(x) = 0$). In optimization, Newton's method is applied to the derivative f' of a twice-differentiable function f to find the roots of the derivative (solutions to $f'(x) = 0$), also known as the stationary points of f .

17.1 Method

Newton's Method attempts to construct a sequence x_n from an initial guess x_0 that converges towards x_* such that $f'(x_*) = 0$. This x_* is a stationary point of f .

The second order Taylor expansion $f_T(x)$ of f around x_n (where $\Delta x := x - x_n$) is:

$$f_T(x_n + \Delta x) = f_T(x) = f(x_n) + f'(x_n)\Delta x + \frac{1}{2}f''(x_n)\Delta x^2,$$

and attains its extremum when its derivative with respect to Δx is equal to zero, i.e. when:

$$\frac{f_T(x_n + \Delta x) - f(x_n)}{\Delta x} = 0 \Leftrightarrow f'(x_n) + f''(x_n)\Delta x = 0.$$

Thus, provided that f is a twice-differentiable function, the sequence (x_n) defined by:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}, \quad n = 0, 1, \dots$$

will converge towards a root of f' , i.e. x_* for which $f'(x_*) = 0$.

17.2 Geometric interpretation

The geometric interpretation of Newton's method is that at each iteration one approximates $f(\mathbf{x})$ by a quadratic function around \mathbf{x}_n , and then takes a step towards the maximum/minimum of that quadratic function (in higher dimensions, this may also be a saddle point). Note that if $f(\mathbf{x})$ happens to be a quadratic function, then the exact extremum is found in one step.

17.3 Higher dimensions

The above iterative scheme can be generalized to several dimensions by replacing the derivative with the gradient, $\nabla f(\mathbf{x})$, and the reciprocal of the second derivative with the inverse of the Hessian matrix, $Hf(\mathbf{x})$. One obtains the iterative scheme

$$\mathbf{x}_{n+1} = \mathbf{x}_n - [Hf(\mathbf{x}_n)]^{-1}\nabla f(\mathbf{x}_n), \quad n \geq 0.$$

Usually Newton's method is modified to include a small step size $\gamma > 0$ instead of $\gamma = 1$

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma[Hf(\mathbf{x}_n)]^{-1}\nabla f(\mathbf{x}_n).$$

This is often done to ensure that the Wolfe conditions are satisfied at each step $\mathbf{x}_n \rightarrow \mathbf{x}_{n+1}$ of the iteration.

Where applicable, Newton's method converges much faster towards a local maximum or minimum than gradient descent. In fact, every local minimum has a neighborhood N such that, if we start with $\mathbf{x}_0 \in N$, Newton's method with step size $\gamma = 1$ converges quadratically (if the Hessian is invertible and a Lipschitz continuous function of \mathbf{x} in that neighborhood).

Finding the inverse of the Hessian in high dimensions can be an expensive operation. In such cases, instead of directly inverting the Hessian it's better to calculate the vector $\mathbf{p}_n = [Hf(\mathbf{x}_n)]^{-1} \nabla f(\mathbf{x}_n)$ as the solution to the system of linear equations

$$[Hf(\mathbf{x}_n)]\mathbf{p}_n = \nabla f(\mathbf{x}_n)$$

which may be solved by various factorizations or approximately (but to great accuracy) using iterative methods. Many of these methods are only applicable to certain types of equations, for example the Cholesky factorization and conjugate gradient will only work if $[Hf(\mathbf{x}_n)]$ is a positive definite matrix. While this may seem like a limitation, it's often useful indicator of something gone wrong, for example if a minimization problem is being approached and $[Hf(\mathbf{x}_n)]$ is not positive definite, then the iterations are converging to a saddle point and not a minimum.

On the other hand, if a constrained optimization is done (for example, with Lagrange multipliers), the problem may become one of saddle point finding, in which case the Hessian will be symmetric indefinite and the solution of \mathbf{p}_n will need to be done with a method that will work for such, such as the \mathbf{LDL}^T variant of Cholesky factorization or the conjugate residual method.

There also exist various quasi-Newton methods, where an approximation for the Hessian (or its inverse directly) is built up from changes in the gradient.

If the Hessian is close to a non-invertible matrix, the inverted Hessian can be numerically unstable and the solution may diverge. In this case, certain workarounds have been tried in the past, which have varied success with certain problems. One can, for example, modify the Hessian by adding a correction matrix B_n so as to make $H_f(\mathbf{x}_n) + B_n$ positive definite. One approach is to diagonalize H_f and choose B_n so that $H_f(\mathbf{x}_n) + B_n$ has the same eigenvectors as H_f , but with each negative eigenvalue replaced by $\epsilon > 0$.

An approach exploited in the Levenberg–Marquardt algorithm (which uses an approximate Hessian) is to add a scaled identity matrix to the Hessian, $\mu \mathbf{I}$, with the scale adjusted at every iteration as needed. For large μ and small Hessian, the iterations will behave like gradient descent with step size $\frac{1}{\mu}$. This results in slower but more reliable convergence where the Hessian doesn't provide useful information.

17.4 See also

- Quasi-Newton method
- Gradient descent
- Gauss–Newton algorithm
- Levenberg–Marquardt algorithm
- Trust region
- Optimization
- Nelder–Mead method

17.5 Notes

17.6 References

- Avriel, Mordecai (2003). *Nonlinear Programming: Analysis and Methods*. Dover Publishing. ISBN 0-486-43227-0.
- Bonnans, J. Frédéric; Gilbert, J. Charles; Lemaréchal, Claude; Sagastizábal, Claudia A. (2006). *Numerical optimization: Theoretical and practical aspects*. Universitext (Second revised ed. of translation of 1997 French ed.). Berlin: Springer-Verlag. pp. xiv+490. doi:10.1007/978-3-540-35447-5. ISBN 3-540-35445-X. MR 2265882.
- Fletcher, Roger (1987). *Practical methods of optimization* (2nd ed.). New York: John Wiley & Sons. ISBN 978-0-471-91547-8..
- Nocedal, Jorge & Wright, Stephen J. (1999). *Numerical Optimization*. Springer-Verlag. ISBN 0-387-98793-2.
- “Newton-Raphson visualization (1D)”. [https://www.mathworks.com/matlabcentral/fileexchange/3208-newtonraphson-visualization-1d](#)

Chapter 18

Barrier function

In constrained optimization, a field of mathematics, a **barrier function** is a continuous function whose value on a point increases to infinity as the point approaches the boundary of the **feasible region** (Nocedal and Wright 1999). It is used as a penalizing term for violations of constraints. The two most common types of barrier functions are **inverse barrier functions** and logarithmic barrier functions. Resumption of interest in logarithmic barrier functions was motivated by their connection with primal-dual interior point method.

When optimizing a function $f(x)$, the variable x can be constrained to be strictly lower than some constant b by instead optimizing the function $f(x) + g(x, b)$. Here, $g(x, b)$ is the barrier function.

18.1 Logarithmic barrier function

For logarithmic barrier functions, $g(x, b)$ is defined as $-\log(b - x)$ when $x < b$ and ∞ otherwise (in 1 dimension). See below for a definition in higher dimensions). This essentially relies on the fact that $\log(t)$ tends to negative infinity as t tends to 0.

This introduces a gradient to the function being optimized which favors less extreme values of x (in this case values lower than b), while having relatively low impact on the function away from these extremes.

Logarithmic barrier functions may be favored over less computationally expensive inverse barrier functions depending on the function being optimized.

18.1.1 Higher dimensions

Extending to higher dimensions is simple, provided each dimension is independent. For each variable x_i which should be limited to be strictly lower than b_i , add $-\log(b_i - x_i)$.

18.1.2 Formal definition

minimize : $\mathbf{c}^T x$

subject to: $\mathbf{a}_i^T x \leq b_i, i = 1, \dots, m$

assume strictly feasible: $\{\mathbf{x} | Ax < b\} \neq \emptyset$

$$\text{define } \begin{aligned} \text{logarithmic barrier } \Phi(x) &= \\ \begin{cases} \sum_{i=1}^m -\log(b_i - a_i^T x) & \text{for } Ax < b \\ +\infty & \text{otherwise} \end{cases} \end{aligned}$$

18.2 References

- Nocedal, Jorge; Stephen Wright (1999). *Numerical Optimization*. New York, NY: Springer. ISBN 0-387-98793-2.
- Lecture 14: Barrier method from Professor Lieven Vandenberghe of UCLA

Chapter 19

Penalty method

Penalty methods are a certain class of algorithms for solving constrained optimization problems.

A penalty method replaces a constrained optimization problem by a series of unconstrained problems whose solutions ideally converge to the solution of the original constrained problem. The unconstrained problems are formed by adding a term, called a **penalty function**, to the **objective function** that consists of a *penalty parameter* multiplied by a measure of violation of the constraints. The measure of violation is nonzero when the constraints are violated and is zero in the region where constraints are not violated.

19.1 Example

Let us say we are solving the following constrained problem:

$$\min f(\mathbf{x})$$

subject to

$$c_i(\mathbf{x}) \geq 0 \quad \forall i \in I.$$

This problem can be solved as a series of unconstrained minimization problems

$$\min \Phi_k(\mathbf{x}) = f(\mathbf{x}) + \sigma_k \sum_{i \in I} g(c_i(\mathbf{x}))$$

where

$$g(c_i(\mathbf{x})) = \min(0, c_i(\mathbf{x}))^2.$$

In the above equations, $g(c_i(\mathbf{x}))$ is the *penalty function* while σ_k are the *penalty coefficients*. In each iteration k of the method, we increase the penalty coefficient σ_k (e.g. by a factor of 10), solve the unconstrained problem and use the solution as the initial guess for the next iteration. Solutions of the successive unconstrained problems will eventually converge to the solution of the original constrained problem.

19.2 Practical application

Image compression optimization algorithms can make use of penalty functions for selecting how best to compress zones of colour to single representative values.^{[1][2]}

19.3 Barrier methods

Barrier methods constitute an alternative class of algorithms for constrained optimization. These methods also add a penalty-like term to the objective function, but in this case the iterates are forced to remain interior to the feasible domain and the barrier is in place to bias the iterates to remain away from the boundary of the feasible region.

19.4 See also

- Barrier function
- Interior point method
- Augmented Lagrangian method

19.5 References

[1] Galar, M.; Jurio, A.; Lopez-Molina, C.; Paternain, D.; Sanz, J.; Bustince, H. (2013). “Aggregation functions to combine RGB color channels in stereo matching”. *Optics Express* **21**: 1247–1257. doi:10.1364/oe.21.001247.

[2] “Researchers restore image using version containing between 1 and 10 percent of information”. Phys.org (Omicron Technology Limited). Retrieved 26 October 2013.

Smith, Alice E.; Coit David W. *Penalty functions Handbook of Evolutionary Computation*, Section C 5.2. Oxford University Press and Institute of Physics Publishing, 1996.

Courant, R. *Variational methods for the solution of problems of equilibrium and vibrations*. Bull. Amer. Math. Soc., 49, 1–23, 1943.

Chapter 20

Augmented Lagrangian method

Augmented Lagrangian methods are a certain class of algorithms for solving constrained optimization problems. They have similarities to **penalty methods** in that they replace a constrained optimization problem by a series of unconstrained problems and add a penalty term to the **objective**; the difference is that the augmented Lagrangian method adds yet another term, designed to mimic a **Lagrange multiplier**. The augmented Lagrangian is not the same as the **method of Lagrange multipliers**.

Viewed differently, the unconstrained objective is the **Lagrangian** of the constrained problem, with an additional penalty term (the **augmentation**).

The method was originally known as the **method of multipliers**, and was studied much in the 1970 and 1980s as a good alternative to penalty methods. It was first discussed by Magnus Hestenes in 1969^[1] and by Powell in 1969.^[2] The method was studied by R. Tyrrell Rockafellar in relation to Fenchel duality, particularly in relation to proximal-point methods, Moreau–Yosida regularization, and maximal monotone operators: These methods were used in structural optimization. The method was also studied by Dimitri Bertsekas, notably in his 1982 book,^[3] together with extensions involving nonquadratic regularization functions, such as **entropic regularization**, which gives rise to the “exponential method of multipliers,” a method that handles inequality constraints with a twice differentiable augmented Lagrangian function.

Since the 1970s, sequential quadratic programming (SQP) and interior point methods (IPM) have had increasing attention, in part because they more easily use sparse matrix subroutines from numerical software libraries, and in part because IPMs have proven complexity results via the theory of **self-concordant functions**. The augmented Lagrangian method was rejuvenated by the optimization systems **LANCELOT** and **AMPL**, which allowed sparse matrix techniques to be used on seemingly dense but “partially separable” problems. The method is still useful for some problems.^[4] Around 2007, there was a resurgence of augmented Lagrangian methods in fields such as **total-variation denoising** and **compressed sensing**. In particular, a variant of the standard augmented Lagrangian method that uses partial updates (similar to the Gauss-Seidel method for solving linear equations) known as the **alternating direction method of multipliers** or

ADMM gained some attention.

20.1 General method

Let us say we are solving the following constrained problem:

$$\min f(\mathbf{x})$$

subject to

$$c_i(\mathbf{x}) = 0 \quad \forall i \in I.$$

This problem can be solved as a series of unconstrained minimization problems. For reference, we first list the **penalty method approach**:

$$\min \Phi_k(\mathbf{x}) = f(\mathbf{x}) + \mu_k \sum_{i \in I} c_i(\mathbf{x})^2$$

The penalty method solves this problem, then at the next iteration it re-solves the problem using a larger value of μ_k (and using the old solution as the initial guess or “warm-start”).

The augmented Lagrangian method uses the following unconstrained objective:

$$\min \Phi_k(\mathbf{x}) = f(\mathbf{x}) + \frac{\mu_k}{2} \sum_{i \in I} c_i(\mathbf{x})^2 - \sum_{i \in I} \lambda_i c_i(\mathbf{x})$$

and after each iteration, in addition to updating μ_k , the variable λ is also updated according to the rule

$$\lambda_i \leftarrow \lambda_i - \mu_k c_i(\mathbf{x}_k)$$

where \mathbf{x}_k is the solution to the unconstrained problem at the k th step, i.e. $\mathbf{x}_k = \operatorname{argmin} \Phi_k(\mathbf{x})$

The variable λ is an estimate of the **Lagrange multiplier**, and the accuracy of this estimate improves at every step.

The major advantage of the method is that unlike the **penalty method**, it is not necessary to take $\mu \rightarrow \infty$ in order to solve the original constrained problem. Instead, because of the presence of the Lagrange multiplier term, μ can stay much smaller.

The method can be extended to handle inequality constraints. For a discussion of practical improvements, see [5].

20.2 Comparison with penalty methods

In [6] it is suggested that the augmented Lagrangian method is generally preferred to the quadratic penalty method since there is little extra computational cost and the parameter μ need not go to infinity, thus avoiding ill-conditioning this is used.

20.3 Alternating direction method of multipliers

The alternating direction method of multipliers (ADMM) is a variant of the augmented Lagrangian scheme that uses partial updates for the dual variables. This method is often applied to solve problems such as

$$\min_x f(x) + g(x).$$

This is equivalent to the constrained problem

$$\min_{x,y} f(x) + g(y), \quad \text{to subject } x = y.$$

Though this change may seem trivial, the problem can now be attacked using methods of constrained optimization (in particular, the augmented Lagrangian method), and the objective function is separable in x and y . The dual update requires solving a proximity function in x and y at the same time; the ADMM technique allows this problem to be solved approximately by first solving for x with y fixed, and then solving for y with x fixed. Rather than iterate until convergence (like the **Jacobi method**), the algorithm proceeds directly to updating the dual variable and then repeating the process. This is not equivalent to the exact minimization, but surprisingly, it can still be shown that this method converges to the right answer (under some assumptions). Because of this approximation, the algorithm is distinct from the pure augmented Lagrangian method.

The ADMM can be viewed as an application of the **Douglas-Rachford splitting algorithm**, and the Douglas-Rachford algorithm is in turn an instance of the **Proximal point algorithm**; details can be found here.^[7] There are several modern software packages that solve **Basis pursuit** and variants and use the ADMM; such packages include **YALL1** (2009), **SpaRSA** (2009) and **SALSA** (2009).

20.4 Software

Some well-known software packages that use the augmented Lagrangian method are **LANCELOT** and **PENNON**. The software **MINOS** also uses an augmented Lagrangian method for some types of problems.

20.5 See also

- Penalty methods
- Barrier method
- Barrier function
- Lagrange multiplier

20.6 References

- [1] M.R. Hestenes, “Multiplier and gradient methods”, *Journal of Optimization Theory and Applications*, 4, 1969, pp. 303–320
- [2] M.J.D. Powell, “A method for nonlinear constraints in minimization problems”, in *Optimization* ed. by R. Fletcher, Academic Press, New York, NY, 1969, pp. 283–298.
- [3] Dimitri P. Bertsekas, *Constrained optimization and Lagrange multiplier methods*, Athena Scientific, 1996 (first published 1982)
- [4] Nocedal & Wright (2006), chapter 17
- [5] Nocedal & Wright (2006), chapter 17
- [6] Nocedal & Wright (2006), chapter 17
- [7] Eckstein, J.; Bertsekas, D. P. (1992). “On the Douglas—Rachford splitting method and the proximal point algorithm for maximal monotone operators”. *Mathematical Programming* 55: 293. doi:10.1007/BF01581204.

20.7 Bibliography

- Bertsekas, Dimitri P. (1999), *Nonlinear Programming* (2nd ed.), Belmont, Mass: Athena Scientific, ISBN 1-886529-00-0
- Nocedal, Jorge; Wright, Stephen J. (2006), *Numerical Optimization* (2nd ed.), Berlin, New York: Springer-Verlag, ISBN 978-0-387-30303-1

Chapter 21

Sequential quadratic programming

Sequential quadratic programming (SQP) is an iterative method for nonlinear optimization. SQP methods are used on problems for which the objective function and the constraints are twice continuously differentiable.

SQP methods solve a sequence of optimization subproblems, each of which optimizes a quadratic model of the objective subject to a linearization of the constraints. If the problem is unconstrained, then the method reduces to Newton's method for finding a point where the gradient of the objective vanishes. If the problem has only equality constraints, then the method is equivalent to applying Newton's method to the first-order optimality conditions, or Karush–Kuhn–Tucker conditions, of the problem. SQP methods have been implemented in many packages, including NPSOL, SNOPT, NLPQL, OPSYC, OPTIMA, MATLAB, GNU Octave and SQP.

21.1 Algorithm basics

Consider a nonlinear programming problem of the form:

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & b(x) \geq 0 \\ & c(x) = 0. \end{aligned}$$

The Lagrangian for this problem is;

$$\mathcal{L}(x, \lambda, \sigma) = f(x) - \lambda^T b(x) - \sigma^T c(x),$$

where λ and σ are Lagrange multipliers. At an iterate x_k , a basic sequential quadratic programming algorithm defines an appropriate search direction d_k as a solution to the quadratic programming subproblem

$$\begin{aligned} \min_d \quad & f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla_{xx}^2 \mathcal{L}(x_k, \lambda_k, \sigma_k) d \\ \text{s.t.} \quad & b(x_k) + \nabla b(x_k)^T d \geq 0 \\ & c(x_k) + \nabla c(x_k)^T d = 0. \end{aligned}$$

Note that the term $f(x_k)$ in the expression above may be left out for the minimization problem, since it is constant.

21.2 See also

- Sequential linear programming
- Secant method
- Newton's method

21.3 References

- Bonnans, J. Frédéric; Gilbert, J. Charles; Lemaréchal, Claude; Sagastizábal, Claudia A. (2006). *Numerical optimization: Theoretical and practical aspects*. Universitext (Second revised ed. of translation of 1997 French ed.). Berlin: Springer-Verlag. pp. xiv+490. doi:10.1007/978-3-540-35447-5. ISBN 3-540-35445-X. MR 2265882.
- Jorge Nocedal and Stephen J. Wright (2006). *Numerical Optimization*. Springer. ISBN 0-387-30303-0.

21.4 External links

- Sequential Quadratic Programming at NEOS guide

Chapter 22

Successive linear programming

Successive Linear Programming (SLP), also known as **Sequential Linear Programming**, is an optimization technique for approximately solving **nonlinear optimization** problems.^[1]

Starting at some estimate of the optimal solution, the method is based on solving a sequence of first-order approximations (i.e. linearizations) of the model. The linearizations are linear programming problems, which can be solved efficiently. As the linearizations need not be bounded, **trust regions** or similar techniques are needed to ensure convergence in theory.^[2]

SLP has been used widely in the petrochemical industry since the 1970s.^[3]

22.1 References

- [1] Nocedal, Jorge; Wright J., Stephen (October 1999), “Numerical Optimization”, *Springer Series in Operations Research* Chapter 15.1, Categorizing Optimization Algorithms
- [2] Bazaraa, Mokhtar S.; Sheraly, Hanif D.; Shetty, C.M. (1993), *Nonlinear Programming, Theory and Applications* (2nd ed.), John Wiley & Sons, p. 432, ISBN 0-471-55793-5.
- [3] Palacios-Gomez, F.; Lasdon, L.; Enquist, M. (October 1982), “Nonlinear Optimization by Successive Linear Programming”, *Management Science* **28** (10)

22.2 External links

- Modelling, Simulation and Optimisation
- Refinery planning, Optimisation

Chapter 23

Cutting-plane method

In mathematical optimization, the **cutting-plane method** is an umbrella term for optimization methods which iteratively refine a feasible set or objective function by means of linear inequalities, termed *cuts*. Such procedures are popularly used to find integer solutions to mixed integer linear programming (MILP) problems, as well as to solve general, not necessarily differentiable convex optimization problems. The use of cutting planes to solve MILP was introduced by Ralph E. Gomory and Václav Chvátal.

Cutting plane methods for MILP work by solving a non-integer linear program, the *linear relaxation* of the given integer program. The theory of Linear Programming dictates that under mild assumptions (if the linear program has an optimal solution, and if the feasible region does not contain a line), one can always find an extreme point or a corner point that is optimal. The obtained optimum is tested for being an integer solution. If it is not, there is guaranteed to exist a linear inequality that *separates* the optimum from the *convex hull* of the true feasible set. Finding such an inequality is the *separation problem*, and such an inequality is a *cut*. A cut can be added to the relaxed linear program. Then, the current non-integer solution is no longer feasible to the relaxation. This process is repeated until an optimal integer solution is found.

Cutting-plane methods for general convex continuous optimization and variants are known under various names: Kelley's method, Kelley-Cheney-Goldstein method, and bundle methods. They are popularly used for non-differentiable convex minimization, where a convex objective function and its subgradient can be evaluated efficiently but usual gradient methods for differentiable optimization can not be used. This situation is most typical for the concave maximization of *Lagrangian dual* functions. Another common situation is the application of the *Dantzig-Wolfe decomposition* to a structured optimization problem in which formulations with an exponential number of variables are obtained. Generating these variables on demand by means of *delayed column generation* is identical to performing a cutting plane on the respective dual problem.

23.1 Gomory's cut

Cutting planes were proposed by Ralph Gomory in the 1950s as a method for solving integer programming and mixed-integer programming problems. However most experts, including Gomory himself, considered them to be impractical due to numerical instability, as well as ineffective because many rounds of cuts were needed to make progress towards the solution. Things turned around when in the mid-1990s Cornuejols and co-workers showed them to be very effective in combination with branch-and-bound (called branch-and-cut) and ways to overcome numerical instabilities. Nowadays, all commercial MILP solvers use Gomory cuts in one way or another. Gomory cuts are very efficiently generated from a simplex tableau, whereas many other types of cuts are either expensive or even NP-hard to separate. Among other general cuts for MILP, most notably lift-and-project dominates Gomory cuts.

Let an integer programming problem be formulated (in Standard Form) as:

Maximize $c^T x$

Subject to $Ax = b$,

$x \geq 0$, x_i all integers.

The method proceeds by first dropping the requirement that the x_i be integers and solving the associated linear programming problem to obtain a basic feasible solution. Geometrically, this solution will be a vertex of the convex polytope consisting of all feasible points. If this vertex is not an integer point then the method finds a hyperplane with the vertex on one side and all feasible integer points on the other. This is then added as an additional linear constraint to exclude the vertex found, creating a modified linear program. The new program is then solved and the process is repeated until an integer solution is found.

Using the *simplex method* to solve a linear program produces a set of equations of the form

$$x_i + \sum \bar{a}_{i,j} x_j = \bar{b}_i$$

where x_i is a basic variable and the x_j 's are the nonbasic variables. Rewrite this equation so that the integer parts

are on the left side and the fractional parts are on the right side:

$$x_i + \sum \lfloor \bar{a}_{i,j} \rfloor x_j - \lfloor \bar{b}_i \rfloor = \bar{b}_i - \lfloor \bar{b}_i \rfloor - \sum (\bar{a}_{i,j} - \lfloor \bar{a}_{i,j} \rfloor) x_j$$

For any integer point in the feasible region the right side of this equation is less than 1 and the left side is an integer, therefore the common value must be less than or equal to 0. So the inequality

$$\bar{b}_i - \lfloor \bar{b}_i \rfloor - \sum (\bar{a}_{i,j} - \lfloor \bar{a}_{i,j} \rfloor) x_j \leq 0$$

must hold for any integer point in the feasible region. Furthermore, nonbasic variables are equal to 0s in any basic solution and if x_i is not an integer for the basic solution x ,

$$\bar{b}_i - \lfloor \bar{b}_i \rfloor - \sum (\bar{a}_{i,j} - \lfloor \bar{a}_{i,j} \rfloor) x_j = \bar{b}_i - \lfloor \bar{b}_i \rfloor > 0.$$

So the inequality above excludes the basic feasible solution and thus is a cut with the desired properties. Introducing a new slack variable x_k for this inequality, a new constraint is added to the linear program, namely

$$x_k + \sum (\lfloor \bar{a}_{i,j} \rfloor - \bar{a}_{i,j}) x_j = \lfloor \bar{b}_i \rfloor - \bar{b}_i, \quad x_k \geq 0, \quad x_k \text{ an integer.}$$

23.2 Convex optimization

Cutting plane methods are also applicable in nonlinear programming. The underlying principle is to approximate the feasible region of a nonlinear (convex) program by a finite set of closed half spaces and to solve a sequence of approximating linear programs.

23.3 See also

- Branch and cut
- Branch and bound
- Dantzig-Wolfe decomposition
- Column generation
- Benders' decomposition

23.4 References

Avriel, Mordecai (2003). *Nonlinear Programming: Analysis and Methods*. Dover Publications. ISBN 0-486-43227-0

Cornuejols, Gerard (2008). Valid Inequalities for Mixed Integer Linear Programs. *Mathematical Programming Ser. B*, (2008) 112:3-44.

Cornuejols, Gerard (2007). Revival of the Gomory Cuts in the 1990s. *Annals of Operations Research*, Vol. 149 (2007), pp. 63–66.

23.5 External links

- “Integer Programming” Section 9.8

Chapter 24

Frank–Wolfe algorithm

The **Frank–Wolfe algorithm** is an iterative first-order optimization algorithm for constrained convex optimization. Also known as the **conditional gradient method**,^[1] **reduced gradient algorithm** and the **convex combination algorithm**, the method was originally proposed by Marguerite Frank and Philip Wolfe in 1956.^[2] In each iteration, the Frank–Wolfe algorithm considers a linear approximation of the objective function, and moves slightly towards a minimizer of this linear function (taken over the same domain).

24.1 Problem statement

Suppose \mathcal{D} is a compact convex set in a vector space and $f: \mathcal{D} \rightarrow \mathbb{R}$ is a convex differentiable real-valued function. The Frank–Wolfe algorithm solves the optimization problem

$$f(\mathbf{x})$$

$$\mathbf{x} \in \mathcal{D}$$

24.2 Algorithm

Initialization: Let $k \leftarrow 0$, and let \mathbf{x}_0 be any point in \mathcal{D} .

Step 1. *Direction-finding subproblem:* Find \mathbf{s}_k solving

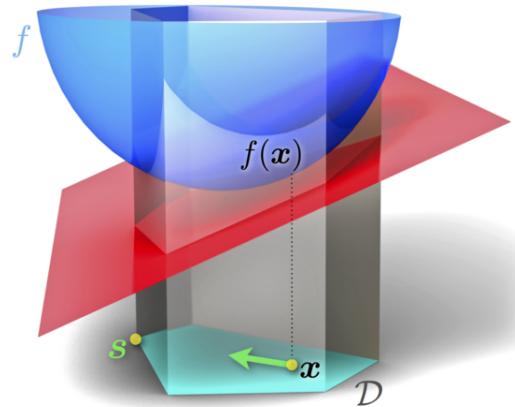
$$\mathbf{s}^T \nabla f(\mathbf{x}_k)$$

$$\mathbf{s} \in \mathcal{D}$$

(Interpretation: Minimize the linear approximation of the problem given by the first-order Taylor approximation of f around \mathbf{x}_k .)

Step 2. *Step size determination:* Set $\gamma \leftarrow \frac{2}{k+2}$, or alternatively find γ that minimizes $f(\mathbf{x}_k + \gamma(\mathbf{s}_k - \mathbf{x}_k))$ subject to $0 \leq \gamma \leq 1$.

Step 3. *Update:* Let $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \gamma(\mathbf{s}_k - \mathbf{x}_k)$, let $k \leftarrow k + 1$ and go to Step 1.



A step of the Frank–Wolfe algorithm

24.3 Properties

While competing methods such as gradient descent for constrained optimization require a projection step back to the feasible set in each iteration, the Frank–Wolfe algorithm only needs the solution of a linear problem over the same set in each iteration, and automatically stays in the feasible set.

The convergence of the Frank–Wolfe algorithm is sublinear in general: the error to the optimum is $O(1/k)$ after k iterations. The same convergence rate can also be shown if the sub-problems are only solved approximately.^[3]

The iterates of the algorithm can always be represented as a sparse convex combination of the extreme points of the feasible set, which has helped to the popularity of the algorithm for sparse greedy optimization in machine learning and signal processing problems,^[4] as well as for example the optimization of minimum-cost flows in transportation networks.^[5]

If the feasible set is given by a set of linear constraints, then the subproblem to be solved in each iteration becomes a linear program.

While the worst-case convergence rate with $O(1/k)$ can not be improved in general, faster convergence can be obtained for special problem classes, such as some strongly

convex problems.^[6]

24.4 Lower bounds on the solution value, and primal-dual analysis

Since f is convex, $f(\mathbf{y})$ is always above the tangent plane of f at any point $\mathbf{x} \in \mathcal{D}$:

$$f(\mathbf{y}) \geq f(\mathbf{x}) + (\mathbf{y} - \mathbf{x})^T \nabla f(\mathbf{x})$$

This holds in particular for the (unknown) optimal solution \mathbf{x}^* . The best lower bound with respect to a given point \mathbf{x} is given by

$$f(\mathbf{x}^*) \geq \min_{\mathbf{y} \in \mathcal{D}} f(\mathbf{y}) + (\mathbf{y} - \mathbf{x})^T \nabla f(\mathbf{x}) = f(\mathbf{x}) - \mathbf{x}^T \nabla f(\mathbf{x}) + \min_{\mathbf{y} \in \mathcal{D}} \mathbf{y}^T \nabla f(\mathbf{x})$$

The latter optimization problem is solved in every iteration of the Frank-Wolfe algorithm, therefore the solution \mathbf{s}_k of the direction-finding subproblem of the k -th iteration can be used to determine increasing lower bounds l_k during each iteration by setting $l_0 = -\infty$ and

$$l_k := \max(l_{k-1}, f(\mathbf{x}_k) + (\mathbf{s}_k - \mathbf{x}_k)^T \nabla f(\mathbf{x}_k))$$

Such lower bounds on the unknown optimal value are important in practice because they can be used as a stopping criterion, and give an efficient certificate of the approximation quality in every iteration, since always $l_k \leq f(\mathbf{x}^*) \leq f(\mathbf{x}_k)$.

It has been shown that this corresponding duality gap, that is the difference between $f(\mathbf{x}_k)$ and the lower bound l_k , decreases with the same convergence rate, i.e. $f(\mathbf{x}_k) - l_k = O(1/k)$.

24.5 Notes

- [1] Levitin, E. S.; Polyak, B. T. (1966). “Constrained minimization methods”. *USSR Computational Mathematics and Mathematical Physics* **6** (5): 1. doi:10.1016/0041-5553(66)90114-5.
- [2] Frank, M.; Wolfe, P. (1956). “An algorithm for quadratic programming”. *Naval Research Logistics Quarterly* **3**: 95. doi:10.1002/nav.3800030109.
- [3] Dunn, J. C.; Harshbarger, S. (1978). “Conditional gradient algorithms with open loop step size rules”. *Journal of Mathematical Analysis and Applications* **62** (2): 432. doi:10.1016/0022-247X(78)90137-3.
- [4] Clarkson, K. L. (2010). “coresets, sparse greedy approximation, and the Frank-Wolfe algorithm”. *ACM Transactions on Algorithms* **6** (4): 1. doi:10.1145/1824777.1824783.

[5] Fukushima, M. (1984). “A modified Frank-Wolfe algorithm for solving the traffic assignment problem”. *Transportation Research Part B: Methodological* **18** (2): 169–153. doi:10.1016/0191-2615(84)90029-8.

[6] Bertsekas, Dimitri (1999). *Nonlinear Programming*. Athena Scientific. p. 215. ISBN 1-886529-00-0.

24.6 Bibliography

- Jaggi, Martin (2013). “Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization”. *Journal of Machine Learning Research: Workshop and Conference Proceedings* **28** (1): 427–435. (Overview paper)
- The Frank-Wolfe algorithm description

24.7 External links

- Marguerite Frank giving a personal account of the history of the algorithm

24.8 See also

- Proximal gradient methods

Chapter 25

Subgradient method

Subgradient methods are iterative methods for solving convex minimization problems. Originally developed by Naum Z. Shor and others in the 1960s and 1970s, subgradient methods are convergent when applied even to a non-differentiable objective function. When the objective function is differentiable, subgradient methods for unconstrained problems use the same search direction as the method of steepest descent.

Subgradient methods are slower than Newton's method when applied to minimize twice continuously differentiable convex functions. However, Newton's method fails to converge on problems that have non-differentiable kinks.

In recent years, some interior-point methods have been suggested for convex minimization problems, but subgradient projection methods and related bundle methods of descent remain competitive. For convex minimization problems with very large number of dimensions, subgradient-projection methods are suitable, because they require little storage.

Subgradient projection methods are often applied to large-scale problems with decomposition techniques. Such decomposition methods often allow a simple distributed method for a problem.

25.1 Classical subgradient rules

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a convex function with domain \mathbb{R}^n . A classical subgradient method iterates

$$x^{(k+1)} = x^{(k)} - \alpha_k g^{(k)}$$

where $g^{(k)}$ denotes a subgradient of f at $x^{(k)}$. If f is differentiable, then its only subgradient is the gradient vector ∇f itself. It may happen that $-g^{(k)}$ is not a descent direction for f at $x^{(k)}$. We therefore maintain a list f_{best} that keeps track of the lowest objective function value found so far, i.e.

$$f_{\text{best}}^{(k)} = \min\{f_{\text{best}}^{(k-1)}, f(x^{(k)})\}.$$

which is resultant convex optimized.

25.1.1 Step size rules

Many different types of step-size rules are used by subgradient methods. This article notes five classical step-size rules for which convergence proofs are known:

- Constant step size, $\alpha_k = \alpha$.
- Constant step length, $\alpha_k = \gamma / \|g^{(k)}\|_2$, which gives $\|x^{(k+1)} - x^{(k)}\|_2 = \gamma$.
- Square summable but not summable step size, i.e. any step sizes satisfying

$$\alpha_k \geq 0, \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty, \quad \sum_{k=1}^{\infty} \alpha_k = \infty.$$

- Nonsummable diminishing, i.e. any step sizes satisfying

$$\alpha_k \geq 0, \quad \lim_{k \rightarrow \infty} \alpha_k = 0, \quad \sum_{k=1}^{\infty} \alpha_k = \infty.$$

- Nonsummable diminishing step lengths, i.e. $\alpha_k = \gamma_k / \|g^{(k)}\|_2$, where

$$\gamma_k \geq 0, \quad \lim_{k \rightarrow \infty} \gamma_k = 0, \quad \sum_{k=1}^{\infty} \gamma_k = \infty.$$

For all five rules, the step-sizes are determined “off-line”, before the method is iterated; the step-sizes do not depend on preceding iterations. This “off-line” property of subgradient methods differs from the “on-line” step-size rules used for descent methods for differentiable functions: Many methods for minimizing differentiable functions satisfy Wolfe’s sufficient conditions for convergence, where step-sizes typically depend on the current point and the current search-direction. An extensive discussion of stepsize rules for subgradient methods, including incremental versions, is given in the books by Bertsekas [1] and by Bertsekas, Nedic, and Ozdaglar. [2]

25.1.2 Convergence results

For constant step-length and scaled subgradients having Euclidean norm equal to one, the subgradient method converges to an arbitrarily close approximation to the minimum value, that is

$$\lim_{k \rightarrow \infty} f_{\text{best}}^{(k)} - f^* < \epsilon \text{ by a result of Shor.}^{[3]}$$

These classical subgradient methods have poor performance and are no longer recommended for general use.^{[4][5]} However, they are still used widely in specialized applications because they are simple and they can be easily adapted to take advantage of the special structure of the problem at hand.

25.2 Subgradient-projection & bundle methods

During the 1970s, Claude Lemaréchal and Phil. Wolfe proposed “bundle methods” of descent for problems of convex minimization.^[6] The meaning of the term “bundle methods” has changed significantly since that time. Modern versions and full convergence analysis were provided by Kiwiel.^[7] Contemporary bundle-methods often use “level control” rules for choosing step-sizes, developing techniques from the “subgradient-projection” method of Boris T. Polyak (1969). However, there are problems on which bundle methods offer little advantage over subgradient-projection methods.^{[4][5]}

25.3 Constrained optimization

25.3.1 Projected subgradient

One extension of the subgradient method is the **projected subgradient method**, which solves the constrained optimization problem

$$f(x)$$

$$x \in \mathcal{C}$$

where \mathcal{C} is a convex set. The projected subgradient method uses the iteration

$$x^{(k+1)} = P \left(x^{(k)} - \alpha_k g^{(k)} \right)$$

where P is projection on \mathcal{C} and $g^{(k)}$ is any subgradient of f at $x^{(k)}$.

25.3.2 General constraints

The subgradient method can be extended to solve the inequality constrained problem

$$f_0(x)$$

$$f_i(x) \leq 0, \quad i = 1, \dots, m$$

where f_i are convex. The algorithm takes the same form as the unconstrained case

$$x^{(k+1)} = x^{(k)} - \alpha_k g^{(k)}$$

where $\alpha_k > 0$ is a step size, and $g^{(k)}$ is a subgradient of the objective or one of the constraint functions at x . Take

$$g^{(k)} = \begin{cases} \partial f_0(x) & \text{if } f_i(x) \leq 0 \ \forall i = 1 \dots m \\ \partial f_j(x) & \text{some for } j \text{ that such } f_j(x) > 0 \end{cases}$$

where ∂f denotes the subdifferential of f . If the current point is feasible, the algorithm uses an objective subgradient; if the current point is infeasible, the algorithm chooses a subgradient of any violated constraint.

25.4 References

- [1] Bertsekas, Dimitri P. (2015). *Convex Optimization Algorithms* (Second ed.). Belmont, MA.: Athena Scientific. ISBN 978-1-886529-28-1.
- [2] Bertsekas, Dimitri P.; Nedic, Angelia; Ozdaglar, Asuman (2003). *Convex Analysis and Optimization* (Second ed.). Belmont, MA.: Athena Scientific. ISBN 1-886529-45-0.
- [3] The approximate convergence of the constant step-size (scaled) subgradient method is stated as Exercise 6.3.14(a) in Bertsekas (page 636): Bertsekas, Dimitri P. (1999). *Nonlinear Programming* (Second ed.). Cambridge, MA.: Athena Scientific. ISBN 1-886529-00-0. On page 636, Bertsekas attributes this result to Shor: Shor, Naum Z. (1985). *Minimization Methods for Non-differentiable Functions*. Springer-Verlag. ISBN 0-387-12763-1.
- [4] Lemaréchal, Claude (2001). “Lagrangian relaxation”. In Michael Jünger and Denis Naddef. *Computational combinatorial optimization: Papers from the Spring School held in Schloss Dagstuhl, May 15–19, 2000*. Lecture Notes in Computer Science **2241**. Berlin: Springer-Verlag. pp. 112–156. doi:10.1007/3-540-45586-8_4. ISBN 3-540-42877-1. MR 1900016.
- [5] Kiwiel, Krzysztof C.; Larsson, Torbjörn; Lindberg, P. O. (August 2007). “Lagrangian relaxation via ballstep subgradient methods”. *Mathematics of Operations Research* **32** (3): 669–686. doi:10.1287/moor.1070.0261. MR 2348241.

- [6] Bertsekas, Dimitri P. (1999). *Nonlinear Programming* (Second ed.). Cambridge, MA.: Athena Scientific. ISBN 1-886529-00-0.
- [7] Kiwiel, Krzysztof (1985). *Methods of Descent for Nondifferentiable Optimization*. Berlin: Springer Verlag. p. 362. ISBN 978-3540156420. MR 0797754.

25.5 Further reading

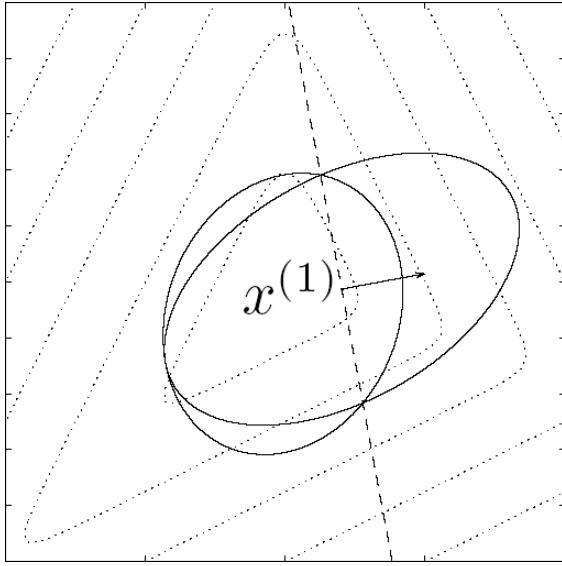
- Bertsekas, Dimitri P. (1999). *Nonlinear Programming*. Belmont, MA.: Athena Scientific. ISBN 1-886529-00-0.
- Bertsekas, Dimitri P.; Nedic, Angelia; Ozdaglar, Asuman (2003). *Convex Analysis and Optimization* (Second ed.). Belmont, MA.: Athena Scientific. ISBN 1-886529-45-0.
- Bertsekas, Dimitri P. (2015). *Convex Optimization Algorithms*. Belmont, MA.: Athena Scientific. ISBN 978-1-886529-28-1.
- Shor, Naum Z. (1985). *Minimization Methods for Non-differentiable Functions*. Springer-Verlag. ISBN 0-387-12763-1.
- Ruszczyński, Andrzej (2006). *Nonlinear Optimization*. Princeton, NJ: Princeton University Press. pp. xii+454. ISBN 978-0691119151. MR 2199043.

25.6 External links

- EE364A and EE364B, Stanford's convex optimization course sequence.

Chapter 26

Ellipsoid method



An iteration of the ellipsoid method

In mathematical optimization, the **ellipsoid method** is an iterative method for minimizing convex functions. When specialized to solving feasible linear optimization problems with rational data, the ellipsoid method is an algorithm which finds an optimal solution in a finite number of steps.

The ellipsoid method generates a sequence of ellipsoids whose volume uniformly decreases at every step, thus enclosing a minimizer of a convex function.

26.1 History

The ellipsoid method has a long history. As an iterative method, a preliminary version was introduced by Naum Z. Shor. In 1972, an approximation algorithm for real convex minimization was studied by Arkadi Nemirovski and David B. Yudin (Judin). As an algorithm for solving linear programming problems with rational data, the ellipsoid algorithm was studied by Leonid Khachiyan: Khachiyan's achievement was to prove the polynomial-time solvability of linear programs.

Following Khachiyan's work, the ellipsoid method was

the only algorithm for solving linear programs whose runtime had been proved to be polynomial until Karmarkar's algorithm. However, the interior-point method and variants of the simplex algorithm are much faster than the ellipsoid method in practice. Karmarkar's algorithm is also faster in the worst case.

However, the ellipsoidal algorithm allows complexity theorists to achieve (worst-case) bounds that depend on the dimension of the problem and on the size of the data, but not on the number of rows, so it remained important in combinatorial optimization theory for many decades.^{[1][2][3][4]} Only in the 21st century have interior-point algorithms with similar complexity properties appeared.

26.2 Description

Main article: Convex optimization

A convex minimization problem consists of a convex function $f_0(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ to be minimized over the variable x , convex inequality constraints of the form $f_i(x) \leq 0$, where the functions f_i are convex, and linear equality constraints of the form $h_i(x) = 0$. We are also given an initial ellipsoid $\mathcal{E}^{(0)} \subset \mathbb{R}^n$ defined as

$$\mathcal{E}^{(0)} = \left\{ z \in \mathbb{R}^n : (z - x_0)^T P_{(0)}^{-1}(z - x_0) \leq 1 \right\}$$

containing a minimizer x^* , where $P \succ 0$ and x_0 is the center of \mathcal{E} . Finally, we require the existence of a cutting-plane oracle for the function f . One example of a cutting-plane is given by a subgradient g of f .

26.3 Unconstrained minimization

At the k -th iteration of the algorithm, we have a point $x^{(k)}$ at the center of an ellipsoid $\mathcal{E}^{(k)} = \left\{ x \in \mathbb{R}^n : (x - x^{(k)})^T P_{(k)}^{-1}(x - x^{(k)}) \leq 1 \right\}$. We query the cutting-plane oracle to obtain a vector $g^{(k+1)} \in$

\mathbb{R}^n such that $g^{(k+1)T}(x^* - x^{(k)}) \leq 0$. We therefore conclude that

$$x^* \in \mathcal{E}^{(k)} \cap \{z : g^{(k+1)T}(z - x^{(k)}) \leq 0\}.$$

We set $\mathcal{E}^{(k+1)}$ to be the ellipsoid of minimal volume containing the half-ellipsoid described above and compute $x^{(k+1)}$. The update is given by

$$\begin{aligned} x^{(k+1)} &= x^{(k)} - \frac{1}{n+1} P_{(k)} \tilde{g}^{(k+1)} \\ P_{(k+1)} &= \frac{n^2}{n^2 - 1} \left(P_{(k)} - \frac{2}{n+1} P_{(k)} \tilde{g}^{(k+1)} \tilde{g}^{(k+1)T} P_{(k)} \right) \end{aligned}$$

where $\tilde{g}^{(k+1)} = (1/\sqrt{g^{(k+1)T} P_{(k)} g^{(k+1)}}) g^{(k+1)}$. The stopping criterion is given by the property that

$$\sqrt{g^{(k)T} P_{(k)} g^{(k)}} \leq \epsilon \Rightarrow f(x^{(k)}) - f(x^*) \leq \epsilon.$$

26.4 Inequality-constrained minimization

At the k -th iteration of the algorithm for constrained minimization, we have a point $x^{(k)}$ at the center of an ellipsoid $\mathcal{E}^{(k)}$ as before. We also must maintain a list of values $f_{\text{best}}^{(k)}$ recording the smallest objective value of feasible iterates so far. Depending on whether or not the point $x^{(k)}$ is feasible, we perform one of two tasks:

- If $x^{(k)}$ is feasible, perform essentially the same update as in the unconstrained case, by choosing a subgradient g_0 that satisfies
- $$g_0^T(x^* - x^{(k)}) + f_0(x^{(k)}) - f_{\text{best}}^{(k)} \leq 0$$
- If $x^{(k)}$ is infeasible and violates the j -th constraint, update the ellipsoid with a feasibility cut. Our feasibility cut may be a subgradient g_j of f_j which must satisfy

$$g_j^T(z - x^{(k)}) + f_j(x^{(k)}) \leq 0$$

for all feasible z .

26.4.1 Application to linear programming

Inequality-constrained minimization of a function that is zero everywhere corresponds to the problem of simply identifying any feasible point. It turns out that any linear programming problem can be reduced to a linear feasibility problem (e.g. minimize the zero function subject to some linear inequality and equality constraints). One way to do this is by combining the primal and dual linear programs together into one program, and adding the

additional (linear) constraint that the value of the primal solution is no worse than the value of the dual solution. Another way is to treat the objective of the linear program as an additional constraint, and use binary search to find the optimum value.

26.5 Performance

The ellipsoid method is used on low-dimensional problems, such as planar location problems, where it is numerically stable. On even “small”-sized problems, it suffers from numerical instability and poor performance in practice.

However, the ellipsoid method is an important theoretical technique in combinatorial optimization. In computational complexity theory, the ellipsoid algorithm is attractive because its complexity depends on the number of columns and the digital size of the coefficients, but not on the number of rows. In the 21st century, interior-point algorithms with similar properties have appeared.

26.6 Notes

- [1] M. Grötschel, L. Lovász, A. Schrijver: *Geometric Algorithms and Combinatorial Optimization*, Springer, 1988.
- [2] L. Lovász: *An Algorithmic Theory of Numbers, Graphs, and Convexity*, CBMS-NSF Regional Conference Series in Applied Mathematics 50, SIAM, Philadelphia, Pennsylvania, 1986.
- [3] V. Chandru and M.R.Rao, Linear Programming, Chapter 31 in *Algorithms and Theory of Computation Handbook*, edited by M. J. Atallah, CRC Press 1999, 31-1 to 31-37.
- [4] V. Chandru and M.R.Rao, Integer Programming, Chapter 32 in *Algorithms and Theory of Computation Handbook*, edited by M.J. Atallah, CRC Press 1999, 32-1 to 32-45.

26.7 Further reading

- Dmitris Alevras and Manfred W. Padberg, *Linear Optimization and Extensions: Problems and Extensions*, Universitext, Springer-Verlag, 2001. (Problems from Padberg with solutions.)
- V. Chandru and M.R.Rao, Linear Programming, Chapter 31 in *Algorithms and Theory of Computation Handbook*, edited by M.J. Atallah, CRC Press 1999, 31-1 to 31-37.
- V. Chandru and M.R.Rao, Integer Programming, Chapter 32 in *Algorithms and Theory of Computation Handbook*, edited by M.J. Atallah, CRC Press 1999, 32-1 to 32-45.

- George B. Dantzig and Mukund N. Thapa. 1997. *Linear programming 1: Introduction.* Springer-Verlag.
- George B. Dantzig and Mukund N. Thapa. 2003. *Linear Programming 2: Theory and Extensions.* Springer-Verlag.
- M. Grötschel, L. Lovász, A. Schrijver: *Geometric Algorithms and Combinatorial Optimization,* Springer, 1988
- L. Lovász: *An Algorithmic Theory of Numbers, Graphs, and Convexity*, CBMS-NSF Regional Conference Series in Applied Mathematics 50, SIAM, Philadelphia, Pennsylvania, 1986
- Katta G. Murty, *Linear Programming*, Wiley, 1983.
- M. Padberg, *Linear Optimization and Extensions*, Second Edition, Springer-Verlag, 1999.
- Christos H. Papadimitriou and Kenneth Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Corrected republication with a new preface, Dover.
- Alexander Schrijver, *Theory of Linear and Integer Programming*. John Wiley & sons, 1998, ISBN 0-471-98232-6

26.8 External links

- EE364b, a Stanford course homepage

Chapter 27

Karmarkar's algorithm

Karmarkar's algorithm is an algorithm introduced by Narendra Karmarkar in 1984 for solving linear programming problems. It was the first reasonably efficient algorithm that solves these problems in polynomial time. The ellipsoid method is also polynomial time but proved to be inefficient in practice.

Where n is the number of variables and L is the number of bits of input to the algorithm, Karmarkar's algorithm requires $O(n^{3.5}L)$ operations on $O(L)$ digit numbers, as compared to $O(n^6L)$ such operations for the ellipsoid algorithm. The runtime of Karmarkar's algorithm is thus

$$O(n^{3.5}L^2 \cdot \log L \cdot \log \log L)$$

using FFT-based multiplication (see Big O notation).

Karmarkar's algorithm falls within the class of interior point methods: the current guess for the solution does not follow the boundary of the feasible set as in the simplex method, but it moves through the interior of the feasible region, improving the approximation of the optimal solution by a definite fraction with every iteration, and converging to an optimal solution with rational data.^[1]

27.1 The Algorithm

Consider a Linear Programming problem in matrix form:

The algorithm determines the next feasible direction toward optimality and scales back by a factor $0 < \gamma \leq 1$.

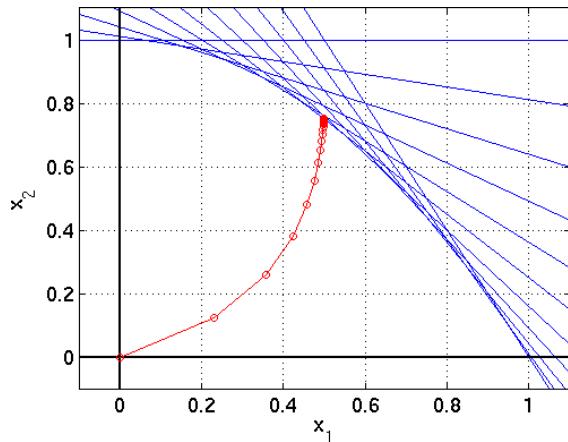
Karmarkar's algorithm is rather complicated. Interested readers can refer.^{[2][3][4] [5] [6] [7] [8]} A simplified version, called the affine-scaling method, analyzed by others,^[9] can be described succinctly as follows. Note that the affine-scaling algorithm, while applicable to small scale problems, is not a polynomial time algorithm. For large scale and complex problems the original approach needs to be followed. Karmarkar also has extended the methodology^{[10][11][12][13]} to solve problems with integer constraints and non-convex problems.^[14]

Algorithm Affine-Scaling Input: A, b, c, x^0 , stopping criterion, γ . $k \leftarrow 0$ **do while** stopping criterion **not satisfied** $v^k \leftarrow b - Ax^k$ $D_v \leftarrow \text{diag}(v_1^k, \dots, v_m^k)$ $h_x \leftarrow$

$(A^T D_v^{-1} A)^{-1} c$ $h_v \leftarrow -Ah_x$ **if** $h_v \geq 0$ **then return** unbounded **end if** $\alpha \leftarrow \gamma \cdot \min\{-v_i^k/(h_v)_i \mid (h_v)_i < 0, i = 1, \dots, m\}$ $x^{k+1} \leftarrow x^k + \alpha h_x$ $k \leftarrow k + 1$ **end do**

- " \leftarrow " is a shorthand for "changes to". For instance, "*largest* \leftarrow *item*" means that the value of *largest* changes to the value of *item*.
- "**return**" terminates the algorithm and outputs the value that follows.

27.2 Example



Example solution

Consider the linear program

That is, there are 2 variables x_1, x_2 and 11 constraints associated with varying values of p . This figure shows each iteration of the algorithm as red circle points. The constraints are shown as blue lines.

27.3 Patent controversy - Can Mathematics be patented?

At the time he invented the algorithm, Narendra Karmarkar was employed by AT&T. After applying the algorithm to optimizing AT&T's telephone network,^[15]

they realized that his invention could be of practical importance. In April 1985, AT&T promptly applied for a patent on Karmarkar's algorithm and that became more fuel for the ongoing controversy over the issue of software patents.^[16] This left many mathematicians uneasy, such as Ronald Rivest (himself one of the holders of the patent on the RSA algorithm), who expressed the opinion that research proceeded on the basis that algorithms should be free. Even before the patent was actually granted, some claimed that there might have been prior art that was applicable.^[17]

Mathematicians who specialize in numerical analysis such as Philip Gill and others claimed that Karmarkar's algorithm is equivalent to a projected Newton barrier method with a logarithmic barrier function, if the parameters are chosen suitably.^[18] However, Gill's argument is flawed, insofar as the method they describe does not even qualify as an "algorithm", since it requires choices of parameters that don't follow from the internal logic of the method, but rely on external guidance, essentially from Karmarkar's algorithm.^[19] Furthermore, Karmarkar's contributions are considered far from obvious in light of all prior work, including Fiacco-McCormick, Gill and others cited by Saltzman.^{[19][20][21]} The patent was debated in the U.S. Senate and granted in recognition of the essential originality of Karmarkar's work, as U.S. Patent 4,744,026: "Methods and apparatus for efficient resource allocation" in May 1988. AT&T supplied the KORBX system^[22] ^[23] based on this patent to the Pentagon,^{[24][25]} which enabled them to solve mathematical programming problems which were previously considered unsolvable.

Opponents of software patents have further alleged that the patents ruined the positive interaction cycles that previously characterized the relationship between researchers in linear programming and industry, and specifically it isolated Karmarkar himself from the network of mathematical researchers in his field.^[26]

The patent itself expired in April 2006, and the algorithm is presently in the public domain.

27.4 References

- Ilan Adler, Narendra Karmarkar, Mauricio G.C. Resende and Geraldo Veiga (1989). "An Implementation of Karmarkar's Algorithm for Linear Programming". *Mathematical Programming*, Vol 44, p. 297–335.
 - Narendra Karmarkar (1984). "A New Polynomial Time Algorithm for Linear Programming", *Combinatorica*, Vol 4, nr. 4, p. 373–395.
- [1] Strang, Gilbert (1 June 1987). "Karmarkar's algorithm and its place in applied mathematics". *The Mathematical Intelligencer* (New York: Springer) **9** (2): 4–10. doi:10.1007/BF03025891. ISSN 0343-6993. MR "883185".
- [2] <http://dl.acm.org/citation.cfm?id=808695>
- [3] <http://link.springer.com/article/10.1007%2FBF02579150>
- [4] <http://onlinelibrary.wiley.com/doi/10.1002/j.1538-7305.1989.tb00316.x/abstract>
- [5] Karmarkar N.K., An InteriorPoint Approach to NPComplete Problems Part I, AMS series on Contemporary Mathematics 114, pp. 297308 (June 1990). <http://www.ams.org/books/conm/114/conm114-endmatter.pdf>
- [6] Karmarkar, N.K., Riemannian Geometry Underlying Interior Point Methods for Linear programming, AMS series on Contemporary Mathematics 114, pp. 5175 (June 1990). <http://www.ams.org/books/conm/114/conm114-endmatter.pdf>
- [7] Karmarkar N. K., Lagarias, J.C., Slutsman, L., and Wang, P., Power Series Variants of KarmarkarType Algorithm, AT & T technical Journal 68, No. 3, May/June (1989).
- [8] <http://sanghv.com/download/Ebook/Machine%20Learning/Kalyanmoy%20Deb,%20Arnab%20Bhattacharya,%20Nirupam%20Chakraborti,%20Partha%20Chakroborty,%20Swagatam%20Das,%20Joydeep%20Dutta,%20Santosh%20K.%20Gupta,%20Ashu%20Jain,%20Varun%20Aggarwal,%20Juergen%20Branke,%20Sushil%20J.%20Louis,%20Kay%20Chen%20Tan%20Simulated%20E.pdf>
- [9] Robert J. Vanderbei; Meketon, Marc, Freedman, Barry (1986). "A Modification of Karmarkar's Linear Programming Algorithm". *Algorithmica* **1**: 395–407. doi:10.1007/BF01840454.
- [10] Karmarkar, N.K.,Interior Point Methods in Optimization, Proceedings of the Second International Conference on Industrial and Applied Mathematics, SIAM, pp. 160181 (1991)
- [11] Karmarkar, N. K. and Kamath, A. P., A continuous Approach to Deriving Upper Bounds in Quadratic Maximization Problems with Integer Constraints, Recent Advances in Global Optimization, pp. 125140, Princeton University Press (1992).
- [12] 26. Karmarkar, N. K., Thakur, S. A., An Interior Point Approach to a Tensor Optimisation Problem with Application to Upper Bounds in Integer Quadratic Optimization Problems, Proceedings of Second Conference on Integer Programming and Combinatorial Optimisation, (May 1992).
- [13] 27. Kamath, A., Karmarkar, N. K., A Continuous Method for Computing Bounds in Integer Quadratic Optimisation Problems, Journal of Global Optimization (1992).
- [14] Karmarkar, N. K., Beyond Convexity: New Perspectives in Computational Optimization. Springer Lecture Notes in Computer Science LNCS 6457, Dec 2010

- [15] Sinha L.P., Freedman, B. A., Karmarkar, N. K., Putcha, A., and Ramakrishnan K.G., Overseas Network Planning, Proceedings of the Third International Network Planning Symposium, NETWORKS' 86, Tarpon Springs, Florida (June 1986).
- [16] Kolata, Gina (1989-03-12). “IDEAS & TRENDS; Mathematicians Are Troubled by Claims on Their Recipes”. *The New York Times*.
- [17] Various posts by Matthew Saltzman, Clemson University
- [18] Gill, Philip E.; Murray, Walter, Saunders, Michael A., Tomlin, J. A. and Wright, Margaret H. (1986). “On projected Newton barrier methods for linear programming and an equivalence to Karmarkar’s projective method”. *Mathematical Programming* **36** (2): 183–209. doi:10.1007/BF02592025.
- [19] Andrew Chin (2009). “On Abstraction and Equivalence in Software Patent Doctrine: A Response to Bessen, Meurer and Klemens”. *Journal Of Intellectual Property Law* **16**: 214–223.
- [20] Mark A. Paley (1995). “The Karmarkar Patent: Why Congress Should “Open the Door” to Algorithms as Patentable Subject Matter”. 22 COMPUTER L. REP. 7
- [21] Margaret H. Wright (2004). “The Interior-Point Revolution in Optimization: History, Recent Developments, and Lasting Consequences”. *Bulletins of the American Mathematical Society* **42**: 39–56.
- [22] Marc S. Meketon; Y.C. Cheng, D.J. Houck, J.M.Liu, L. Slutzman, Robert J. Vanderbei, and P. Wang (1989). “The AT&T KORBX System”. *AT&T Technical Journal* **68**: 7–19.
- [23] <http://www.nytimes.com/1988/08/13/business/big-at-t-computer-for-complexities.html>
- [24] <http://www.apnewsarchive.com/1989/Military-Is-First-Announced-Customer-Of-AT-T-Software/> id-8a376783cd62cdf141de700a7c948f61
- [25] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=70419&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D70419
- [26] “**数学：Karmarkar专利和软件 – 有数学成为可专利吗？** (Konno Hiroshi: The Kamarkar Patent and Software – Has Mathematics Become Patentable?)”. FFII. Retrieved 2008-06-27.

Chapter 28

Simplex algorithm

This article is about the linear programming algorithm. For the non-linear optimization heuristic, see [Nelder–Mead method](#).

In mathematical optimization, Dantzig's **simplex algorithm** (or **simplex method**) is a popular algorithm for linear programming.^{[1][2][3][4][5]} The journal *Computing in Science and Engineering* listed it as one of the top 10 algorithms of the twentieth century.^[6]

The name of the algorithm is derived from the concept of a **simplex** and was suggested by T. S. Motzkin.^[7] Simplices are not actually used in the method, but one interpretation of it is that it operates on simplicial *cones*, and these become proper simplices with an additional constraint.^{[8][9][10][11]} The simplicial cones in question are the corners (i.e., the neighborhoods of the vertices) of a geometric object called a **polytope**. The shape of this polytope is defined by the **constraints** applied to the objective function.

28.1 Overview

Further information: [Linear programming](#)

The simplex algorithm operates on linear programs in *standard form*, that is linear programming problems of the form,

Maximize

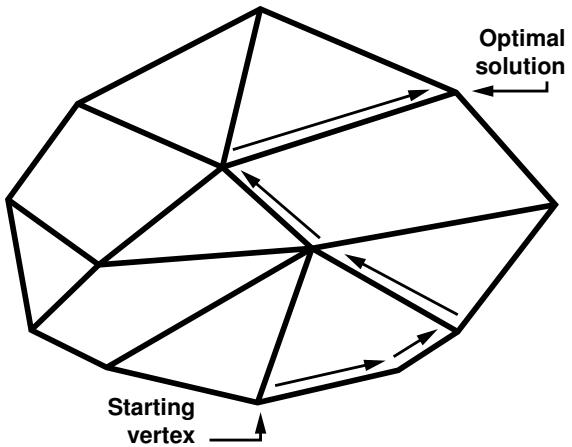
$$\mathbf{c}^T \cdot \mathbf{x}$$

Subject to

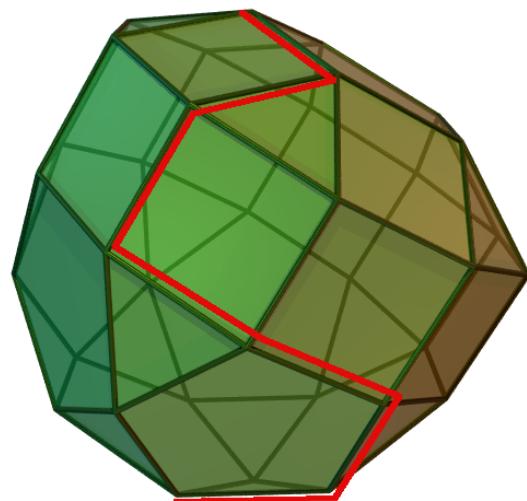
$$\mathbf{Ax} \leq \mathbf{b}, x_i \geq 0$$

with $x = (x_1, \dots, x_n)$ the variables of the problem, $c = (c_1, \dots, c_n)$ are the coefficients of the objective function, A a $p \times n$ matrix, and $b = (b_1, \dots, b_p)$ constants with $b_j \geq 0$. There is a straightforward process to convert any linear program into one in standard form so this results in no loss of generality.

In geometric terms, the **feasible region**



A system of linear inequalities defines a polytope as a feasible region. The simplex algorithm begins at a starting vertex and moves along the edges of the polytope until it reaches the vertex of the optimum solution.



Polyhedron of simplex algorithm in 3D

$$\mathbf{Ax} \leq \mathbf{b}, x_i \geq 0$$

is a (possibly unbounded) convex polytope. There is a simple characterization of the extreme points or vertices of this polytope, namely $x = (x_1, \dots, x_n)$ is an extreme point if and only if the subset of column vectors A_i corresponding to the nonzero entries of x ($x_i \neq 0$) are linearly independent.^[12] In this context such a point is known as a *basic feasible solution* (BFS).

It can be shown that for a linear program in standard form, if the objective function has a minimum value on the feasible region then it has this value on (at least) one of the extreme points.^[13] This in itself reduces the problem to a finite computation since there is a finite number of extreme points, but the number of extreme points is unmanageably large for all but the smallest linear programs.^[14]

It can also be shown that if an extreme point is not a minimum point of the objective function then there is an edge containing the point so that the objective function is strictly decreasing on the edge moving away from the point.^[15] If the edge is finite then the edge connects to another extreme point where the objective function has a smaller value, otherwise the objective function is unbounded below on the edge and the linear program has no solution. The simplex algorithm applies this insight by walking along edges of the polytope to extreme points with lower and lower objective values. This continues until the minimum value is reached or an unbounded edge is visited, concluding that the problem has no solution. The algorithm always terminates because the number of vertices in the polytope is finite; moreover since we jump between vertices always in the same direction (that of the objective function), we hope that the number of vertices visited will be small.^[15]

The solution of a linear program is accomplished in two steps. In the first step, known as Phase I, a starting extreme point is found. Depending on the nature of the program this may be trivial, but in general it can be solved by applying the simplex algorithm to a modified version of the original program. The possible results of Phase I are either a basic feasible solution is found or that the feasible region is empty. In the latter case the linear program is called *infeasible*. In the second step, Phase II, the simplex algorithm is applied using the basic feasible solution found in Phase I as a starting point. The possible results from Phase II are either an optimum basic feasible solution or an infinite edge on which the objective function is unbounded below.^{[3][16][17]}

28.2 Standard form

The transformation of a linear program to one in standard form may be accomplished as follows.^[18] First, for each variable with a lower bound other than 0, a new variable is introduced representing the difference between the variable and bound. The original variable can then be eliminated by substitution. For example, given the constraint

$$x_1 \geq 5$$

a new variable, y_1 , is introduced with

$$y_1 = x_1 - 5$$

$$x_1 = y_1 + 5$$

The second equation may be used to eliminate x_1 from the linear program. In this way, all lower bound constraints may be changed to non-negativity restrictions.

Second, for each remaining inequality constraint, a new variable, called a *slack variable*, is introduced to change the constraint to an equality constraint. This variable represents the difference between the two sides of the inequality and is assumed to be nonnegative. For example the inequalities

$$x_2 + 2x_3 \leq 3$$

$$-x_4 + 3x_5 \geq 2$$

are replaced with

$$x_2 + 2x_3 + s_1 = 3$$

$$-x_4 + 3x_5 - s_2 = 2$$

$$s_1, s_2 \geq 0$$

It is much easier to perform algebraic manipulation on inequalities in this form. In inequalities where \geq appears such as the second one, some authors refer to the variable introduced as a *surplus variable*.

Third, each unrestricted variable is eliminated from the linear program. This can be done in two ways, one is by solving for the variable in one of the equations in which it appears and then eliminating the variable by substitution. The other is to replace the variable with the difference of two restricted variables. For example if z_1 is unrestricted then write

$$z_1 = z_1^+ - z_1^-$$

$$z_1^+, z_1^- \geq 0$$

The equation may be used to eliminate z_1 from the linear program.

When this process is complete the feasible region will be in the form

$$\mathbf{Ax} = \mathbf{b}, x_i \geq 0$$

It is also useful to assume that the rank of \mathbf{A} is the number of rows. This results in no loss of generality since otherwise either the system $\mathbf{Ax} \geq \mathbf{b}$ has redundant equations which can be dropped, or the system is inconsistent and the linear program has no solution.^[19]

28.3 Simplex tableaux

A linear program in standard form can be represented as a *tableau* of the form

$$\begin{bmatrix} 1 & -\mathbf{c}^T & 0 \\ 0 & \mathbf{A} & \mathbf{b} \end{bmatrix}$$

The first row defines the objective function and the remaining rows specify the constraints. (Note, different authors use different conventions as to the exact layout.) If the columns of \mathbf{A} can be rearranged so that it contains the identity matrix of order p (the number of rows in \mathbf{A}) then the tableau is said to be in *canonical form*.^[20] The variables corresponding to the columns of the identity matrix are called *basic variables* while the remaining variables are called *nonbasic* or *free variables*. If the nonbasic variables are assumed to be 0, then the values of the basic variables are easily obtained as entries in \mathbf{b} and this solution is a basic feasible solution.

Conversely, given a basic feasible solution, the columns corresponding to the nonzero variables can be expanded to a nonsingular matrix. If the corresponding tableau is multiplied by the inverse of this matrix then the result is a tableau in canonical form.^[21]

Let

$$\begin{bmatrix} 1 & -\mathbf{c}_B^T & -\mathbf{c}_D^T & 0 \\ 0 & I & \mathbf{D} & \mathbf{b} \end{bmatrix}$$

be a tableau in canonical form. Additional *row-addition transformations* can be applied to remove the coefficients \mathbf{c}_D^T

B from the objective function. This process is called *pivoting out* and results in a canonical tableau

$$\begin{bmatrix} 1 & 0 & -\bar{\mathbf{c}}_D^T & z_B \\ 0 & I & \mathbf{D} & \mathbf{b} \end{bmatrix}$$

where z_B is the value of the objective function at the corresponding basic feasible solution. The updated coefficients, also known as *relative cost coefficients*, are the rates of change of the objective function with respect to the nonbasic variables.^[16]

28.4 Pivot operations

The geometrical operation of moving from a basic feasible solution to an adjacent basic feasible solution is implemented as a *pivot operation*. First, a nonzero *pivot element* is selected in a nonbasic column. The row containing this element is *multiplied* by its reciprocal to change this element to 1, and then multiples of the row are added to the other rows to change the other entries in the column

to 0. The result is that, if the pivot element is in row r , then the column becomes the r -th column of the identity matrix. The variable for this column is now a basic variable, replacing the variable which corresponded to the r -th column of the identity matrix before the operation. In effect, the variable corresponding to the pivot column enters the set of basic variables and is called the *entering variable*, and the variable being replaced leaves the set of basic variables and is called the *leaving variable*. The tableau is still in canonical form but with the set of basic variables changed by one element.^{[3][16]}

28.5 Algorithm

Let a linear program be given by a canonical tableau. The simplex algorithm proceeds by performing successive pivot operations which each give an improved basic feasible solution; the choice of pivot element at each step is largely determined by the requirement that this pivot improve the solution.

28.5.1 Entering variable selection

Since the entering variable will, in general, increase from 0 to a positive number, the value of the objective function will decrease if the derivative of the objective function with respect to this variable is negative. Equivalently, the value of the objective function is decreased if the pivot column is selected so that the corresponding entry in the objective row of the tableau is positive.

If there is more than one column so that the entry in the objective row is positive then the choice of which one to add to the set of basic variables is somewhat arbitrary and several *entering variable choice rules*^[22] have been developed.

If all the entries in the objective row are less than or equal to 0 then no choice of entering variable can be made and the solution is in fact optimal. It is easily seen to be optimal since the objective row now corresponds to an equation of the form

$$z(\mathbf{x}) = z_B + \text{variables nonbasic to corresponding terms nonnegative}$$

Note that by changing the entering variable choice rule so that it selects a column where the entry in the objective row is negative, the algorithm is changed so that it finds the maximum of the objective function rather than the minimum.

28.5.2 Leaving variable selection

Once the pivot column has been selected, the choice of pivot row is largely determined by the requirement that

the resulting solution be feasible. First, only positive entries in the pivot column are considered since this guarantees that the value of the entering variable will be non-negative. If there are no positive entries in the pivot column then the entering variable can take any nonnegative value with the solution remaining feasible. In this case the objective function is unbounded below and there is no minimum.

Next, the pivot row must be selected so that all the other basic variables remain positive. A calculation shows that this occurs when the resulting value of the entering variable is at a minimum. In other words, if the pivot column is c , then the pivot row r is chosen so that

$$b_r/a_{rc}$$

is the minimum over all r so that $a_{rc} > 0$. This is called the *minimum ratio test*.^[22] If there is more than one row for which the minimum is achieved then a *dropping variable choice rule*^[23] can be used to make the determination.

28.5.3 Example

See also: Revised simplex algorithm § Numerical example

Consider the linear program

Minimize

$$Z = -2x - 3y - 4z$$

Subject to

$$3x + 2y + z \leq 10$$

$$2x + 5y + 3z \leq 15$$

$$x, y, z \geq 0$$

With the addition of slack variables s and t , this is represented by the canonical tableau

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 0 & 0 & 0 \\ 0 & 3 & 2 & 1 & 1 & 0 & 10 \\ 0 & 2 & 5 & 3 & 0 & 1 & 15 \end{bmatrix}$$

where columns 5 and 6 represent the basic variables s and t and the corresponding basic feasible solution is

$$x = y = z = 0, s = 10, t = 15.$$

Columns 2, 3, and 4 can be selected as pivot columns, for this example column 4 is selected. The values of z resulting from the choice of rows 2 and 3 as pivot rows are $10/1 = 10$ and $15/3 = 5$ respectively. Of these the minimum is

5, so row 3 must be the pivot row. Performing the pivot produces

$$\begin{bmatrix} 1 & -\frac{2}{3} & -\frac{11}{3} & 0 & 0 & -\frac{4}{3} & -20 \\ 0 & \frac{7}{3} & \frac{1}{3} & 0 & 1 & -\frac{1}{3} & 5 \\ 0 & \frac{2}{3} & \frac{5}{3} & 1 & 0 & \frac{1}{3} & 5 \end{bmatrix}$$

Now columns 4 and 5 represent the basic variables z and s and the corresponding basic feasible solution is

$$x = y = t = 0, z = 5, s = 5.$$

For the next step, there are no positive entries in the objective row and in fact

$$Z = -20 + \frac{2}{3}x + \frac{11}{3}y + \frac{4}{3}t$$

so the minimum value of Z is -20 .

28.6 Finding an initial canonical tableau

In general, a linear program will not be given in canonical form and an equivalent canonical tableau must be found before the simplex algorithm can start. This can be accomplished by the introduction of *artificial variables*. Columns of the identity matrix are added as column vectors for these variables. If the b value for a constraint equation is negative, the equation is negated before adding the identity matrix columns. This does not change the set of feasible solutions or the optimal solution, and it ensures that the slack variables will constitute an initial feasible solution. The new tableau is in canonical form but it is not equivalent to the original problem. So a new objective function, equal to the sum of the artificial variables, is introduced and the simplex algorithm is applied to find the minimum; the modified linear program is called the *Phase I* problem.^[24]

The simplex algorithm applied to the Phase I problem must terminate with a minimum value for the new objective function since, being the sum of nonnegative variables, its value is bounded below by 0. If the minimum is 0 then the artificial variables can be eliminated from the resulting canonical tableau producing a canonical tableau equivalent to the original problem. The simplex algorithm can then be applied to find the solution; this step is called *Phase II*. If the minimum is positive then there is no feasible solution for the Phase I problem where the artificial variables are all zero. This implies that the feasible region for the original problem is empty, and so the original problem has no solution.^{[3][16][25]}

28.6.1 Example

Consider the linear program

Minimize

$$Z = -2x - 3y - 4z$$

Subject to

$$3x + 2y + z = 10$$

$$2x + 5y + 3z = 15$$

$$x, y, z \geq 0$$

This is represented by the (non-canonical) tableau

$$\left[\begin{array}{ccccc} 1 & 2 & 3 & 4 & 0 \\ 0 & 3 & 2 & 1 & 10 \\ 0 & 2 & 5 & 3 & 15 \end{array} \right]$$

Introduce artificial variables u and v and objective function $W = u + v$, giving a new tableau

$$\left[\begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 1 & 2 & 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 3 & 2 & 1 & 1 & 0 & 10 \\ 0 & 0 & 2 & 5 & 3 & 0 & 1 & 15 \end{array} \right]$$

Note that the equation defining the original objective function is retained in anticipation of Phase II.

After pricing out this becomes

$$\left[\begin{array}{ccccccc} 1 & 0 & 5 & 7 & 4 & 0 & 0 & 25 \\ 0 & 1 & 2 & 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 3 & 2 & 1 & 1 & 0 & 10 \\ 0 & 0 & 2 & 5 & 3 & 0 & 1 & 15 \end{array} \right]$$

Select column 5 as a pivot column, so the pivot row must be row 4, and the updated tableau is

$$\left[\begin{array}{ccccccc} 1 & 0 & \frac{7}{3} & \frac{1}{3} & 0 & 0 & -\frac{4}{3} & 5 \\ 0 & 1 & -\frac{2}{3} & -\frac{11}{3} & 0 & 0 & -\frac{4}{3} & -20 \\ 0 & 0 & \frac{7}{3} & \frac{1}{3} & 0 & 1 & -\frac{1}{3} & 5 \\ 0 & 0 & \frac{3}{3} & \frac{3}{3} & 1 & 0 & \frac{1}{3} & 5 \end{array} \right]$$

Now select column 3 as a pivot column, for which row 3 must be the pivot row, to get

$$\left[\begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 1 & 0 & -\frac{25}{7} & 0 & \frac{2}{7} & -\frac{10}{7} & -\frac{130}{7} \\ 0 & 0 & 1 & \frac{1}{7} & 0 & \frac{3}{7} & -\frac{1}{7} & -\frac{15}{7} \\ 0 & 0 & 0 & \frac{11}{7} & 1 & -\frac{2}{7} & \frac{3}{7} & \frac{25}{7} \end{array} \right]$$

The artificial variables are now 0 and they may be dropped giving a canonical tableau equivalent to the original problem:

$$\left[\begin{array}{ccccc} 1 & 0 & -\frac{25}{7} & 0 & -\frac{130}{7} \\ 0 & 1 & \frac{1}{7} & 0 & \frac{15}{7} \\ 0 & 0 & \frac{11}{7} & 1 & \frac{25}{7} \end{array} \right]$$

This is, fortuitously, already optimal and the optimum value for the original linear program is $-130/7$.

28.7 Advanced topics

28.7.1 Implementation

Main article: [Revised simplex algorithm](#)

The tableau form used above to describe the algorithm lends itself to an immediate implementation in which the tableau is maintained as a rectangular $(m+1)$ -by- $(m+n+1)$ array. It is straightforward to avoid storing the m explicit columns of the identity matrix that will occur within the tableau by virtue of \mathbf{B} being a subset of the columns of $[\mathbf{A}, \mathbf{I}]$. This implementation is referred to as the "*standard simplex algorithm*". The storage and computation overhead are such that the standard simplex method is a prohibitively expensive approach to solving large linear programming problems.

In each simplex iteration, the only data required are the first row of the tableau, the (pivot) column of the tableau corresponding to the entering variable and the right-hand-side. The latter can be updated using the pivotal column and the first row of the tableau can be updated using the (pivot) row corresponding to the leaving variable. Both the pivotal column and pivotal row may be computed directly using the solutions of linear systems of equations involving the matrix \mathbf{B} and a matrix-vector product using \mathbf{A} . These observations motivate the "*revised simplex algorithm*", for which implementations are distinguished by their invertible representation of \mathbf{B} .^[4]

In large linear-programming problems \mathbf{A} is typically a [sparse matrix](#) and, when the resulting sparsity of \mathbf{B} is exploited when maintaining its invertible representation, the revised simplex algorithm is a much more efficient than the standard simplex method. Commercial simplex solvers are based on the revised simplex algorithm.^{[4][25][26][27][28]}

28.7.2 Degeneracy: Stalling and cycling

If the values of all basic variables are strictly positive, then a pivot must result in an improvement in the objective value. When this is always the case no set of basic variables occurs twice and the simplex algorithm must

terminate after a finite number of steps. Basic feasible solutions where at least one of the *basic* variables is zero are called *degenerate* and may result in pivots for which there is no improvement in the objective value. In this case there is no actual change in the solution but only a change in the set of basic variables. When several such pivots occur in succession, there is no improvement; in large industrial applications, degeneracy is common and such "*stalling*" is notable. Worse than stalling is the possibility the same set of basic variables occurs twice, in which case, the deterministic pivoting rules of the simplex algorithm will produce an infinite loop, or "cycle". While degeneracy is the rule in practice and stalling is common, cycling is rare in practice. A discussion of an example of practical cycling occurs in Padberg.^[25] Bland's rule prevents cycling and thus guarantees that the simplex algorithm always terminates.^{[25][29][30]} Another pivoting algorithm, the *criss-cross* algorithm never cycles on linear programs.^[31]

28.7.3 Efficiency

The simplex method is remarkably efficient in practice and was a great improvement over earlier methods such as Fourier–Motzkin elimination. However, in 1972, Klee and Minty^[32] gave an example showing that the worst-case complexity of simplex method as formulated by Dantzig is exponential time. Since then, for almost every variation on the method, it has been shown that there is a family of linear programs for which it performs badly. It is an open question if there is a variation with polynomial time, or even sub-exponential worst-case complexity.^{[33][34]}

Analyzing and quantifying the observation that the simplex algorithm is efficient in practice, even though it has exponential worst-case complexity, has led to the development of other measures of complexity. The simplex algorithm has polynomial-time average-case complexity under various probability distributions, with the precise average-case performance of the simplex algorithm depending on the choice of a probability distribution for the random matrices.^{[34][35]} Another approach to studying "typical phenomena" uses Baire category theory from general topology, and to show that (topologically) "most" matrices can be solved by the simplex algorithm in a polynomial number of steps. Another method to analyze the performance of the simplex algorithm studies the behavior of worst-case scenarios under small perturbation – are worst-case scenarios stable under a small change (in the sense of structural stability), or do they become tractable? Formally, this method uses random problems to which is added a Gaussian random vector ("smoothed complexity").^[36]

28.8 Other algorithms

Other algorithms for solving linear-programming problems are described in the linear-programming article. Another basis-exchange pivoting algorithm is the *criss-cross algorithm*.^{[37][38]} There are polynomial-time algorithms for linear programming that use interior point methods: These include Khachiyan's ellipsoidal algorithm, Karmarkar's projective algorithm, and path-following algorithms.^[17]

28.9 Linear-fractional programming

Main article: Linear-fractional programming

Linear-fractional programming (LFP) is a generalization of linear programming (LP) where the objective function of linear programs are linear functions and the objective function of a linear-fractional program is a ratio of two linear functions. In other words, a linear program is a fractional-linear program in which the denominator is the constant function having the value one everywhere. A linear-fractional program can be solved by a variant of the simplex algorithm^{[39][40][41][42]} or by the *criss-cross algorithm*.^[43]

28.10 See also

- Criss-cross algorithm
- Fourier–Motzkin elimination
- Karmarkar's algorithm
- Nelder–Mead simplicial heuristic
- Pivoting rule of Bland, which avoids cycling

28.11 Notes

- [1] Murty, Katta G. (1983). *Linear programming*. New York: John Wiley & Sons Inc. pp. xix+482. ISBN 0-471-09725-X. MR 720547.
- [2] Richard W. Cottle, ed. *The Basic George B. Dantzig*. Stanford Business Books, Stanford University Press, Stanford, California, 2003. (Selected papers by George B. Dantzig)
- [3] George B. Dantzig and Mukund N. Thapa. 1997. *Linear programming I: Introduction*. Springer-Verlag.
- [4] George B. Dantzig and Mukund N. Thapa. 2003. *Linear Programming 2: Theory and Extensions*. Springer-Verlag.

- [5] Michael J. Todd (February 2002). “The many facets of linear programming”. *Mathematical Programming* **91** (3). (Invited survey, from the International Symposium on Mathematical Programming.)
- [6] Computing in Science and Engineering, volume 2, no. 1, 2000 html version
- [7] Murty (1983, Comment 2.2)
- [8] Murty (1983, Note 3.9)
- [9] Stone, Richard E.; Tovey, Craig A. (1991). “The simplex and projective scaling algorithms as iteratively reweighted least squares methods”. *SIAM Review* **33** (2): 220–237. doi:10.1137/1033049. JSTOR 2031142. MR 1124362.
- [10] Stone, Richard E.; Tovey, Craig A. (1991). “Erratum: The simplex and projective scaling algorithms as iteratively reweighted least squares methods”. *SIAM Review* **33** (3): 461. doi:10.1137/1033100. JSTOR 2031443. MR 1124362.
- [11] Strang, Gilbert (1 June 1987). “Karmarkar’s algorithm and its place in applied mathematics”. *The Mathematical Intelligencer* (New York: Springer) **9** (2): 4–10. doi:10.1007/BF03025891. ISSN 0343-6993. MR “883185”.
- [12] Murty (1983, Theorem 3.1)
- [13] Murty (1983, Theorem 3.3)
- [14] Murty (1983, p. 143, Section 3.13)
- [15] Murty (1983, p. 137, Section 3.8)
- [16] Evar D. Nering and Albert W. Tucker, 1993, *Linear Programs and Related Problems*, Academic Press. (elementary)
- [17] Robert J. Vanderbei, *Linear Programming: Foundations and Extensions*, 3rd ed., International Series in Operations Research & Management Science, Vol. 114, Springer Verlag, 2008. ISBN 978-0-387-74387-5.
- [18] Murty (1983, Section 2.2)
- [19] Murty (1983, p. 173)
- [20] Murty (1983, section 2.3.2)
- [21] Murty (1983, section 3.12)
- [22] Murty (1983, p. 66)
- [23] Murty (1983, p. 67)
- [24] Murty (1983, p. 60)
- [25] M. Padberg, *Linear Optimization and Extensions*, Second Edition, Springer-Verlag, 1999.
- [26] Dimitris Alevras and Manfred W. Padberg, *Linear Optimization and Extensions: Problems and Extensions*, Universitext, Springer-Verlag, 2001. (Problems from Padberg with solutions.)
- [27] Maros, István; Mitra, Gautam (1996). “Simplex algorithms”. In J. E. Beasley. *Advances in linear and integer programming*. Oxford Science. pp. 1–46. MR 1438309.
- [28] Maros, István (2003). *Computational techniques of the simplex method*. International Series in Operations Research & Management Science **61**. Boston, MA: Kluwer Academic Publishers. pp. xx+325. ISBN 1-4020-7332-1. MR 1960274.
- [29] Bland, Robert G. (May 1977). “New finite pivoting rules for the simplex method”. *Mathematics of Operations Research* **2** (2): 103–107. doi:10.1287/moor.2.2.103. JSTOR 3689647. MR 459599.
- [30] Murty (1983, p. 79)
- [31] There are abstract optimization problems, called oriented matroid programs, on which Bland’s rule cycles (incorrectly) while the criss-cross algorithm terminates correctly.
- [32] Klee, Victor; Minty, George J. (1972). “How good is the simplex algorithm?”. In Shisha, Oved. *Inequalities III (Proceedings of the Third Symposium on Inequalities held at the University of California, Los Angeles, Calif., September 1–9, 1969, dedicated to the memory of Theodore S. Motzkin)*. New York-London: Academic Press. pp. 159–175. MR 332165.
- [33] Christos H. Papadimitriou and Kenneth Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Corrected republication with a new preface, Dover. (computer science)
- [34] Alexander Schrijver, *Theory of Linear and Integer Programming*. John Wiley & sons, 1998, ISBN 0-471-98232-6 (mathematical)
- [35] The simplex algorithm takes on average D steps for a cube. Borgwardt (1987): Borgwardt, Karl-Heinz (1987). *The simplex method: A probabilistic analysis*. Algorithms and Combinatorics (Study and Research Texts) **1**. Berlin: Springer-Verlag. pp. xii+268. ISBN 3-540-17096-0. MR 868467.
- [36] Spielman, Daniel; Teng, Shang-Hua (2001). “Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time”. *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*. ACM. pp. 296–305. arXiv:cs/0111050. doi:10.1145/380752.380813. ISBN 978-1-58113-349-3.
- [37] Terlaky, Tamás; Zhang, Shu Zhong (1993). “Pivot rules for linear programming: A Survey on recent theoretical developments”. *Annals of Operations Research* (Springer Netherlands). **46–47** (1): 203–233. doi:10.1007/BF02096264. ISSN 0254-5330. MR 1260019. CiteSeerX: 10.1.1.36.7658.
- [38] Fukuda, Komei; Terlaky, Tamás (1997). Thomas M. Liebling and Dominique de Werra, ed. “Criss-cross methods: A fresh view on pivot algorithms”. *Mathematical Programming: Series B* **79** (1–3) (Amsterdam: North-Holland Publishing Co.). pp. 369–395. doi:10.1007/BF02614325. MR 1464775.
- [39] Murty (1983, Chapter 3.20 (pp. 160–164) and pp. 168 and 179)

- [40] Chapter five: Craven, B. D. (1988). *Fractional programming*. Sigma Series in Applied Mathematics **4**. Berlin: Heldermann Verlag. p. 145. ISBN 3-88538-404-3. MR 949209.
- [41] Kruk, Serge; Wolkowicz, Henry (1999). “Pseudolinear programming”. *SIAM Review* **41** (4): 795–805. doi:10.1137/S0036144598335259. JSTOR 2653207. MR 1723002.
- [42] Mathis, Frank H.; Mathis, Lenora Jane (1995). “A non-linear programming algorithm for hospital management”. *SIAM Review* **37** (2): 230–234. doi:10.1137/1037046. JSTOR 2132826. MR 1343214.
- [43] Illés, Tibor; Szirmai, Ákos; Terlaky, Tamás (1999). “The finite criss-cross method for hyperbolic programming”. *European Journal of Operational Research* **114** (1): 198–214. doi:10.1016/S0377-2217(98)00049-6. ISSN 0377-2217. PDF preprint.
- Simplex Method A tutorial for Simplex Method with examples (also two-phase and M-method).
- Example of Simplex Procedure for a Standard Linear Programming Problem by Thomas McFarland of the University of Wisconsin-Whitewater.
- PHPSimplex: online tool to solve Linear Programming Problems by Daniel Izquierdo and Juan José Ruiz of the University of Málaga (UMA, Spain)
- simplex-m Online Simplex Solver

28.12 References

- Murty, Katta G. (1983). *Linear programming*. New York: John Wiley & Sons, Inc. pp. xix+482. ISBN 0-471-09725-X. MR 720547.

28.13 Further reading

These introductions are written for students of computer science and operations research:

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 29.3: The simplex algorithm, pp. 790–804.
- Frederick S. Hillier and Gerald J. Lieberman: *Introduction to Operations Research*, 8th edition. McGraw-Hill. ISBN 0-07-123828-X
- Rardin, Ronald L. (1997). *Optimization in operations research*. Prentice Hall. p. 919. ISBN 0-02-398415-5.

28.14 External links

- An Introduction to Linear Programming and the Simplex Algorithm by Spyros Reveliotis of the Georgia Institute of Technology.
- Greenberg, Harvey J., *Klee-Minty Polytope Shows Exponential Time Complexity of Simplex Method* University of Colorado at Denver (1997) PDF download

Chapter 29

Revised simplex method

In mathematical optimization, the **revised simplex method** is a variant of George Dantzig's simplex method for linear programming.

The revised simplex method is mathematically equivalent to the standard simplex method but differs in implementation. Instead of maintaining a tableau which explicitly represents the constraints adjusted to a set of basic variables, it maintains a representation of a **basis** of the **matrix** representing the constraints. The matrix-oriented approach allows for greater computational efficiency by enabling sparse matrix operations.^[1]

29.1 Problem formulation

For the rest of the discussion, it is assumed that a linear programming problem has been converted into the following standard form:

$$\begin{aligned} \text{minimize } & c^T x \\ \text{to subject } & Ax = b, x \geq 0 \end{aligned}$$

where $A \in \mathbf{R}^{m \times n}$. Without loss of generality, it is assumed that the constraint matrix A has full row rank and that the problem is feasible, i.e., there is at least one $x \geq 0$ such that $Ax = b$. If A is rank-deficient, either there are redundant constraints, or the problem is infeasible. Both situations can be handled by a presolve step.

29.2 Algorithmic description

29.2.1 Optimality conditions

For linear programming, the **Karush–Kuhn–Tucker conditions** are both **necessary** and **sufficient** for optimality. The KKT conditions of a linear programming problem in the standard form is

$$\begin{aligned} Ax &= b, \\ A^T \lambda + s &= c, \\ x &\geq 0, \\ s &\geq 0, \\ s^T x &= 0 \end{aligned}$$

where λ and s are the **Lagrange multipliers** associated with the constraints $Ax = b$ and $x \geq 0$, respectively.^[2] The last condition, which is equivalent to $s_i x_i = 0$ for all $1 < i < n$, is called the *complementary slackness condition*.

By what is sometimes known as the *fundamental theorem of linear programming*, a vertex x of the feasible polytope can be identified by being a basis B of A chosen from the latter's columns.^[lower-alpha 1] Since A has full rank, B is nonsingular. Without loss of generality, assume that $A = [B \ N]$. Then x is given by

$$x = \begin{bmatrix} x_B \\ x_N \end{bmatrix} = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$$

where $xB \geq 0$. Partition c and s accordingly into

$$\begin{aligned} c &= \begin{bmatrix} c_B \\ c_N \end{bmatrix}, \\ s &= \begin{bmatrix} s_B \\ s_N \end{bmatrix}. \end{aligned}$$

To satisfy the complementary slackness condition, let $sB = 0$. It follows that

$$\begin{aligned} B^T \lambda &= c_B, \\ N^T \lambda + s_N &= c_N, \end{aligned}$$

which implies that

$$\begin{aligned} \lambda &= (B^T)^{-1} c_B, \\ s_N &= c_N - N^T \lambda. \end{aligned}$$

If $sN \geq 0$ at this point, the KKT conditions are satisfied, and thus x is optimal.

29.2.2 Pivot operation

If the KKT conditions are violated, a *pivot operation* consisting of introducing a column of \mathbf{N} into the basis at the expense of an existing column in \mathbf{B} is performed. In the absence of *degeneracy*, a pivot operation always results in a strict decrease in $\mathbf{c}^T \mathbf{x}$. Therefore, if the problem is bounded, the revised simplex method must terminate at an optimal vertex after repeated pivot operations because there are only a finite number of vertices.^[4]

Select an index $m < q \leq n$ such that $s_q < 0$ as the *entering index*. The corresponding column of \mathbf{A} , \mathbf{A}_q , will be moved into the basis, and x_q will be allowed to increase from zero. It can be shown that

$$\frac{\partial(\mathbf{c}^T \mathbf{x})}{\partial x_q} = s_q,$$

i.e., every unit increase in x_q will result in a decrease by $-s_q$ in $\mathbf{c}^T \mathbf{x}$.^[5] Since

$$\mathbf{Bx_B} + \mathbf{A}_q x_q = \mathbf{b},$$

$\mathbf{x_B}$ must be correspondingly decreased by $\Delta \mathbf{x_B} = \mathbf{B}^{-1} \mathbf{A}_q x_q$ subject to $\mathbf{x_B} - \Delta \mathbf{x_B} \geq \mathbf{0}$. Let $\mathbf{d} = \mathbf{B}^{-1} \mathbf{A}_q$. If $\mathbf{d} \leq \mathbf{0}$, no matter how much x_q is increased, $\mathbf{x_B} - \Delta \mathbf{x_B}$ will stay nonnegative. Hence, $\mathbf{c}^T \mathbf{x}$ can be arbitrarily decreased, and thus the problem is unbounded. Otherwise, select an index $p = \operatorname{argmin}_{1 \leq i \leq m} \{x_i/d_i \mid d_i > 0\}$ as the *leaving index*. This choice effectively increases x_q from zero until x_p is reduced to zero while maintaining feasibility. The pivot operation concludes with replacing \mathbf{A}_p with \mathbf{A}_q in the basis.

29.3 Numerical example

See also: Simplex method § Example

Consider a linear program where

$$\mathbf{c} = [-2 \quad -3 \quad -4 \quad 0 \quad 0]^T,$$

$$\mathbf{A} = \begin{bmatrix} 3 & 2 & 1 & 1 & 0 \\ 2 & 5 & 3 & 0 & 1 \end{bmatrix},$$

$$\mathbf{b} = \begin{bmatrix} 10 \\ 15 \end{bmatrix}.$$

Let

$$\mathbf{B} = [\mathbf{A}_4 \quad \mathbf{A}_5],$$

$$\mathbf{N} = [\mathbf{A}_1 \quad \mathbf{A}_2 \quad \mathbf{A}_3]$$

initially, which corresponds to a feasible vertex $\mathbf{x} = [0 \quad 0 \quad 10 \quad 15]^T$. At this moment,

$$\boldsymbol{\lambda} = [0 \quad 0]^T,$$

$$\mathbf{s}_N = [-2 \quad -3 \quad -4]^T.$$

Choose $q = 3$ as the entering index. Then $\mathbf{d} = [1 \quad 3]^T$, which means a unit increase in x_3 will result in x_4 and x_5 being decreased by 1 and 3, respectively. Therefore, x_3 is increased to 5, at which point x_5 is reduced to zero, and $p = 5$ becomes the leaving index.

After the pivot operation,

$$\mathbf{B} = [\mathbf{A}_3 \quad \mathbf{A}_4],$$

$$\mathbf{N} = [\mathbf{A}_1 \quad \mathbf{A}_2 \quad \mathbf{A}_5].$$

Correspondingly,

$$\mathbf{x} = [0 \quad 0 \quad 5 \quad 5 \quad 0]^T,$$

$$\boldsymbol{\lambda} = [0 \quad -4/3]^T,$$

$$\mathbf{s}_N = [2/3 \quad 11/3 \quad 4].$$

A positive \mathbf{s}_N indicates that \mathbf{x} is now optimal.

29.4 Practical issues

29.4.1 Degeneracy

See also: Simplex method § Degeneracy: Stalling and cycling

Because the revised simplex method is mathematically equivalent to the simplex method, it also suffers from degeneracy, where a pivot operation does not result in a decrease in $\mathbf{c}^T \mathbf{x}$, and a chain of pivot operations causes the basis to cycle. A perturbation or lexicographic strategy can be used to prevent cycling and guarantee termination.^[6]

29.4.2 Basis representation

Two types of linear systems involving \mathbf{B} are present in the revised simplex method:

$$\mathbf{Bz} = \mathbf{y},$$

$$\mathbf{B}^T \mathbf{z} = \mathbf{y}.$$

Instead of refactorizing \mathbf{B} , usually an LU factorization is directly updated after each pivot operation, for which purpose there exist several strategies such as the Forrest–Tomlin and Bartels–Golub methods. However, the amount of data representing the updates as well as numerical errors builds up over time and makes periodic refactorization necessary.^{[1][7]}

29.5 Notes and references

29.5.1 Notes

- [1] The same theorem also states that the feasible polytope has at least one vertex and that there is at least one vertex which is optimal.^[3]

29.5.2 References

- [1] Morgan 1997, §2.
- [2] Nocedal & Wright 2006, p. 358, Eq. 13.4.
- [3] Nocedal & Wright 2006, p. 363, Theorem 13.2.
- [4] Nocedal & Wright 2006, p. 370, Theorem 13.4.
- [5] Nocedal & Wright 2006, p. 369, Eq. 13.24.
- [6] Nocedal & Wright 2006, p. 381, §13.5.
- [7] Nocedal & Wright 2006, p. 372, §13.4.

29.5.3 Bibliography

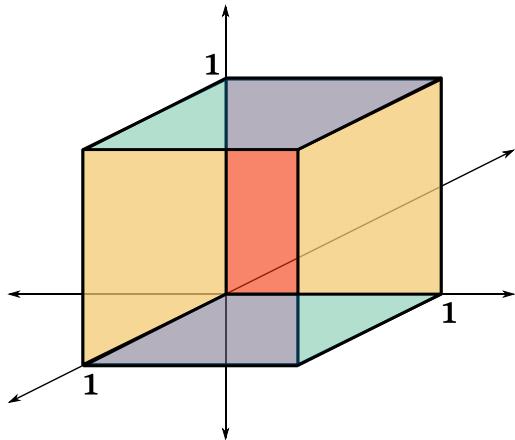
- Morgan, S. S. (1997). *A Comparison of Simplex Method Algorithms* (MSc thesis). University of Florida.
- Nocedal, J.; Wright, S. J. (2006). Mikosch, T. V.; Resnick, S. I.; Robinson, S. M., eds. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering (2nd ed.). New York, NY, USA: Springer. ISBN 978-0-387-30303-1.

Chapter 30

Criss-cross algorithm

This article is about an algorithm for mathematical optimization. For the naming of chemicals, see crisscross method.

In mathematical optimization, the **criss-cross algo-**



The criss-cross algorithm visits all 8 corners of the Klee–Minty cube in the worst case. It visits 3 additional corners on average. The Klee–Minty cube is a perturbation of the cube shown here.

rithm denotes a family of algorithms for linear programming. Variants of the criss-cross algorithm also solve more general problems with linear inequality constraints and nonlinear objective functions; there are criss-cross algorithms for linear-fractional programming problems,^{[1][2]} quadratic-programming problems, and linear complementarity problems.^[3]

Like the simplex algorithm of George B. Dantzig, the criss-cross algorithm is not a polynomial-time algorithm for linear programming. Both algorithms visit all 2^D corners of a (perturbed) cube in dimension D , the Klee–Minty cube (after Victor Klee and George J. Minty), in the worst case.^{[4][5]} However, when it is started at a random corner, the criss-cross algorithm on average visits only D additional corners.^{[6][7][8]} Thus, for the three-dimensional cube, the algorithm visits all 8 corners in the worst case and exactly 3 additional corners on average.

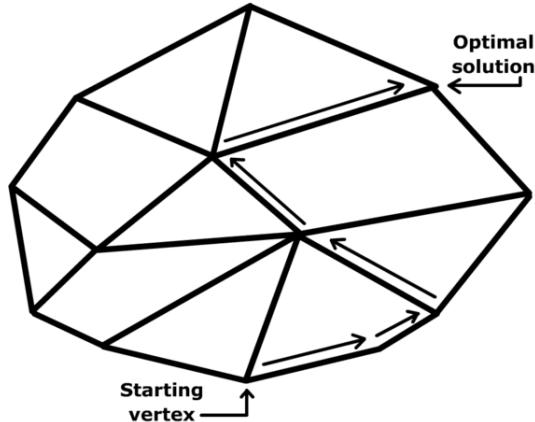
30.1 History

The criss-cross algorithm was published independently by Tamás Terlaky^[9] and by Zhe-Min Wang;^[10] related algorithms appeared in unpublished reports by other authors.^[3]

30.2 Comparison with the simplex algorithm for linear optimization

See also: Linear programming, Simplex algorithm and Bland's rule

In linear programming, the criss-cross algorithm pivots



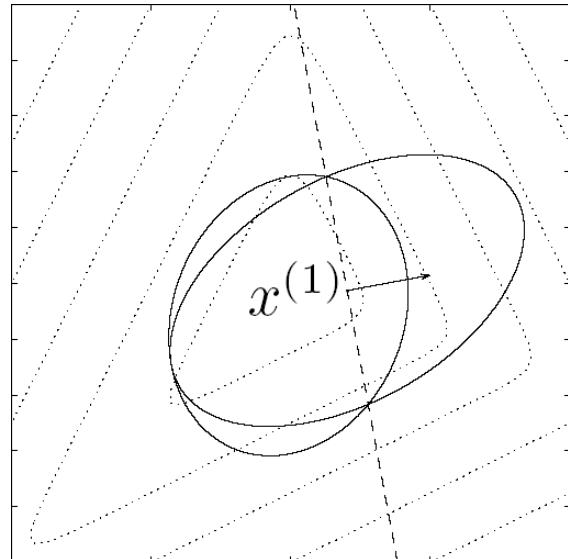
In its second phase, the simplex algorithm crawls along the edges of the polytope until it finally reaches an optimum vertex. The criss-cross algorithm considers bases that are not associated with vertices, so that some iterates can be in the interior of the feasible region, like interior-point algorithms; the criss-cross algorithm can also have infeasible iterates outside the feasible region.

between a sequence of bases but differs from the simplex algorithm of George Dantzig. The simplex algorithm first finds a (primal-) feasible basis by solving a "phase-one problem"; in "phase two", the simplex algorithm pivots between a sequence of basic feasible solutions so that the objective function is non-decreasing with each pivot, ter-

minating when with an optimal solution (also finally finding a “dual feasible” solution).^{[3][11]}

The criss-cross algorithm is simpler than the simplex algorithm, because the criss-cross algorithm only has one-phase. Its pivoting rules are similar to the least-index pivoting rule of Bland.^[12] Bland’s rule uses only signs of coefficients rather than their (real-number) order when deciding eligible pivots. Bland’s rule selects an entering variables by comparing values of reduced costs, using the real-number ordering of the eligible pivots.^{[12][13]} Unlike Bland’s rule, the criss-cross algorithm is “purely combinatorial”, selecting an entering variable and a leaving variable by considering only the signs of coefficients rather than their real-number ordering.^{[3][11]} The criss-cross algorithm has been applied to furnish constructive proofs of basic results in **real linear algebra**, such as the **lemma of Farkas**.^[14]

While most simplex variants are monotonic in the objective (strictly in the non-degenerate case), most variants of the criss-cross algorithm lack a monotone merit function which can be a disadvantage in practice.



The worst-case computational complexity of Khachiyan’s ellipsoidal algorithm is a polynomial. The criss-cross algorithm has exponential complexity.

30.3 Description

The criss-cross algorithm works on a standard pivot tableau (or on-the-fly calculated parts of a tableau, if implemented like the revised simplex method). In a general step, if the tableau is primal or dual infeasible, it selects one of the infeasible rows / columns as the pivot row / column using an index selection rule. An important property is that the selection is made on the union of the infeasible indices and the standard version of the algorithm does not distinguish column and row indices (that is, the column indices basic in the rows). If a row is selected then the algorithm uses the index selection rule to identify a position to a dual type pivot, while if a column is selected then it uses the index selection rule to find a row position and carries out a primal type pivot.

30.4 Computational complexity: Worst and average cases

The time complexity of an algorithm counts the number of arithmetic operations sufficient for the algorithm to solve the problem. For example, Gaussian elimination requires on the order of D^3 operations, and so it is said to have polynomial time-complexity, because its complexity is bounded by a **cubic polynomial**. There are examples of algorithms that do not have polynomial-time complexity. For example, a generalization of Gaussian elimination called **Buchberger’s algorithm** has for its complexity an exponential function of the problem data (the degree of the polynomials and the number of variables of the multivariate polynomials). Because exponential

functions eventually grow much faster than polynomial functions, an exponential complexity implies that an algorithm has slow performance on large problems.

Several algorithms for linear programming—Khachiyan’s ellipsoidal algorithm, Karmarkar’s projective algorithm, and central-path algorithms—have polynomial time-complexity (in the worst case and thus **on average**). The ellipsoidal and projective algorithms were published before the criss-cross algorithm.

However, like the simplex algorithm of Dantzig, the criss-cross algorithm is *not* a polynomial-time algorithm for linear programming. Terlaky’s criss-cross algorithm visits all the 2^D corners of a (perturbed) cube in dimension D , according to a paper of Roos; Roos’s paper modifies the **Klee–Minty construction** of a **cube** on which the simplex algorithm takes 2^D steps.^{[3][4][5]} Like the simplex algorithm, the criss-cross algorithm visits all 8 corners of the three-dimensional cube in the worst case.

When it is initialized at a random corner of the cube, the criss-cross algorithm visits only D additional corners, however, according to a 1994 paper by Fukuda and Namiki.^{[6][7]} Trivially, the simplex algorithm takes on average D steps for a cube.^{[8][15]} Like the simplex algorithm, the criss-cross algorithm visits exactly 3 additional corners of the three-dimensional cube on average.

30.5 Variants

The criss-cross algorithm has been extended to solve more general problems than linear programming problems.

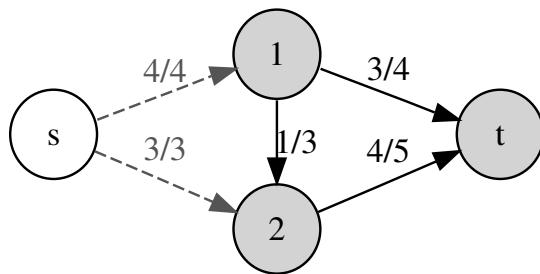
30.5.1 Other optimization problems with linear constraints

There are variants of the criss-cross algorithm for linear programming, for quadratic programming, and for the linear-complementarity problem with “sufficient matrices”;^{[3][6][16][17][18][19]} conversely, for linear complementarity problems, the criss-cross algorithm terminates finitely only if the matrix is a sufficient matrix.^{[18][19]} A sufficient matrix is a generalization both of a positive-definite matrix and of a P-matrix, whose principal minors are each positive.^{[18][19][20]} The criss-cross algorithm has been adapted also for linear-fractional programming.^{[1][2]}

30.5.2 Vertex enumeration

The criss-cross algorithm was used in an algorithm for enumerating all the vertices of a polytope, which was published by David Avis and Komei Fukuda in 1992.^[21] Avis and Fukuda presented an algorithm which finds the v vertices of a polyhedron defined by a nondegenerate system of n linear inequalities in D dimensions (or, dually, the v facets of the convex hull of n points in D dimensions, where each facet contains exactly D given points) in time $O(nDv)$ and $O(nD)$ space.^[22]

30.5.3 Oriented matroids



The max-flow min-cut theorem states that the maximum flow through a network is exactly the capacity of its minimum cut. This theorem can be proved using the criss-cross algorithm for oriented matroids.

The criss-cross algorithm is often studied using the theory of oriented matroids (OMs), which is a combinatorial abstraction of linear-optimization theory.^{[17][23]} Indeed, Bland's pivoting rule was based on his previous papers on oriented-matroid theory. However, Bland's rule exhibits cycling on some oriented-matroid linear-programming problems.^[17] The first purely combinatorial algorithm for linear programming was devised by Michael J. Todd.^{[17][24]} Todd's algorithm was developed not only for linear-programming in the setting of oriented matroids, but also for quadratic-programming problems and linear-complementarity problems.^{[17][24]}

Todd's algorithm is complicated even to state, unfortunately, and its finite-convergence proofs are somewhat complicated.^[17]

The criss-cross algorithm and its proof of finite termination can be simply stated and readily extend the setting of oriented matroids. The algorithm can be further simplified for *linear feasibility problems*, that is for linear systems with nonnegative variables; these problems can be formulated for oriented matroids.^[14] The criss-cross algorithm has been adapted for problems that are more complicated than linear programming: There are oriented-matroid variants also for the quadratic-programming problem and for the linear-complementarity problem.^{[3][16][17]}

30.6 Summary

The criss-cross algorithm is a simply stated algorithm for linear programming. It was the second fully combinatorial algorithm for linear programming. The partially combinatorial simplex algorithm of Bland cycles on some (nonrealizable) oriented matroids. The first fully combinatorial algorithm was published by Todd, and it is also like the simplex algorithm in that it preserves feasibility after the first feasible basis is generated; however, Todd's rule is complicated. The criss-cross algorithm is not a simplex-like algorithm, because it need not maintain feasibility. The criss-cross algorithm does not have polynomial time-complexity, however.

Researchers have extended the criss-cross algorithm for many optimization-problems, including linear-fractional programming. The criss-cross algorithm can solve quadratic programming problems and linear complementarity problems, even in the setting of oriented matroids. Even when generalized, the criss-cross algorithm remains simply stated.

30.7 See also

- Jack Edmonds (pioneer of combinatorial optimization and oriented-matroid theorist; doctoral advisor of Komei Fukuda)

30.8 Notes

- [1] Illés, Szirmai & Terlaky (1999)
- [2] Stancu-Minasian, I. M. (August 2006). “A sixth bibliography of fractional programming”. *Optimization* **55** (4): 405–428. doi:10.1080/02331930600819613. MR 2258634.
- [3] Fukuda & Terlaky (1997)
- [4] Roos (1990)

- [5] Klee, Victor; Minty, George J. (1972). “How good is the simplex algorithm?”. In Shisha, Oved. *Inequalities III (Proceedings of the Third Symposium on Inequalities held at the University of California, Los Angeles, Calif., September 1–9, 1969, dedicated to the memory of Theodore S. Motzkin)*. New York-London: Academic Press. pp. 159–175. MR 332165.
- [6] Fukuda & Terlaky (1997, p. 385)
- [7] Fukuda & Namiki (1994, p. 367)
- [8] The simplex algorithm takes on average D steps for a cube. Borgwardt (1987): Borgwardt, Karl-Heinz (1987). *The simplex method: A probabilistic analysis*. Algorithms and Combinatorics (Study and Research Texts) **1**. Berlin: Springer-Verlag. pp. xii+268. ISBN 3-540-17096-0. MR 868467.
- [9] Terlaky (1985) and Terlaky (1987)
- [10] Wang (1987)
- [11] Terlaky & Zhang (1993)
- [12] Bland, Robert G. (May 1977). “New finite pivoting rules for the simplex method”. *Mathematics of Operations Research* **2** (2): 103–107. doi:10.1287/moor.2.2.103. JSTOR 3689647. MR 459599.
- [13] Bland’s rule is also related to an earlier least-index rule, which was proposed by Katta G. Murty for the linear complementarity problem, according to Fukuda & Namiki (1994).
- [14] Klafszky & Terlaky (1991)
- [15] More generally, for the simplex algorithm, the expected number of steps is proportional to D for linear-programming problems that are randomly drawn from the Euclidean unit sphere, as proved by Borgwardt and by Smale.
- [16] Fukuda & Namiki (1994)
- [17] Björner, Anders; Las Vergnas, Michel; Sturmfels, Bernd; White, Neil; Ziegler, Günter (1999). “10 Linear programming”. *Oriented Matroids*. Cambridge University Press. pp. 417–479. doi:10.1017/CBO9780511586507. ISBN 978-0-521-77750-6. MR 1744046.
- [18] den Hertog, D.; Roos, C.; Terlaky, T. (1 July 1993). “The linear complementarity problem, sufficient matrices, and the criss-cross method” (pdf). *Linear Algebra and its Applications* **187**: 1–14. doi:10.1016/0024-3795(93)90124-7.
- [19] Csizmadia, Zsolt; Illés, Tibor (2006). “New criss-cross type algorithms for linear complementarity problems with sufficient matrices” (pdf). *Optimization Methods and Software* **21** (2): 247–266. doi:10.1080/10556780500095009. MR 2195759.
- [20] Cottle, R. W.; Pang, J.-S.; Venkateswaran, V. (March–April 1989). “Sufficient matrices and the linear complementarity problem”. *Linear Algebra and its Applications*. 114–115: 231–249. doi:10.1016/0024-3795(89)90463-1. MR 986877.
- [21] Avis & Fukuda (1992, p. 297)
- [22] The v vertices in a simple arrangement of n hyperplanes in D dimensions can be found in $O(n^2 D v)$ time and $O(nD)$ space complexity.
- [23] The theory of oriented matroids was initiated by R. Tyrrell Rockafellar. (Rockafellar 1969):
Rockafellar, R. T. (1969). “The elementary vectors of a subspace of R^N (1967)”. In R. C. Bose and T. A. Dowling. *Combinatorial Mathematics and its Applications*. The University of North Carolina Monograph Series in Probability and Statistics (4). Chapel Hill, North Carolina: University of North Carolina Press. pp. 104–127. MR 278972. PDF reprint.
Rockafellar was influenced by the earlier studies of Albert W. Tucker and George J. Minty. Tucker and Minty had studied the sign patterns of the matrices arising through the pivoting operations of Dantzig’s simplex algorithm.
- [24] Todd, Michael J. (1985). “Linear and quadratic programming in oriented matroids”. *Journal of Combinatorial Theory. Series B* **39** (2): 105–133. doi:10.1016/0095-8956(85)90042-5. MR 811116.

30.9 References

- Avis, David; Fukuda, Komei (December 1992). “A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra”. *Discrete and Computational Geometry* **8** (ACM Symposium on Computational Geometry (North Conway, NH, 1991) number 1): 295–313. doi:10.1007/BF02293050. MR 1174359.
- Csizmadia, Zsolt; Illés, Tibor (2006). “New criss-cross type algorithms for linear complementarity problems with sufficient matrices” (pdf). *Optimization Methods and Software* **21** (2): 247–266. doi:10.1080/10556780500095009. MR 2195759.
- Fukuda, Komei; Namiki, Makoto (March 1994). “On extremal behaviors of Murty’s least index method”. *Mathematical Programming* **64** (1): 365–370. doi:10.1007/BF01582581. MR 1286455.
- Fukuda, Komei; Terlaky, Tamás (1997). Liebling, Thomas M.; de Werra, Dominique, eds. “Criss-cross methods: A fresh view on pivot algorithms”. *Mathematical Programming: Series B* (Amsterdam: North-Holland Publishing Co.) **79** (Papers from the 16th International Symposium on Mathematical Programming held in Lausanne, 1997, number 1–3): 369–395. doi:10.1007/BF02614325. MR 1464775. Postscript preprint.
- den Hertog, D.; Roos, C.; Terlaky, T. (1 July 1993). “The linear complementarity problem, sufficient matrices, and the criss-cross method” (pdf). *Linear Algebra and its Applications* **187**:

- 1–14. doi:10.1016/0024-3795(93)90124-7. MR 1221693.
- Illés, Tibor; Szirmai, Ákos; Terlaky, Tamás (1999). “The finite criss-cross method for hyperbolic programming”. *European Journal of Operational Research* **114** (1): 198–214. doi:10.1016/S0377-2217(98)00049-6. Zbl 0953.90055. Postscript preprint.
 - Klafszky, Emil; Terlaky, Tamás (June 1991). “The role of pivoting in proving some fundamental theorems of linear algebra” (postscript). *Linear Algebra and its Applications* **151**: 97–118. doi:10.1016/0024-3795(91)90356-2. MR 1102142.
 - Roos, C. (1990). “An exponential example for Terlaky’s pivoting rule for the criss-cross simplex method”. *Mathematical Programming. Series A* **46** (1): 79–84. doi:10.1007/BF01585729. MR 1045573.
 - Terlaky, T. (1985). “A convergent criss-cross method”. *Optimization: A Journal of Mathematical Programming and Operations Research* **16** (5): 683–690. doi:10.1080/02331938508843067. ISSN 0233-1934. MR 798939.
 - Terlaky, Tamás (1987). “A finite crisscross method for oriented matroids”. *Journal of Combinatorial Theory. Series B* **42** (3): 319–327. doi:10.1016/0095-8956(87)90049-9. ISSN 0095-8956. MR 888684.
 - Terlaky, Tamás; Zhang, Shu Zhong (1993) [1991]. “Pivot rules for linear programming: A Survey on recent theoretical developments”. *Annals of Operations Research* (Springer Netherlands). 46–47 (Degeneracy in optimization problems, number 1): 203–233. doi:10.1007/BF02096264. ISSN 0254-5330. MR 1260019. CiteSeerX: 10.1.1.36.7658.
 - Wang, Zhe Min (1987). “A finite conformal-elimination free algorithm over oriented matroid programming”. *Chinese Annals of Mathematics (Shuxue Niankan B Ji)*. Series B **8** (1): 120–125. ISSN 0252-9599. MR 886756.

30.10 External links

- Komei Fukuda (ETH Zentrum, Zurich) with publications
- Tamás Terlaky (Lehigh University) with publications

Chapter 31

Lemke's algorithm

In mathematical optimization, **Lemke's algorithm** is a procedure for solving linear complementarity problems, and more generally mixed linear complementarity problems.

Lemke's algorithm is of pivoting or basis-exchange type. Similar algorithms can compute Nash equilibria for two-person matrix and bimatrix games.

31.1 References

- Cottle, Richard W.; Pang, Jong-Shi; Stone, Richard E. (1992). *The linear complementarity problem*. Computer Science and Scientific Computing. Boston, MA: Academic Press, Inc. pp. xxiv+762 pp. ISBN 0-12-192350-9. MR 1150683
- Murty, K. G. (1988). *Linear complementarity, linear and nonlinear programming*. Sigma Series in Applied Mathematics 3. Berlin: Heldermann Verlag. pp. xlviii+629 pp. ISBN 3-88538-403-5. (Available for download at the website of Professor Katta G. Murty.) MR 949214

31.2 External links

- OMatrix manual on Lemke
- Chris Hecker's GDC presentation on MLCPs and Lemke
- Linear Complementarity and Mathematical (Non-linear) Programming
- Siconos/Numerics open-source GPL implementation in C of Lemke's algorithm and other methods to solve LCPs and MLCPs

Chapter 32

Approximation algorithm

In computer science and operations research, **approximation algorithms** are algorithms used to find approximate solutions to **optimization problems**. Approximation algorithms are often associated with **NP-hard** problems; since it is unlikely that there can ever be efficient polynomial-time exact algorithms solving NP-hard problems, one settles for polynomial-time sub-optimal solutions. Unlike **heuristics**, which usually only find reasonably good solutions reasonably fast, one wants provable solution quality and provable run-time bounds. Ideally, the approximation is optimal up to a small constant factor (for instance within 5% of the optimal solution). Approximation algorithms are increasingly being used for problems where exact polynomial-time algorithms are known but are too expensive due to the input size. A typical example for an approximation algorithm is the one for **vertex cover** in **graphs**: find an uncovered edge and add *both* endpoints to the vertex cover, until none remain. It is clear that the resulting cover is at most twice as large as the optimal one. This is a **constant factor approximation algorithm** with a factor of 2.

NP-hard problems vary greatly in their approximability; some, such as the **bin packing problem**, can be approximated within any factor greater than 1 (such a family of approximation algorithms is often called a **polynomial time approximation scheme** or **PTAS**). Others are impossible to approximate within any constant, or even polynomial factor unless **P = NP**, such as the **maximum clique problem**.

NP-hard problems can often be expressed as **integer programs** (IP) and solved exactly in **exponential time**. Many approximation algorithms emerge from the linear programming relaxation of the integer program.

Not all approximation algorithms are suitable for all practical applications. They often use IP/LP/Semidefinite solvers, complex data structures or sophisticated algorithmic techniques which lead to difficult implementation problems. Also, some approximation algorithms have impractical running times even though they are polynomial time, for example $O(n^{2^{156}})$ ^[1]. Yet the study of even very expensive algorithms is not a completely theoretical pursuit as they can yield valuable insights. A classic example is the initial PTAS for **Euclidean TSP** due

to Sanjeev Arora which had prohibitive running time, yet within a year, Arora refined the ideas into a linear time algorithm. Such algorithms are also worthwhile in some applications where the running times and cost can be justified e.g. **computational biology**, **financial engineering**, **transportation planning**, and **inventory management**. In such scenarios, they must compete with the corresponding direct IP formulations.

Another limitation of the approach is that it applies only to optimization problems and not to “pure” **decision problems** like **satisfiability**, although it is often possible to conceive optimization versions of such problems, such as the **maximum satisfiability problem** (Max SAT).

Inapproximability has been a fruitful area of research in computational complexity theory since the 1990 result of Feige, Goldwasser, Lovász, Safra and Szegedy on the inapproximability of **Independent Set**. After Arora et al. proved the **PCP theorem** a year later, it has now been shown that Johnson’s 1974 approximation algorithms for Max SAT, Set Cover, Independent Set and Coloring all achieve the optimal approximation ratio, assuming $P \neq NP$.

32.1 Performance guarantees

For some approximation algorithms it is possible to prove certain properties about the approximation of the optimum result. For example, a **ϱ -approximation algorithm** A is defined to be an algorithm for which it has been proven that the value/cost, $f(x)$, of the approximate solution $A(x)$ to an instance x will not be more (or less, depending on the situation) than a factor ϱ times the value, OPT , of an optimum solution.

$$\begin{cases} OPT \leq f(x) \leq \varrho OPT, & \text{if } \varrho > 1; \\ \varrho OPT \leq f(x) \leq OPT, & \text{if } \varrho < 1. \end{cases}$$

The factor ϱ is called the *relative performance guarantee*. An approximation algorithm has an *absolute performance guarantee* or *bounded error* c , if it has been proven for every instance x that

$$(\text{OPT} - c) \leq f(x) \leq (\text{OPT} + c).$$

Similarly, the *performance guarantee*, $R(x, y)$, of a solution y to an instance x is defined as

$$R(x, y) = \max \left(\frac{\text{OPT}}{f(y)}, \frac{f(y)}{\text{OPT}} \right),$$

where $f(y)$ is the value/cost of the solution y for the instance x . Clearly, the performance guarantee is greater than or equal to 1 and equal to 1 if and only if y is an optimal solution. If an algorithm A guarantees to return solutions with a performance guarantee of at most $r(n)$, then A is said to be an $r(n)$ -approximation algorithm and has an *approximation ratio* of $r(n)$. Likewise, a problem with an $r(n)$ -approximation algorithm is said to be $r(n)$ -*approximable* or have an approximation ratio of $r(n)$.^{[2][3]}

One may note that for minimization problems, the two different guarantees provide the same result and that for maximization problems, a relative performance guarantee of ρ is equivalent to a performance guarantee of $r = \rho^{-1}$. In the literature, both definitions are common but it is clear which definition is used since, for maximization problems, as $\rho \leq 1$ while $r \geq 1$.

The *absolute performance guarantee* P_A of some approximation algorithm A , where x refers to an instance of a problem, and where $R_A(x)$ is the performance guarantee of A on x (i.e. ρ for problem instance x) is:

$$P_A = \inf\{r \geq 1 \mid R_A(x) \leq r, \forall x\}.$$

That is to say that P_A is the largest bound on the approximation ratio, r , that one sees over all possible instances of the problem. Likewise, the *asymptotic performance ratio* R_A^∞ is:

$$R_A^\infty = \inf\{r \geq 1 \mid \exists n \in \mathbb{Z}^+, R_A(x) \leq r, \forall x, |x| \geq n\}.$$

That is to say that it is the same as the *absolute performance ratio*, with a lower bound n on the size of problem instances. These two types of ratios are used because there exist algorithms where the difference between these two is significant.

32.2 Algorithm design techniques

By now there are several standard techniques that one tries to design an approximation algorithm. These include the following ones.

- 1. Greedy algorithm

2. Local search

- 3. Enumeration and dynamic programming
- 4. Solving a convex programming relaxation to get a fractional solution. Then converting this fractional solution into a feasible solution by some appropriate rounding. The popular relaxations include the following.
 - (a) Linear programming relaxation
 - (b) Semidefinite programming relaxation
- 5. Embedding the problem in some simple metric and then solving the problem on the metric. This is also known as metric embedding.

32.3 Epsilon terms

In the literature, an approximation ratio for a maximization (minimization) problem of $c - \epsilon$ ($\min: c + \epsilon$) means that the algorithm has an approximation ratio of $c \mp \epsilon$ for arbitrary $\epsilon > 0$ but that the ratio has not (or cannot) be shown for $\epsilon = 0$. An example of this is the optimal inapproximability — inexistence of approximation — ratio of $7/8 + \epsilon$ for satisfiable MAX-3SAT instances due to Johan Håstad.^[4] As mentioned previously, when $c = 1$, the problem is said to have a polynomial-time approximation scheme.

An ϵ -term may appear when an approximation algorithm introduces a multiplicative error and a constant error while the minimum optimum of instances of size n goes to infinity as n does. In this case, the approximation ratio is $c \mp k / \text{OPT} = c \mp o(1)$ for some constants c and k . Given arbitrary $\epsilon > 0$, one can choose a large enough N such that the term $k / \text{OPT} < \epsilon$ for every $n \geq N$. For every fixed ϵ , instances of size $n < N$ can be solved by brute force, thereby showing an approximation ratio — existence of approximation algorithms with a guarantee — of $c \mp \epsilon$ for every $\epsilon > 0$.

32.4 See also

- Domination analysis considers guarantees in terms of the rank of the computed solution.
- PTAS - a type of approximation algorithm that takes the approximation ratio as a parameter
- APX is the class of problems with some constant-factor approximation algorithm
- Approximation-preserving reduction

32.5 Citations

- [1] Zych, Anna; Bilò, Davide (2011). “New Reoptimization Techniques applied to Steiner Tree Problem”. *Electronic Notes in Discrete Mathematics* 37: 387–392. doi:10.1016/j.endm.2011.05.066. ISSN 1571-0653.
- [2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi (1999). *Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties*.
- [3] Viggo Kann (1992). *On the Approximability of NP-complete Optimization Problems*.
- [4] Johan Håstad (1999). “Some Optimal Inapproximability Results”. *Journal of the ACM*.

32.6 References

- Vazirani, Vijay V. (2003). *Approximation Algorithms*. Berlin: Springer. ISBN 3-540-65367-8.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Chapter 35: Approximation Algorithms, pp. 1022–1056.
- Dorit H. Hochbaum, ed. *Approximation Algorithms for NP-Hard problems*, PWS Publishing Company, 1997. ISBN 0-534-94968-1. Chapter 9: Various Notions of Approximations: Good, Better, Best, and More
- Williamson, David P.; Shmoys, David B. (April 26, 2011), *The Design of Approximation Algorithms*, Cambridge University Press, ISBN 978-0521195270

32.7 External links

- Pierluigi Crescenzi, Viggo Kann, Magnús Halldórsson, Marek Karpinski and Gerhard Woeginger, *A compendium of NP optimization problems*.

Chapter 33

Dynamic programming

Not to be confused with **Dynamic programming language**, **Dynamic algorithm**, or **Dynamic problem**.

In mathematics, computer science, economics, and bioinformatics, **dynamic programming** is a method for solving a complex problem by breaking it down into a collection of simpler subproblems. It is applicable to problems exhibiting the properties of **overlapping subproblems**^[1] and **optimal substructure** (described below). When applicable, the method takes far less time than naive methods that don't take advantage of the subproblem overlap (like **depth-first search**).

In order to solve a given problem, using a dynamic programming approach, we need to solve different parts of the problem (subproblems), then combine the solutions of the subproblems to reach an overall solution. Often when using a more naive method, many of the subproblems are generated and solved many times. The dynamic programming approach seeks to solve each subproblem only once, thus reducing the number of computations: once the solution to a given subproblem has been computed, it is stored or "**memo-ized**": the next time the same solution is needed, it is simply looked up. This approach is especially useful when the number of repeating subproblems **grows exponentially** as a function of the size of the input.

Dynamic programming algorithms are used for optimization (for example, finding the shortest path between two points, or the fastest way to multiply many matrices). A dynamic programming algorithm will examine the previously solved subproblems and will combine their solutions to give the best solution for the given problem. The alternatives are many, such as using a **greedy algorithm**, which picks the locally optimal choice at each branch in the road. The locally optimal choice may be a poor choice for the overall solution. While a greedy algorithm does not guarantee an optimal solution, it is often faster to calculate. Fortunately, some greedy algorithms (such as **minimum spanning trees**) are proven to lead to the optimal solution.

For example, let's say that you have to get from point A to point B as fast as possible, in a given city, during rush hour. A dynamic programming algorithm will

look at finding the shortest paths to points close to A, and use those solutions to eventually find the shortest path to B. On the other hand, a greedy algorithm will start you driving immediately and will pick the road that looks the fastest at every intersection. As you can imagine, this strategy might not lead to the fastest arrival time, since you might take some "easy" streets and then find yourself hopelessly stuck in a traffic jam.

33.1 Overview

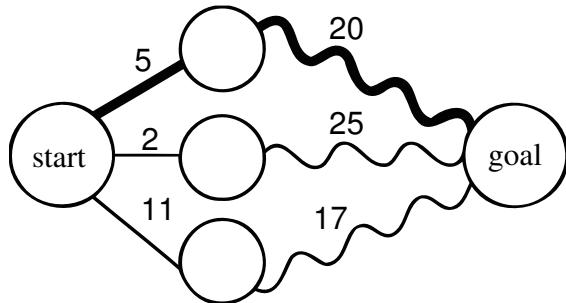


Figure 1. Finding the shortest path in a graph using optimal substructure; a straight line indicates a single edge; a wavy line indicates a shortest path between the two vertices it connects (other nodes on these paths are not shown); the bold line is the overall shortest path from start to goal.

Dynamic programming is both a mathematical optimization method and a computer programming method. In both contexts it refers to simplifying a complicated problem by breaking it down into simpler subproblems in a recursive manner. While some decision problems cannot be taken apart this way, decisions that span several points in time do often break apart recursively; Bellman called this the "**Principle of Optimality**". Likewise, in computer science, a problem that can be solved optimally by breaking it into subproblems and then recursively finding the optimal solutions to the subproblems is said to have **optimal substructure**.

If subproblems can be nested recursively inside larger problems, so that dynamic programming methods are applicable, then there is a relation between the value of the larger problem and the values of the subproblems.^[2] In

in the optimization literature this relationship is called the **Bellman equation**.

33.1.1 Dynamic programming in mathematical optimization

In terms of mathematical optimization, dynamic programming usually refers to simplifying a decision by breaking it down into a sequence of decision steps over time. This is done by defining a sequence of **value functions** V_1, V_2, \dots, V_n , with an argument y representing the **state** of the system at times i from 1 to n . The definition of $V_n(y)$ is the value obtained in state y at the last time n . The values V_i at earlier times $i = n-1, n-2, \dots, 2, 1$ can be found by working backwards, using a recursive relationship called the Bellman equation. For $i = 2, \dots, n$, V_{i-1} at any state y is calculated from V_i by maximizing a simple function (usually the sum) of the gain from a decision at time $i-1$ and the function V_i at the new state of the system if this decision is made. Since V_i has already been calculated for the needed states, the above operation yields V_{i-1} for those states. Finally, V_1 at the initial state of the system is the value of the optimal solution. The optimal values of the decision variables can be recovered, one by one, by tracking back the calculations already performed.

33.1.2 Dynamic programming in bioinformatics

Dynamic programming is widely used in bioinformatics for the tasks such as **sequence alignment**, protein folding, RNA structure prediction and protein-DNA binding. First dynamic programming algorithms for protein-DNA binding were developed in the 1970s independently by Charles DeLisi in USA^[3] and Georgii Gurskii and Alexander Zasedatelev in USSR.^[4] Recently these algorithms have become very popular in bioinformatics and computational biology, particularly in the studies of nucleosome positioning and transcription factor binding.

33.1.3 Dynamic programming in computer programming

There are two key attributes that a problem must have in order for dynamic programming to be applicable: **optimal substructure** and **overlapping subproblems**. If a problem can be solved by combining optimal solutions to *non-overlapping* subproblems, the strategy is called "divide and conquer" instead. This is why mergesort and quicksort are not classified as dynamic programming problems.

Optimal substructure means that the solution to a given optimization problem can be obtained by the combination of optimal solutions to its subproblems. Consequently,

the first step towards devising a dynamic programming solution is to check whether the problem exhibits such optimal substructure. Such optimal substructures are usually described by means of **recursion**. For example, given a graph $G=(V,E)$, the shortest path p from a vertex u to a vertex v exhibits optimal substructure: take any intermediate vertex w on this shortest path p . If p is truly the shortest path, then it can be split into subpaths p_1 from u to w and p_2 from w to v such that these, in turn, are indeed the shortest paths between the corresponding vertices (by the simple cut-and-paste argument described in *Introduction to Algorithms*). Hence, one can easily formulate the solution for finding shortest paths in a recursive manner, which is what the **Bellman–Ford** algorithm or the **Floyd–Warshall** algorithm does.

Overlapping subproblems means that the space of subproblems must be small, that is, any recursive algorithm solving the problem should solve the same subproblems over and over, rather than generating new subproblems. For example, consider the recursive formulation for generating the Fibonacci series: $F_i = F_{i-1} + F_{i-2}$, with base case $F_1 = F_2 = 1$. Then $F_{43} = F_{42} + F_{41}$, and $F_{42} = F_{41} + F_{40}$. Now F_{41} is being solved in the recursive subtrees of both F_{43} as well as F_{42} . Even though the total number of subproblems is actually small (only 43 of them), we end up solving the same problems over and over if we adopt a naive recursive solution such as this. Dynamic programming takes account of this fact and solves each subproblem only once.

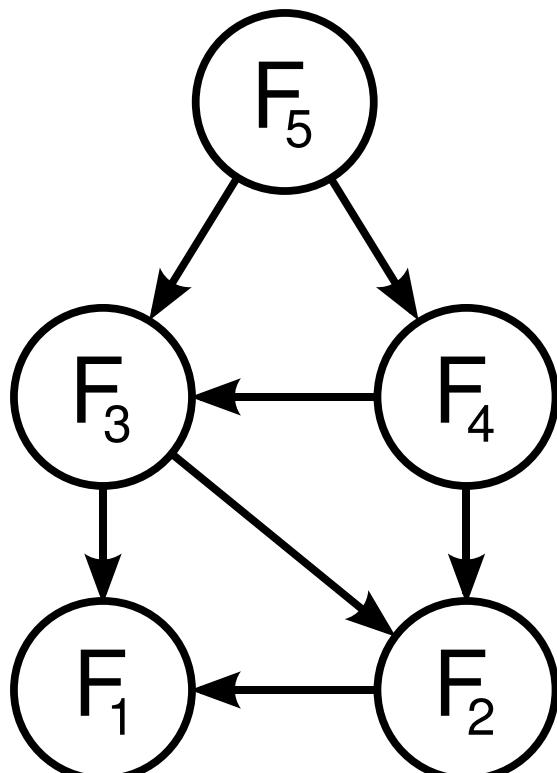


Figure 2. The subproblem graph for the Fibonacci sequence. The fact that it is not a tree indicates overlapping subproblems.

This can be achieved in either of two ways:

- **Top-down approach:** This is the direct fall-out of the recursive formulation of any problem. If the solution to any problem can be formulated recursively using the solution to its subproblems, and if its subproblems are overlapping, then one can easily memoize or store the solutions to the subproblems in a table. Whenever we attempt to solve a new subproblem, we first check the table to see if it is already solved. If a solution has been recorded, we can use it directly, otherwise we solve the subproblem and add its solution to the table.
- **Bottom-up approach:** Once we formulate the solution to a problem recursively as in terms of its subproblems, we can try reformulating the problem in a bottom-up fashion: try solving the subproblems first and use their solutions to build-on and arrive at solutions to bigger subproblems. This is also usually done in a tabular form by iteratively generating solutions to bigger and bigger subproblems by using the solutions to small subproblems. For example, if we already know the values of F_{41} and F_{40} , we can directly calculate the value of F_{42} .

Some programming languages can automatically memoize the result of a function call with a particular set of arguments, in order to speed up **call-by-name** evaluation (this mechanism is referred to as **call-by-need**). Some languages make it possible portably (e.g. Scheme, Common Lisp or Perl). Some languages have automatic memoization built in, such as tabled Prolog and J, which supports memoization with the *M.* adverb.^[5] In any case, this is only possible for a referentially transparent function.

33.2 Example: Mathematical optimization

33.2.1 Optimal consumption and saving

A mathematical optimization problem that is often used in teaching dynamic programming to economists (because it can be solved by hand^[6]) concerns a consumer who lives over the periods $t = 0, 1, 2, \dots, T$ and must decide how much to consume and how much to save in each period.

Let c_t be consumption in period t , and assume consumption yields utility $u(c_t) = \ln(c_t)$ as long as the consumer lives. Assume the consumer is impatient, so that he **discounts** future utility by a factor b each period, where $0 < b < 1$. Let k_t be **capital** in period t . Assume initial capital is a given amount $k_0 > 0$, and suppose that this period's capital and consumption determine next period's

capital as $k_{t+1} = Ak_t^a - c_t$, where A is a positive constant and $0 < a < 1$. Assume capital cannot be negative. Then the consumer's decision problem can be written as follows:

$$\max \sum_{t=0}^T b^t \ln(c_t) \text{ subject to } k_{t+1} = Ak_t^a - c_t \geq 0 \text{ for all } t = 0, 1, 2, \dots, T$$

Written this way, the problem looks complicated, because it involves solving for all the choice variables $c_0, c_1, c_2, \dots, c_T$. (Note that k_0 is not a choice variable—the consumer's initial capital is taken as given.)

The dynamic programming approach to solving this problem involves breaking it apart into a sequence of smaller decisions. To do so, we define a sequence of **value functions** $V_t(k)$, for $t = 0, 1, 2, \dots, T, T+1$ which represent the value of having any amount of capital k at each time t . Note that $V_{T+1}(k) = 0$, that is, there is (by assumption) no utility from having capital after death.

The value of any quantity of capital at any previous time can be calculated by **backward induction** using the **Bellman equation**. In this problem, for each $t = 0, 1, 2, \dots, T$, the Bellman equation is

$$V_t(k_t) = \max (\ln(c_t) + bV_{t+1}(k_{t+1})) \text{ to subject } k_{t+1} = Ak_t^a - c_t \geq 0$$

This problem is much simpler than the one we wrote down before, because it involves only two decision variables, c_t and k_{t+1} . Intuitively, instead of choosing his whole lifetime plan at birth, the consumer can take things one step at a time. At time t , his current capital k_t is given, and he only needs to choose current consumption c_t and saving k_{t+1} .

To actually solve this problem, we work backwards. For simplicity, the current level of capital is denoted as k . $V_{T+1}(k)$ is already known, so using the Bellman equation once we can calculate $V_T(k)$, and so on until we get to $V_0(k)$, which is the **value** of the initial decision problem for the whole lifetime. In other words, once we know $V_{T-j+1}(k)$, we can calculate $V_{T-j}(k)$, which is the maximum of $\ln(c_{T-j}) + bV_{T-j+1}(Ak^a - c_{T-j})$, where c_{T-j} is the choice variable and $Ak^a - c_{T-j} \geq 0$.

Working backwards, it can be shown that the value function at time $t = T - j$ is

$$V_{T-j}(k) = a \sum_{i=0}^j a^i b^i \ln k + v_{T-j}$$

where each v_{T-j} is a constant, and the optimal amount to consume at time $t = T - j$ is

$$c_{T-j}(k) = \frac{1}{\sum_{i=0}^j a^i b^i} Ak^a$$

which can be simplified to

$$c_T(k) = Ak^a, \text{ and } c_{T-1}(k) = \frac{Ak^a}{1+ab}, \text{ and} \\ c_{T-2}(k) = \frac{Ak^a}{1+ab+a^2b^2}, \text{ etc.}$$

We see that it is optimal to consume a larger fraction of current wealth as one gets older, finally consuming all remaining wealth in period T , the last period of life.

33.3 Examples: Computer algorithms

33.3.1 Dijkstra's algorithm for the shortest path problem

From a dynamic programming point of view, Dijkstra's algorithm for the **shortest path problem** is a successive approximation scheme that solves the dynamic programming functional equation for the shortest path problem by the **Reaching** method.^{[7][8][9]}

In fact, Dijkstra's explanation of the logic behind the algorithm,^[10] namely

Problem 2. Find the path of minimum total length between two given nodes P and Q

We use the fact that, if R is a node on the minimal path from P to Q , knowledge of the latter implies the knowledge of the minimal path from P to R .

is a paraphrasing of Bellman's famous Principle of Optimality in the context of the shortest path problem.

33.3.2 Fibonacci sequence

Here is a naïve implementation of a function finding the n th member of the **Fibonacci sequence**, based directly on the mathematical definition:

```
function fib(n) if n <=1 return n return fib(n - 1) + fib(n - 2)
```

Notice that if we call, say, $\text{fib}(5)$, we produce a call tree that calls the function on the same value many different times:

1. $\text{fib}(5)$
2. $\text{fib}(4) + \text{fib}(3)$
3. $(\text{fib}(3) + \text{fib}(2)) + (\text{fib}(2) + \text{fib}(1))$
4. $((\text{fib}(2) + \text{fib}(1)) + (\text{fib}(1) + \text{fib}(0))) + ((\text{fib}(1) + \text{fib}(0)) + \text{fib}(1))$
5. $((\text{fib}(1) + \text{fib}(0)) + \text{fib}(1)) + (\text{fib}(1) + \text{fib}(0)) + ((\text{fib}(1) + \text{fib}(0)) + \text{fib}(1))$

In particular, $\text{fib}(2)$ was calculated three times from scratch. In larger examples, many more values of fib , or *subproblems*, are recalculated, leading to an exponential time algorithm.

Now, suppose we have a simple **map** object, m , which maps each value of fib that has already been calculated to its result, and we modify our function to use it and update it. The resulting function requires only $O(n)$ time instead of exponential time (but requires $O(n)$ space):

```
var m := map(0 → 0, 1 → 1) function fib(n) if key n is not in map m[m[n]] := fib(n - 1) + fib(n - 2) return m[n]
```

This technique of saving values that have already been calculated is called **memoization**; this is the top-down approach, since we first break the problem into subproblems and then calculate and store values.

In the **bottom-up** approach, we calculate the smaller values of fib first, then build larger values from them. This method also uses $O(n)$ time since it contains a loop that repeats $n - 1$ times, but it only takes constant ($O(1)$) space, in contrast to the top-down approach which requires $O(n)$ space to store the map.

```
function fib(n) if n = 0 return 0 else var previousFib := 0, currentFib := 1 repeat n - 1 times //loop is skipped if n = 1 var newFib := previousFib + currentFib previousFib := currentFib currentFib := newFib return currentFib
```

In both examples, we only calculate $\text{fib}(2)$ one time, and then use it to calculate both $\text{fib}(4)$ and $\text{fib}(3)$, instead of computing it every time either of them is evaluated.

Note that the above method actually takes $\Omega(n^2)$ time for large n because addition of two integers with $\Omega(n)$ bits each takes $\Omega(n)$ time. (The n th fibonacci number has $\Omega(n)$ bits.) Also, there is a closed form for the Fibonacci sequence, known as **Binet's formula**, from which the n -th term can be computed in approximately $O(n(\log n)^2)$ time, which is more efficient than the above dynamic programming technique. However, the simple recurrence directly gives the matrix form that leads to an approximately $O(n \log n)$ algorithm by fast matrix exponentiation.

33.3.3 A type of balanced 0–1 matrix

Consider the problem of assigning values, either zero or one, to the positions of an $n \times n$ matrix, with n even, so that each row and each column contains exactly $n/2$ zeros and $n/2$ ones. We ask how many different assignments there are for a given n . For example, when $n = 4$, four possible solutions are

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \text{ and } \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

There are at least three possible approaches: **brute force**, **backtracking**, and **dynamic programming**.

Brute force consists of checking all assignments of zeros and ones and counting those that have balanced rows and columns ($n / 2$ zeros and $n / 2$ ones). As there are $\binom{n}{n/2}^n$ possible assignments, this strategy is not practical except maybe up to $n = 6$.

Backtracking for this problem consists of choosing some order of the matrix elements and recursively placing ones or zeros, while checking that in every row and column the number of elements that have not been assigned plus the number of ones or zeros are both at least $n / 2$. While more sophisticated than brute force, this approach will visit every solution once, making it impractical for n larger than six, since the number of solutions is already 116,963,796,250 for $n = 10$, as we shall see.

Dynamic programming makes it possible to count the number of solutions without visiting them all. Imagine backtracking values for the first row – what information would we require about the remaining rows, in order to be able to accurately count the solutions obtained for each first row value? We consider $k \times n$ boards, where $1 \leq k \leq n$, whose k rows contain $n/2$ zeros and $n/2$ ones. The function f to which memoization is applied maps vectors of n pairs of integers to the number of admissible boards (solutions). There is one pair for each column, and its two components indicate respectively the number of zeros and ones that have yet to be placed in that column. We seek the value of $f((n/2, n/2), (n/2, n/2), \dots, (n/2, n/2))$ (n arguments or one vector of n elements). The process of subproblem creation involves iterating over every one of $\binom{n}{n/2}$ possible assignments for the top row of the board, and going through every column, subtracting one from the appropriate element of the pair for that column, depending on whether the assignment for the top row contained a zero or a one at that position. If any one of the results is negative, then the assignment is invalid and does not contribute to the set of solutions (recursion stops). Otherwise, we have an assignment for the top row of the $k \times n$ board and recursively compute the number of solutions to the remaining $(k - 1) \times n$ board, adding the numbers of solutions for every admissible assignment of the top row and returning the sum, which is being memoized. The base case is the trivial subproblem, which occurs for a $1 \times n$ board. The number of solutions for this board is either zero or one, depending on whether the vector is a permutation of $n/2$ $(0, 1)$ and $n/2$ $(1, 0)$ pairs or not.

For example, in the first two boards shown above the sequences of vectors would be

$$\begin{aligned} ((2, 2) (2, 2) (2, 2) (2, 2)) ((2, 2) (2, 2) (2, 2) (2, 2)) k = \\ 4 0 1 0 1 0 0 1 1 ((1, 2) (2, 1) (1, 2) (2, 1)) ((1, 2) (1, 2) \\ (2, 1) (2, 1)) k = 3 1 0 1 0 0 0 1 1 ((1, 1) (1, 1) (1, 1) (1, \\ 1)) ((0, 2) (0, 2) (2, 0) (2, 0)) k = 2 0 1 0 1 1 1 0 0 ((0, \\ 1) (1, 0) (0, 1) (1, 0)) ((0, 1) (0, 1) (1, 0) (1, 0)) k = 1 1 \\ 0 1 0 1 1 0 0 ((0, 0) (0, 0) (0, 0) (0, 0)) ((0, 0) (0, 0), (0, \\ 0)) \end{aligned}$$

$0) (0, 0))$

The number of solutions (sequence [A058527](#) in OEIS) is

1, 2, 90, 297200, 116963796250, 6736218287430460752, ...

Links to the MAPLE implementation of the dynamic programming approach may be found among the [external links](#).

33.3.4 Checkerboard

Consider a **checkerboard** with $n \times n$ squares and a cost-function $c(i, j)$ which returns a cost associated with square i, j (i being the row, j being the column). For instance (on a 5×5 checkerboard),

Thus $c(1, 3) = 5$

Let us say you had a checker that could start at any square on the first rank (i.e., row) and you wanted to know the shortest path (sum of the costs of the visited squares are at a minimum) to get to the last rank, assuming the checker could move only diagonally left forward, diagonally right forward, or straight forward. That is, a checker on $(1, 3)$ can move to $(2, 2)$, $(2, 3)$ or $(2, 4)$.

This problem exhibits **optimal substructure**. That is, the solution to the entire problem relies on solutions to subproblems. Let us define a function $q(i, j)$ as

$$q(i, j) = \text{the minimum cost to reach square } (i, j).$$

If we can find the values of this function for all the squares at rank n , we pick the minimum and follow that path backwards to get the shortest path.

Note that $q(i, j)$ is equal to the minimum cost to get to any of the three squares below it (since those are the only squares that can reach it) plus $c(i, j)$. For instance:

$$q(A) = \min(q(B), q(C), q(D)) + c(A)$$

Now, let us define $q(i, j)$ in somewhat more general terms:

$$q(i, j) = \begin{cases} \infty & \\ c(i, j) & \\ \min(q(i - 1, j - 1), q(i - 1, j), q(i - 1, j + 1)) + c(i, j) & \end{cases}$$

The first line of this equation is there to make the recursive property simpler (when dealing with the edges, so we need only one recursion). The second line says what happens in the last rank, to provide a base case. The third line, the recursion, is the important part. It is similar to the A,B,C,D example. From this definition we can make

a straightforward recursive code for $q(i, j)$. In the following pseudocode, n is the size of the board, $c(i, j)$ is the cost-function, and $\min()$ returns the minimum of a number of values:

```
function minCost( $i, j$ ) if  $j < 1$  or  $j > n$  return infinity
else if  $i = 1$  return  $c(i, j)$  else return  $\min(\minCost(i-1,$ 
 $j-1), \minCost(i-1, j), \minCost(i-1, j+1)) + c(i, j)$ 
```

It should be noted that this function only computes the path-cost, not the actual path. We will get to the path soon. This, like the Fibonacci-numbers example, is horribly slow since it wastes time recomputing the same shortest paths over and over. However, we can compute it much faster in a bottom-up fashion if we store path-costs in a two-dimensional array $q[i, j]$ rather than using a function. This avoids recompilation; before computing the cost of a path, we check the array $q[i, j]$ to see if the path cost is already there.

We also need to know what the actual shortest path is. To do this, we use another array $p[i, j]$, a *predecessor array*. This array implicitly stores the path to any square s by storing the previous node on the shortest path to s , i.e. the predecessor. To reconstruct the path, we lookup the predecessor of s , then the predecessor of that square, then the predecessor of that square, and so on, until we reach the starting square. Consider the following code:

```
function computeShortestPathArrays() for  $x$  from 1 to
 $n$   $q[1, x] := c(1, x)$  for  $y$  from 1 to  $n$   $q[y, 0] :=$  infinity
 $q[y, n+1] :=$  infinity for  $y$  from 2 to  $n$  for  $x$  from 1 to
 $n$   $m := \min(q[y-1, x-1], q[y-1, x], q[y-1, x+1])$   $q[y, x] :=$ 
 $m + c(y, x)$  if  $m = q[y-1, x-1]$   $p[y, x] := -1$  else if  $m =$ 
 $q[y-1, x]$   $p[y, x] := 0$  else  $p[y, x] := 1$ 
```

Now the rest is a simple matter of finding the minimum and printing it.

```
function computeShortestPath() computeShortest-
PathArrays()  $\minIndex := 1$   $\min := q[n, 1]$  for  $i$  from 2
to  $n$  if  $q[n, i] < \min$   $\minIndex := i$   $\min := q[n, i]$  print-
Path( $n, \minIndex$ ) function printPath( $y, x$ ) print(x)
print("->") if  $y = 2$  print( $x + p[y, x]$ ) else printPath( $y-1,$ 
 $x + p[y, x]$ )
```

33.3.5 Sequence alignment

In genetics, sequence alignment is an important application where dynamic programming is essential.^[11] Typically, the problem consists of transforming one sequence into another using edit operations that replace, insert, or remove an element. Each operation has an associated cost, and the goal is to find the sequence of edits with the lowest total cost.

The problem can be stated naturally as a recursion, a sequence A is optimally edited into a sequence B by either:

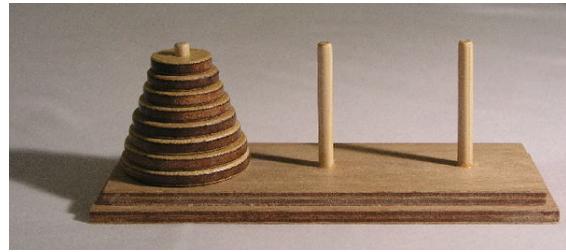
1. inserting the first character of B, and performing an optimal alignment of A and the tail of B

2. deleting the first character of A, and performing the optimal alignment of the tail of A and B
3. replacing the first character of A with the first character of B, and performing optimal alignments of the tails of A and B.

The partial alignments can be tabulated in a matrix, where cell (i,j) contains the cost of the optimal alignment of $A[1..i]$ to $B[1..j]$. The cost in cell (i,j) can be calculated by adding the cost of the relevant operations to the cost of its neighboring cells, and selecting the optimum.

Different variants exist, see Smith–Waterman algorithm and Needleman–Wunsch algorithm.

33.3.6 Tower of Hanoi



A model set of the Towers of Hanoi (with 8 disks)



An animated solution of the Tower of Hanoi puzzle for T(4,3).

The **Tower of Hanoi** or **Towers of Hanoi** is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:

- Only one disk may be moved at a time.
- Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
- No disk may be placed on top of a smaller disk.

The dynamic programming solution consists of solving the functional equation

$$S(n,h,t) = S(n-1,h, \text{not}(h,t)) ; S(1,h,t) ; S(n-1,\text{not}(h,t),t)$$

where n denotes the number of disks to be moved, h denotes the home rod, t denotes the target rod, $\text{not}(h,t)$ denotes the third rod (neither h nor t), ";" denotes concatenation, and

$S(n, h, t)$:= solution to a problem consisting of n disks that are to be moved from rod h to rod t .

Note that for $n=1$ the problem is trivial, namely $S(1,h,t)$ = "move a disk from rod h to rod t " (there is only one disk left).

The number of moves required by this solution is $2^n - 1$. If the objective is to **maximize** the number of moves (without cycling) then the dynamic programming functional equation is slightly more complicated and $3^n - 1$ moves are required.^[12]

33.3.7 Egg dropping puzzle

The following is a description of the instance of this famous **puzzle** involving $n=2$ eggs and a building with $H=36$ floors:^[13]

Suppose that we wish to know which stories in a 36-story building are safe to drop eggs from, and which will cause the eggs to break on landing (using U.S. English terminology, in which the first floor is at ground level). We make a few assumptions:

- An egg that survives a fall can be used again.
- A broken egg must be discarded.
- The effect of a fall is the same for all eggs.
- If an egg breaks when dropped, then it would break if dropped from a higher window.
- If an egg survives a fall, then it would survive a shorter fall.
- It is not ruled out that the first-floor windows break eggs, nor is it ruled out that eggs can survive the 36th-floor windows.

If only one egg is available and we wish to be sure of obtaining the right result, the experiment can be carried out in only one way. Drop the egg from the first-floor window; if it survives, drop it from the second-floor window. Continue upward until it breaks. In the worst case, this method may require 36 droppings. Suppose 2 eggs are available. What is the least number of egg-droppings that is guaranteed to work in all cases?

To derive a dynamic programming **functional equation** for this puzzle, let the **state** of the dynamic programming model be a pair $s = (n,k)$, where

n = number of test eggs available, $n = 0, 1, 2, 3, \dots, N - 1$.

k = number of (consecutive) floors yet to be tested, $k = 0, 1, 2, \dots, H - 1$.

For instance, $s = (2,6)$ indicates that two test eggs are available and 6 (consecutive) floors are yet to be tested. The initial state of the process is $s = (N,H)$ where N denotes the number of test eggs available at the commencement of the experiment. The process terminates either when there are no more test eggs ($n = 0$) or when $k = 0$, whichever occurs first. If termination occurs at state $s = (0,k)$ and $k > 0$, then the test failed.

Now, let

$W(n,k)$ = minimum number of trials required to identify the value of the critical floor under the worst-case scenario given that the process is in state $s = (n,k)$.

Then it can be shown that^[14]

$$W(n,k) = 1 + \min\{\max(W(n-1, x-1), W(n,k-x)): x = 1, 2, \dots, k\}$$

with $W(n,1) = 1$ for all $n > 0$ and $W(1,k) = k$ for all k . It is easy to solve this equation iteratively by systematically increasing the values of n and k .

An interactive online facility is available for experimentation with this model as well as with other versions of this puzzle (e.g. when the objective is to minimize the **expected value** of the number of trials.)^[14]

Faster DP solution using a different parametrization

Notice that the above solution takes $O(nk^2)$ time with a DP solution. This can be improved to $O(nk \log k)$ time by binary searching on the optimal x in the above recurrence, since $W(n-1, x-1)$ is increasing in x while $W(n, k-x)$ is decreasing in x , thus a local minimum of $\max(W(n-1, x-1), W(n, k-x))$ is a global minimum. Also, by storing the optimal x for each cell in the DP table and referring to its value for the previous cell, the optimal x for each cell can be found in constant time, improving it to $O(nk)$ time. However, there is an even faster solution that involves a different parametrization of the problem:

Let k be the total number of floors such that the eggs break when dropped from the k th floor (The example above is equivalent to taking $k = 37$).

Let m be the minimum floor from which the egg must be dropped to be broken.

Let $f(t, n)$ be the maximum number of values of m that are distinguishable using t tries and n eggs.

Then $f(t, 0) = f(0, n) = 1$ for all $t, n \geq 0$.

Let a be the floor from which the first egg is dropped in the optimal strategy.

If the first egg broke, m is from 1 to a and distinguishable using at most $t - 1$ tries and $n - 1$ eggs.

If the first egg did not break, m is from $a + 1$ to k and distinguishable using $t - 1$ tries and n eggs.

Therefore $f(t, n) = f(t - 1, n - 1) + f(t - 1, n)$.

Then the problem is equivalent to finding the minimum x such that $f(x, n) \geq k$.

To do so, we could compute $\{f(t, i) : 0 \leq i \leq n\}$ in order of increasing t , which would take $O(nx)$ time.

Thus, if we separately handle the case of $n = 1$, the algorithm would take $O(n\sqrt{k})$ time.

But the recurrence relation can in fact be solved, giving $f(t, n) = \sum_{i=0}^n \binom{t}{i}$, which can be computed in $O(n)$ time using the identity $\binom{t}{i+1} = \binom{t}{i} \frac{t-i}{i+1}$ for all $i \geq 0$.

Since $f(t, n) \leq f(t + 1, n)$ for all $t \geq 0$, we can binary search on t to find x , giving an $O(n \log k)$ algorithm.^[15]

33.3.8 Matrix chain multiplication

Main article: Matrix chain multiplication

Matrix chain multiplication is a well known example that demonstrates utility of dynamic programming. For example, engineering applications often have to multiply a chain of matrices. It is not surprising to find matrices of large dimensions, for example 100×100 . Therefore, our task is to multiply matrices A_1, A_2, \dots, A_n . As we know from basic linear algebra, matrix multiplication is not commutative, but is associative; and we can multiply only two matrices at a time. So, we can multiply this chain of matrices in many different ways, for example:

$$((A_1 \times A_2) \times A_3) \times \dots \times A_n$$

$$A_1 \times ((A_2 \times A_3) \times \dots) \times A_n$$

$$(A_1 \times A_2) \times (A_3 \times \dots) \times A_n$$

and so on. There are numerous ways to multiply this chain of matrices. They will all produce the same final result, however they will take more or less time to compute, based on which particular matrices are multiplied. If matrix A has dimensions $m \times n$ and matrix B has dimensions $n \times q$, then matrix $C = A \times B$ will have dimensions

$m \times q$, and will require $m * n * q$ scalar multiplications (using a simplistic matrix multiplication algorithm for purposes of illustration).

For example, let us multiply matrices A , B and C . Let us assume that their dimensions are $m \times n$, $n \times p$, and $p \times s$, respectively. Matrix $A \times B \times C$ will be of size $m \times s$ and can be calculated in two ways shown below:

1. $A \times (B \times C)$ This order of matrix multiplication will require $nps + mns$ scalar multiplications.
2. $(A \times B) \times C$ This order of matrix multiplication will require $mnp + mps$ scalar calculations.

Let us assume that $m = 10$, $n = 100$, $p = 10$ and $s = 1000$. So, the first way to multiply the chain will require $1,000,000 + 1,000,000$ calculations. The second way will require only $10,000 + 100,000$ calculations. Obviously, the second way is faster, and we should multiply the matrices using that arrangement of parenthesis.

Therefore, our conclusion is that the order of parenthesis matters, and that our task is to find the optimal order of parenthesis.

At this point, we have several choices, one of which is to design a dynamic programming algorithm that will split the problem into overlapping problems and calculate the optimal arrangement of parenthesis. The dynamic programming solution is presented below.

Let's call $m[i, j]$ the minimum number of scalar multiplications needed to multiply a chain of matrices from matrix i to matrix j (i.e. $A_i \times \dots \times A_j$, i.e. $i \leq j$). We split the chain at some matrix k , such that $i \leq k < j$, and try to find out which combination produces minimum $m[i, j]$.

The formula is:

if $i = j$, $m[i, j] = 0$ **if** $i < j$, $m[i, j] = \min$ over all possible values of k ()

where k is changed from i to $j - 1$.

- is the row dimension of matrix i ,
- is the column dimension of matrix k ,
- is the column dimension of matrix j .

This formula can be coded as shown below, where input parameter "chain" is the chain of matrices, i.e. :

```
function OptimalMatrixChainParenthesis(chain) n = length(chain)
for i = 1, n m[i,i] = 0 //since it takes no calculations to multiply one matrix
for len = 2, n for i = 1, n - len + 1 for j = i, len - 1 m[i,j] = infinity //so that the first calculation updates
for k = i, j-1 if q < m[i, j] //the new order of parenthesis is better than what we had
m[i, j] = q //update s[i, j] = k //record which k to split on, i.e. where to place the parenthesis
```

So far, we have calculated values for all possible $m[i, j]$, the minimum number of calculations to multiply a chain

from matrix i to matrix j , and we have recorded the corresponding “split point” $s[i, j]$. For example, if we are multiplying chain $A_1 \times A_2 \times A_3 \times A_4$, and it turns out that $m[1, 3] = 100$ and $s[1, 3] = 2$, that means that the optimal placement of parenthesis for matrices 1 to 3 is $(A_1 \times A_2) \times A_3$ and to multiply those matrices will require 100 scalar calculation.

This algorithm will produce “tables” $m[,]$ and $s[,]$ that will have entries for all possible values of i and j . The final solution for the entire chain is $m[1, n]$, with corresponding split at $s[1, n]$. Unraveling the solution will be recursive, starting from the top and continuing until we reach the base case, i.e. multiplication of single matrices.

Therefore, the next step is to actually split the chain, i.e. to place the parenthesis where they (optimally) belong. For this purpose we could use the following algorithm:

```
function PrintOptimalParenthesis(s, i, j) if i = j print "A"i else print "(" PrintOptimalParenthesis(s, i, s[i, j]) PrintOptimalParenthesis(s, s[i, j] + 1, j) ")"
```

Of course, this algorithm is not useful for actual multiplication. This algorithm is just a user-friendly way to see what the result looks like.

To actually multiply the matrices using the proper splits, we need the following algorithm:

```
function MatrixChainMultiply(chain from 1 to n) // returns the final matrix, i.e.  $A_1 \times A_2 \times \dots \times A_n$ 
OptimalMatrixChainParenthesis(chain from 1 to n) // this will produce s[ . ] and m[ . ] “tables”
OptimalMatrixMultiplication(s, chain from 1 to n) // actually multiply
function OptimalMatrixMultiplication(s, i, j) // returns the result of multiplying a chain of matrices from  $A_i$  to  $A_j$  in optimal way
if i < j
// keep on splitting the chain and multiplying the matrices in left and right sides
LeftSide = OptimalMatrixMultiplication(s, i, s[i, j])
RightSide = OptimalMatrixMultiplication(s, s[i, j] + 1, j)
return MatrixMultiply(LeftSide, RightSide)
else if i = j
return  $A_i$  // matrix at position i
else
print “error,  $i \leq j$  must hold”
function MatrixMultiply(A, B) // function that multiplies two matrices
if columns(A) = rows(B)
for i = 1, rows(A)
for j = 1, columns(B)
C[i, j] = 0
for k = 1, columns(A)
C[i, j] = C[i, j] + A[i, k] * B[k, j]
return C
else
print “error, incompatible dimensions.”
```

33.4 History

The term *dynamic programming* was originally used in the 1940s by **Richard Bellman** to describe the process of solving problems where one needs to find the best decisions one after another. By 1953, he refined this to the modern meaning, referring specifically to nesting smaller decision problems inside larger decisions,^[16] and the field was thereafter recognized by the **IEEE** as a **systems analysis** and **engineering** topic. Bellman’s contribution is remembered in the name of the **Bellman equation**, a central result of dynamic programming which restates an optimization problem in recursive form.

Bellman explains the reasoning behind the term *dynamic programming* in his autobiography, *Eye of the Hurricane: An Autobiography* (1984). He explains:

“I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, Where did the name, dynamic programming, come from? The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named **Wilson**. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word research. I’m not using the term lightly; I’m using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term research in his presence. You can imagine how he felt, then, about the term mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word “programming”. I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying I thought, lets kill two birds with one stone. Lets take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is it’s impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It’s impossible. Thus, I thought dynamic programming was a good name. It was some-

thing not even a Congressman could object to.
So I used it as an umbrella for my activities.”

The word *dynamic* was chosen by Bellman to capture the time-varying aspect of the problems, and because it sounded impressive.^[11] The word *programming* referred to the use of the method to find an optimal *program*, in the sense of a military schedule for training or logistics. This usage is the same as that in the phrases *linear programming* and *mathematical programming*, a synonym for mathematical optimization.^[17]

33.5 Algorithms that use dynamic programming

- Recurrent solutions to lattice models for protein-DNA binding
- Backward induction as a solution method for finite-horizon discrete-time dynamic optimization problems
- Method of undetermined coefficients can be used to solve the Bellman equation in infinite-horizon, discrete-time, discounted, time-invariant dynamic optimization problems
- Many string algorithms including longest common subsequence, longest increasing subsequence, longest common substring, Levenshtein distance (edit distance)
- Many algorithmic problems on graphs can be solved efficiently for graphs of bounded treewidth or bounded clique-width by using dynamic programming on a tree decomposition of the graph.
- The Cocke–Younger–Kasami (CYK) algorithm which determines whether and how a given string can be generated by a given context-free grammar
- Knuth’s word wrapping algorithm that minimizes raggedness when word wrapping text
- The use of transposition tables and refutation tables in computer chess
- The Viterbi algorithm (used for hidden Markov models)
- The Earley algorithm (a type of chart parser)
- The Needleman–Wunsch and other algorithms used in bioinformatics, including sequence alignment, structural alignment, RNA structure prediction
- Floyd’s all-pairs shortest path algorithm
- Optimizing the order for chain matrix multiplication
- Pseudo-polynomial time algorithms for the subset sum and knapsack and partition problems
- The dynamic time warping algorithm for computing the global distance between two time series
- The Selinger (a.k.a. System R) algorithm for relational database query optimization
- De Boor algorithm for evaluating B-spline curves
- Duckworth–Lewis method for resolving the problem when games of cricket are interrupted
- The value iteration method for solving Markov decision processes
- Some graphic image edge following selection methods such as the “magnet” selection tool in Photoshop
- Some methods for solving interval scheduling problems
- Some methods for solving word wrap problems
- Some methods for solving the travelling salesman problem, either exactly (in exponential time) or approximately (e.g. via the bitonic tour)
- Recursive least squares method
- Beat tracking in music information retrieval
- Adaptive-critic training strategy for artificial neural networks
- Stereo algorithms for solving the correspondence problem used in stereo vision
- Seam carving (content aware image resizing)
- The Bellman–Ford algorithm for finding the shortest distance in a graph
- Some approximate solution methods for the linear search problem
- Kadane’s algorithm for the maximum subarray problem

33.6 See also

- Convexity in economics
- Greedy algorithm
- Non-convexity (economics)
- Stochastic programming

33.7 References

- [1] S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani, '**Algorithms**', p 173, available at <http://www.cs.berkeley.edu/~vazirani/algorithms.html>
- [2] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. (2001), Introduction to Algorithms (2nd ed.), MIT Press & McGraw-Hill, ISBN 0-262-03293-7 . pp. 327–8.
- [3] DeLisi, Biopolymers, 1974, Volume 13, Issue 7, pages 1511–1512, July 1974
- [4] Gurskiĭ GV, Zasedatelev AS, Biofizika, 1978 Sep-Oct;23(5):932-46
- [5] "M. Memo". J Vocabulary. J Software. Retrieved 28 October 2011.
- [6] Stokey et al., 1989, Chap. 1
- [7] Sniedovich, M. (2006), "Dijkstra's algorithm revisited: the dynamic programming connexion" (PDF), *Journal of Control and Cybernetics* 35 (3): 599–620. Online version of the paper with interactive computational modules.
- [8] Denardo, E.V. (2003), *Dynamic Programming: Models and Applications*, Mineola, NY: Dover Publications, ISBN 978-0-486-42810-9
- [9] Sniedovich, M. (2010), *Dynamic Programming: Foundations and Principles*, Taylor & Francis, ISBN 978-0-8247-4099-3
- [10] Dijkstra 1959, p. 270
- [11] Eddy, S. R., What is dynamic programming?, *Nature Biotechnology*, 22, 909–910 (2004).
- [12] Moshe Sniedovich (2002), "OR/MS Games: 2. The Towers of Hanoi Problem.", *INFORMS Transactions on Education* 3 (1): 34–51.
- [13] Konhauser J.D.E., Velleman, D., and Wagon, S. (1996). Which way did the Bicycle Go? Dolciani Mathematical Expositions – No 18. The Mathematical Association of America.
- [14] Sniedovich, M. (2003). The joy of egg-dropping in Braunschweig and Hong Kong. *INFORMS Transactions on Education*, 4(1) 48–64.
- [15] Dean Connable Wills, *Connections between combinatorics of permutations and algorithms and geometry*
- [16] http://www.wu-wien.ac.at/usr/h99c/h9951826/bellman_dynprog.pdf
- [17] Nocedal, J.; Wright, S. J.: Numerical Optimization, page 9, Springer, 2006..

33.8 Further reading

- Adda, Jerome; Cooper, Russell (2003), *Dynamic Economics*, MIT Press. An accessible introduction to dynamic programming in economics. The link contains sample programs.
- Bellman, Richard (1954), "The theory of dynamic programming", *Bulletin of the American Mathematical Society* 60 (6): 503–516, doi:10.1090/S0002-9904-1954-09848-8, MR 0067459. Includes an extensive bibliography of the literature in the area, up to the year 1954.
- Bellman, Richard (1957), *Dynamic Programming*, Princeton University Press. Dover paperback edition (2003), ISBN 0-486-42809-5.
- Bertsekas, D. P. (2000), *Dynamic Programming and Optimal Control* (2nd ed.), Athena Scientific, ISBN 1-886529-09-4. In two volumes.
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001), *Introduction to Algorithms* (2nd ed.), MIT Press & McGraw-Hill, ISBN 0-262-03293-7. Especially pp. 323–69.
- Dreyfus, Stuart E.; Law, Averill M. (1977), *The Art and Theory of Dynamic Programming*, Academic Press, ISBN 978-0-12-221860-6.
- Giegerich, R.; Meyer, C.; Steffen, P. (2004), "A Discipline of Dynamic Programming over Sequence Data", *Science of Computer Programming* 51 (3): 215–263, doi:10.1016/j.scico.2003.12.005.
- Meyn, Sean (2007), *Control Techniques for Complex Networks*, Cambridge University Press, ISBN 978-0-521-88441-9.
- S. S. Sritharan (1991), "Dynamic Programming of the Navier-Stokes Equations", *Systems and Control Letters*, 16(4), 299–307.
- Stokey, Nancy; Lucas, Robert E.; Prescott, Edward (1989), *Recursive Methods in Economic Dynamics*, Harvard Univ. Press, ISBN 978-0-674-75096-8.

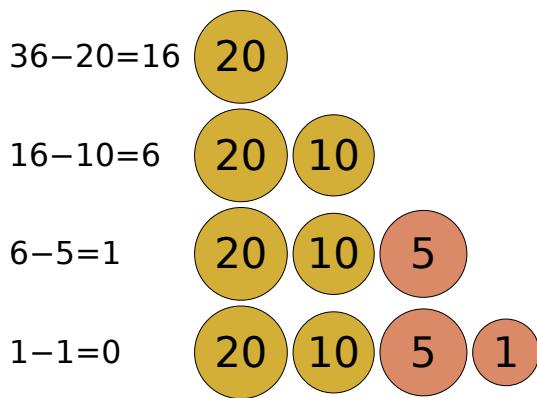
33.9 External links

- An Introduction to Dynamic Programming
- Dyna, a declarative programming language for dynamic programming algorithms
- Wagner, David B., 1995, "Dynamic Programming." An introductory article on dynamic programming in Mathematica.
- Ohio State University: CIS 680: class notes on dynamic programming, by Eitan M. Gurari

- A Tutorial on Dynamic programming
- MIT course on algorithms – Includes a video lecture on DP along with lecture notes
- More DP Notes
- King, Ian, 2002 (1987), "A Simple Introduction to Dynamic Programming in Macroeconomic Models." An introduction to dynamic programming as an important tool in economic theory.
- Dynamic Programming: from novice to advanced A TopCoder.com article by Dumitru on Dynamic Programming
- Algebraic Dynamic Programming – a formalized framework for dynamic programming, including an entry-level course to DP, University of Bielefeld
- Dreyfus, Stuart, "Richard Bellman on the birth of Dynamic Programming."
- Dynamic programming tutorial
- A Gentle Introduction to Dynamic Programming and the Viterbi Algorithm
- Tabled Prolog BProlog and XSB
- Online interactive dynamic programming modules including, shortest path, traveling salesman, knapsack, false coin, egg dropping, bridge and torch, replacement, chained matrix products, and critical path problem.

Chapter 34

Greedy algorithm



Greedy algorithms determine minimum number of coins to give while making change. These are the steps a human would take to emulate a greedy algorithm to represent 36 cents using only coins with values {1, 5, 10, 20}. The coin of the highest value, less than the remaining change owed, is the local optimum. (Note that in general the change-making problem requires dynamic programming or integer programming to find an optimal solution; however, most currency systems, including the Euro and US Dollar, are special cases where the greedy strategy does find an optimal solution.)

A **greedy algorithm** is an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage^[1] with the hope of finding a global optimum. In many problems, a greedy strategy does not in general produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a global optimal solution in a reasonable time.

For example, a greedy strategy for the traveling salesman problem (which is of a high computational complexity) is the following heuristic: “At each stage visit an unvisited city nearest to the current city”. This heuristic need not find a best solution, but terminates in a reasonable number of steps; finding an optimal solution typically requires unreasonably many steps. In mathematical optimization, greedy algorithms solve combinatorial problems having the properties of matroids.

34.1 Specifics

In general, greedy algorithms have five components:

1. A candidate set, from which a solution is created
2. A selection function, which chooses the best candidate to be added to the solution
3. A feasibility function, that is used to determine if a candidate can be used to contribute to a solution
4. An objective function, which assigns a value to a solution, or a partial solution, and
5. A solution function, which will indicate when we have discovered a complete solution

Greedy algorithms produce good solutions on some mathematical problems, but not on others. Most problems for which they work, will have two properties:

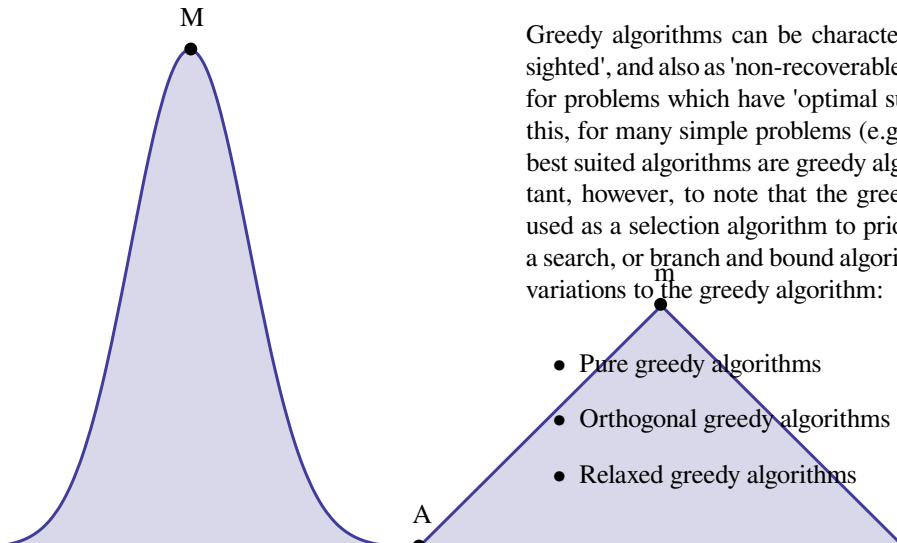
Greedy choice property We can make whatever choice seems best at the moment and then solve the subproblems that arise later. The choice made by a greedy algorithm may depend on choices made so far but not on future choices or all the solutions to the subproblem. It iteratively makes one greedy choice after another, reducing each given problem into a smaller one. In other words, a greedy algorithm never reconsiders its choices. This is the main difference from dynamic programming, which is exhaustive and is guaranteed to find the solution. After every stage, dynamic programming makes decisions based on all the decisions made in the previous stage, and may reconsider the previous stage’s algorithmic path to solution.

Optimal substructure “A problem exhibits optimal substructure if an optimal solution to the problem contains optimal solutions to the sub-problems.”^[2]

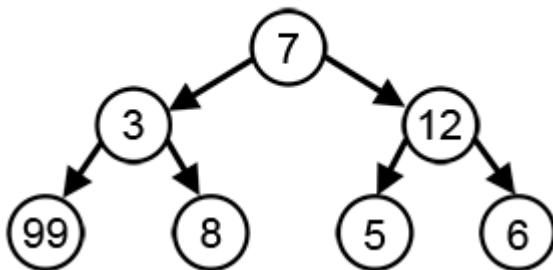
34.1.1 Cases of failure

Examples on how a greedy algorithm may fail to achieve the optimal solution.

34.2 Types



Starting at A, a greedy algorithm will find the local maximum at "m", oblivious of the global maximum at "M".



With a goal of reaching the largest-sum, at each step, the greedy algorithm will choose what appears to be the optimal immediate choice, so it will choose 12 instead of 3 at the second step, and will not reach the best solution, which contains 99.

For many other problems, greedy algorithms fail to produce the optimal solution, and may even produce the *unique worst possible* solution. One example is the traveling salesman problem mentioned above: for each number of cities, there is an assignment of distances between the cities for which the nearest neighbor heuristic produces the unique worst possible tour.^[3]

Imagine the coin example with only 25-cent, 10-cent, and 4-cent coins. The greedy algorithm would not be able to make change for 41 cents, since after committing to use one 25-cent coin and one 10-cent coin it would be impossible to use 4-cent coins for the balance of 6 cents, whereas a person or a more sophisticated algorithm could make change for 41 cents with one 25-cent coin and four 4-cent coins.

Greedy algorithms can be characterized as being 'short sighted', and also as 'non-recoverable'. They are ideal only for problems which have 'optimal substructure'. Despite this, for many simple problems (e.g. giving change), the best suited algorithms are greedy algorithms. It is important, however, to note that the greedy algorithm can be used as a selection algorithm to prioritize options within a search, or branch and bound algorithm. There are a few variations to the greedy algorithm:

- Pure greedy algorithms
- Orthogonal greedy algorithms
- Relaxed greedy algorithms

34.3 Applications

Greedy algorithms mostly (but not always) fail to find the globally optimal solution, because they usually do not operate exhaustively on all the data. They can make commitments to certain choices too early which prevent them from finding the best overall solution later. For example, all known greedy coloring algorithms for the [graph coloring problem](#) and all other [NP-complete](#) problems do not consistently find optimum solutions. Nevertheless, they are useful because they are quick to think up and often give good approximations to the optimum.

If a greedy algorithm can be proven to yield the global optimum for a given problem class, it typically becomes the method of choice because it is faster than other optimization methods like [dynamic programming](#). Examples of such greedy algorithms are [Kruskal's algorithm](#) and [Prim's algorithm](#) for finding [minimum spanning trees](#), and the algorithm for finding optimum [Huffman trees](#).

The theory of [matroids](#), and the more general theory of [greedoids](#), provide whole classes of such algorithms.

Greedy algorithms appear in network routing as well. Using greedy routing, a message is forwarded to the neighboring node which is "closest" to the destination. The notion of a node's location (and hence "closeness") may be determined by its physical location, as in [geographic routing](#) used by [ad hoc networks](#). Location may also be an entirely artificial construct as in [small world routing](#) and [distributed hash table](#).

34.4 Examples

- The [activity selection problem](#) is characteristic to this class of problems, where the goal is to pick the maximum number of activities that do not clash with each other.

- In the Macintosh computer game *Crystal Quest* the objective is to collect crystals, in a fashion similar to the [travelling salesman problem](#). The game has a demo mode, where the game uses a greedy algorithm to go to every crystal. The [artificial intelligence](#) does not account for obstacles, so the demo mode often ends quickly.
- The [matching pursuit](#) is an example of greedy algorithm applied on signal approximation.
- A greedy algorithm finds the optimal solution to [Malfatti's problem](#) of finding three disjoint circles within a given triangle that maximize the total area of the circles; it is conjectured that the same greedy algorithm is optimal for any number of circles.
- A greedy algorithm is used to construct a Huffman tree during [Huffman coding](#) where it finds an optimal solution.
- In [decision tree learning](#) greedy algorithms are commonly used however they are not guaranteed to find the optimal solution.
- G. Gutin, A. Yeo and A. Zverovich, Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP. *Discrete Applied Mathematics* 117 (2002), 81–86.
- J. Bang-Jensen, G. Gutin and A. Yeo, When the greedy algorithm fails. *Discrete Optimization* 1 (2004), 121–127.
- G. Bendall and F. Margot, Greedy Type Resistance of Combinatorial Problems, *Discrete Optimization* 3 (2006), 288–298.

34.8 External links

- Hazewinkel, Michiel, ed. (2001), “Greedy algorithm”, *Encyclopedia of Mathematics*, Springer, ISBN 978-1-55608-010-4
- [Greedy algorithm visualization](#) A visualization of a greedy solution to the N-Queens puzzle by Yuval Baror.
- [Python greedy coin example](#) by Noah Gift.

34.5 See also

- Epsilon-greedy strategy
- Greedy algorithm for Egyptian fractions
- Greedy source
- Matroid

34.6 Notes

- [1] Black, Paul E. (2 February 2005). “greedy algorithm”. *Dictionary of Algorithms and Data Structures*. U.S. National Institute of Standards and Technology (NIST). Retrieved 17 August 2012.
- [2] Introduction to Algorithms (Cormen, Leiserson, Rivest, and Stein) 2001, Chapter 16 “Greedy Algorithms”.
- [3] (G. Gutin, A. Yeo and A. Zverovich, 2002)

34.7 References

- *Introduction to Algorithms* (Cormen, Leiserson, and Rivest) 1990, Chapter 17 “Greedy Algorithms” p. 329.
- *Introduction to Algorithms* (Cormen, Leiserson, Rivest, and Stein) 2001, Chapter 16 “Greedy Algorithms”.

Chapter 35

Integer programming

An **integer programming** problem is a mathematical optimization or feasibility program in which some or all of the variables are restricted to be integers. In many settings the term refers to **integer linear programming** (ILP), in which the objective function and the constraints (other than the integer constraints) are linear.

Integer programming is NP-hard. A special case, 0-1 integer linear programming, in which unknowns are binary, is one of Karp's 21 NP-complete problems.

35.1 Canonical and standard form for ILPs

An integer linear program in canonical form is expressed as:^[1]

$$\begin{aligned} & \text{maximize } \mathbf{c}^T \mathbf{x} \\ & \text{to subject } A\mathbf{x} \leq \mathbf{b}, \\ & \quad \mathbf{x} \geq \mathbf{0}, \\ & \text{and } \mathbf{x} \in \mathbb{Z}^n, \end{aligned}$$

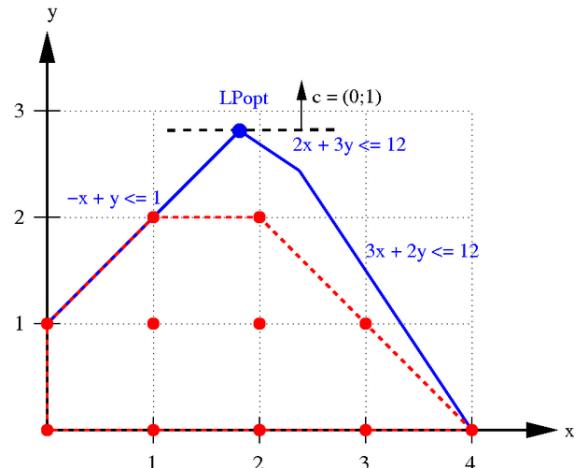
and an ILP in standard form is expressed as

$$\begin{aligned} & \text{maximize } \mathbf{c}^T \mathbf{x} \\ & \text{to subject } A\mathbf{x} + \mathbf{s} = \mathbf{b}, \\ & \quad \mathbf{s} \geq \mathbf{0}, \\ & \text{and } \mathbf{x} \in \mathbb{Z}^n, \end{aligned}$$

where the entries of \mathbf{c}, \mathbf{b} are vectors and A is a matrix, having integer values. Note that similar to linear programs, ILPs not in standard form can be converted to standard form by eliminating inequalities by introducing slack variables (\mathbf{s}) and replacing variables that are not sign-constrained with the difference of two sign-constrained variables

35.2 Example

The graph on the right shows the following problem.



IP polytope with LP relaxation

$$\begin{aligned} & \max y \\ & -x + y \leq 1 \\ & 3x + 2y \leq 12 \\ & 2x + 3y \leq 12 \\ & x, y \geq 0 \\ & x, y \in \mathbb{Z} \end{aligned}$$

The feasible integer points are shown in red, and the red dashed lines indicate their convex hull, which is the smallest polyhedron that contains all of these points. The blue lines together with the coordinate axes define the polyhedron of the LP relaxation, which is given by the inequalities without the integrality constraint. The goal of the optimization is to move the black dotted line as far upward while still touching the polyhedron. The optimal solutions of the integer problem are the points (1, 2) and (2, 2) which both have an objective value of 2. The unique optimum of the relaxation is (1.8, 2.8) with objective value of 2.8. Note that if the solution of the relaxation is rounded to the nearest integers, it is not feasible for the ILP.

35.3 Variants

Mixed integer linear programming (MILP) involves problems in which only some of the variables, x_i , are constrained to be integers, while other variables are allowed to be non-integers.

Zero-one linear programming involves problems in which the variables are restricted to be either 0 or 1. Note that any bounded integer variable can be expressed as a combination of binary variables.^[2] For example, given an integer variable, $0 \leq x \leq U$, the variable can be expressed using $\lfloor \log_2 U \rfloor + 1$ binary variables:

$$x = x_1 + 2x_2 + 4x_3 + \dots + 2^{\lfloor \log_2 U \rfloor} x_{\lfloor \log_2 U \rfloor + 1}.$$

35.4 Example problems that can be formulated as ILPs

A large number of problems can be formulated as ILPs. These include

- Travelling Salesman
- Vertex Cover and other Covering problems
- Set packing and other Packing problems
- Boolean satisfiability

Since the decision version of integer linear programming is in NP (solutions can be verified in polynomial time) and there are NP-complete problems that can be polynomially reduced to ILPs, the decision version of integer linear programming is NP-complete.

35.5 Applications

There are two main reasons for using integer variables when modeling problems as a linear program:

1. The integer variables represent quantities that can only be integer. For example, it is not possible to build 3.7 cars.
2. The integer variables represent decisions and so should only take on the value 0 or 1.

These considerations occur frequently in practice and so integer linear programming can be used in many applications areas, some of which are briefly described below.

35.5.1 Production planning

Mixed integer programming has many applications in industrial production, including job-shop modelling. One important example happens in agricultural production planning involves determining production yield for several crops that can share resources (e.g. Land, labor, capital seeds fertilizer, etc.). A possible objective is to maximize the total production, without exceeding the available resources. In some cases, this can be expressed in terms of a linear program, but variables must be constrained to be integer.

35.5.2 Scheduling

These problems involve service and vehicle scheduling in transportation networks. For example, a problem may involve assigning buses or subways to individual routes so that a timetable can be met, and also to equip them with drivers. Here binary decision variables indicate whether a bus or subway is assigned to a route and whether a driver is assigned to a particular train or subway.

35.5.3 Telecommunications networks

The goal of these problems is to design a network of lines to install so that a predefined set of communication requirements are met and the total cost of the network is minimal.^[3] This requires optimizing both the topology of the network along with the setting the capacities of the various lines. In many cases, the capacities are constrained to be integer quantities. Usually there are, depending on the technology used, additional restrictions that can be modeled as a linear inequalities with integer or binary variables.

35.5.4 Cellular networks

The task of frequency planning in GSM mobile networks involves distributing available frequencies across the antennas so that users can be served and interference is minimized between the antennas.^[4] This problem can be formulated as an integer linear program in which binary variables indicate whether a frequency is assigned to an antenna.

35.6 Algorithms

The naive way to solve an ILP is to simply remove the constraint that \mathbf{x} is integral, solve the corresponding LP (called the LP relaxation of the ILP), and then round the entries of the solution to the LP relaxation. But, not only may this solution not be optimal, it may not even be feasible, that is it may violate some constraint.

35.6.1 Using total unimodularity

While in general the solution to LP relaxation will not be guaranteed to be optimal, if the ILP has the form $\max \mathbf{c}^T \mathbf{x}$ such that $A\mathbf{x} = \mathbf{b}$ where A , \mathbf{b} , and \mathbf{c} have all integer entries and A is totally unimodular, then every basic feasible solution is integral. Consequently, the solution returned by the simplex algorithm is guaranteed to be integral. To show that every basic feasible solution is integral, let \mathbf{x} be an arbitrary basic feasible solution. Since \mathbf{x} is feasible, we know that $A\mathbf{x} = \mathbf{b}$. Let $\mathbf{x}_0 = [x_{n_1}, x_{n_2}, \dots, x_{n_j}]$ be the elements corresponding to the basis columns for the basic solution \mathbf{x} . By definition of a basis, there is some square submatrix B of A with linearly independent columns such that $B\mathbf{x}_0 = \mathbf{b}$.

Since the columns of B are linearly independent and B is square, B is nonsingular, and therefore by assumption, B is unimodular and so $\det(B) = \pm 1$. Also, since B is nonsingular, it is invertible and therefore $\mathbf{x}_0 = B^{-1}\mathbf{b}$. By definition, $B^{-1} = \frac{B^{adj}}{\det(B)} = \pm B^{adj}$. Note that B^{adj} denotes the adjugate of B and is integral because B is integer. Therefore,

$$\begin{aligned} &\Rightarrow B^{-1} = \pm B^{adj} \text{ integral. is} \\ &\Rightarrow \mathbf{x}_0 = B^{-1}\mathbf{b} \text{ integral. is} \\ &\Rightarrow \text{integral. is solution feasible basic Every} \end{aligned}$$

Thus, if the matrix A of an ILP is totally unimodular, rather than use an ILP algorithm, the simplex method can be used to solve the LP relaxation and the solution will be integer.

35.6.2 Exact algorithms

When the matrix A is not totally unimodular, there are a variety of algorithms that can be used to solve integer linear programs exactly. One class of algorithms are cutting plane methods which work by solving the LP relaxation and then adding linear constraints that drive the solution towards being integer without excluding any integer feasible points.

Another class of algorithms are variants of the branch and bound method. For example, the branch and cut method that combines both branch and bound and cutting plane methods. Branch and bound algorithms have a number of advantages over algorithms that only use cutting planes. One advantage is that the algorithms can be terminated early and as long as at least one integral solution has been found, a feasible, although not necessarily optimal, solution can be returned. Further, the solutions of the LP relaxations can be used to provide a worst-case estimate of how far from optimality the returned solution is. Finally, branch and bound methods can be used to return multiple optimal solutions.

Lenstra in 1983 showed,^[5] that when number of variables

is fixed, integer programming problem can be solved in a polynomial time.

35.6.3 Heuristic methods

Since integer linear programming is NP-complete, many problem instances are intractable and so heuristic methods must be used instead. For example, tabu search can be used to search for solutions to ILPs.^[6] To use tabu search to solve ILPs, moves can be defined as incrementing or decrementing an integer constrained variable of a feasible solution, while keeping all other integer-constrained variables constant. The unrestricted variables are then solved for. Short term memory can consist of previous tried solutions while medium term memory can consist of values for the integer constrained variables that have resulted in high objective values (assuming the ILP is a maximization problem). Finally, long term memory can guide the search towards integer values that have not previously been tried.

Other heuristic methods that can be applied to ILPs include

- Hill climbing
- Simulated annealing
- Reactive search optimization
- Ant colony optimization
- Hopfield neural networks

There are also a variety of other problem-specific heuristics, such as the k-opt heuristic for the travelling salesman problem. Note that a disadvantage of heuristic methods is that if they fail to find a solution, it cannot be determined whether it is because there is no feasible solution or whether the algorithm simply was unable to find one. Further, it is usually impossible to quantify how close to optimal a solution returned by these methods is.

35.7 References

- [1] Papadimitriou, C.H.; Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Mineola, NY: Dover. ISBN 0486402584.
- [2] Williams, H.P. (2009). *Logic and integer programming*. International Series in Operations Research & Management Science **130**. ISBN 978-0-387-92280-5.
- [3] Borndörfer, R.; Grötschel, M. (2012). “Designing telecommunication networks by integer programming”.
- [4] Sharma, Deepak (2010). “Frequency Planning”.
- [5] H.W.Lenstra, “Integer programming with a fixed number of variables”, Mathematics of operations research, Vol 8, No 8, November 1983

- [6] Glover, F. (1989). “Tabu search-Part II”. *ORSA Journal on computing* **1** (3): 4–32. doi:10.1287/ijoc.2.1.4.

35.8 Further reading

- George L. Nemhauser; Laurence A. Wolsey (1988). *Integer and combinatorial optimization*. Wiley. ISBN 978-0-471-82819-8.
- Alexander Schrijver (1998). *Theory of linear and integer programming*. John Wiley and Sons. ISBN 978-0-471-98232-6.
- Laurence A. Wolsey (1998). *Integer programming*. Wiley. ISBN 978-0-471-28366-9.
- Dimitris Bertsimas; Robert Weismantel (2005). *Optimization over integers*. Dynamic Ideas. ISBN 978-0-9759146-2-5.
- John K. Karlof (2006). *Integer programming: theory and practice*. CRC Press. ISBN 978-0-8493-1914-3.
- H. Paul Williams (2009). *Logic and Integer Programming*. Springer. ISBN 978-0-387-92279-9.
- Michael Jünger, Thomas M. Liebling, Denis Naddef, George Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi and Laurence A. Wolsey, ed. (2009). *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Springer. ISBN 978-3-540-68274-5.
- Der-San Chen; Robert G. Batson; Yu Dang (2010). *Applied Integer Programming: Modeling and Solution*. John Wiley and Sons. ISBN 978-0-470-37306-4.

35.9 External links

- A Tutorial on Integer Programming
- Conference Integer Programming and Combinatorial Optimization, IPCO
- The Aussois Combinatorial Optimization Workshop

Chapter 36

Branch and bound

Branch and bound (BB or B&B) is an algorithm design paradigm for discrete and combinatorial optimization problems, as well as general real valued problems. A branch-and-bound algorithm consists of a systematic enumeration of candidate solutions by means of state space search: the set of candidate solutions is thought of as forming a *rooted tree* with the full set at the root. The algorithm explores *branches* of this tree, which represent subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated *bounds* on the optimal solution, and is discarded if it cannot produce a better solution than the best one found so far by the algorithm.

The method was first proposed by A. H. Land and A. G. Doig^[1] in 1960 for *discrete programming*, and has become the most commonly used tool for solving **NP-hard** optimization problems.^[2] The name “branch and bound” first occurred in the work of Little *et al.* on the *traveling salesman problem*.^{[3][4]}

36.1 Overview

In order to facilitate a concrete description, we assume that the goal is to find the *minimum* value of a function $f(x)$, where x ranges over some set S of *admissible* or *candidate solutions* (the *search space* or *feasible region*). Note that one can find the *maximum* value of $f(x)$ by finding the minimum of $g(x) = -f(x)$. (For example, S could be the set of all possible trip schedules for a bus fleet, and $f(x)$ could be the expected revenue for schedule x .)

A branch-and-bound procedure requires two tools. The first one is a *splitting* procedure that, given a set S of candidates, returns two or more smaller sets S_1, S_2, \dots whose union covers S . Note that the minimum of $f(x)$ over S is $\min\{v_1, v_2, \dots\}$, where each v_i is the minimum of $f(x)$ within S_i . This step is called **branching**, since its recursive application defines a *search tree* whose *nodes* are the subsets of S .

The second tool is a procedure that computes upper and lower bounds for the minimum value of $f(x)$ within a given subset of S . This step is called **bounding**.

The key idea of the BB algorithm is: if the *lower* bound for some tree node (set of candidates) A is greater than the *upper* bound for some other node B , then A may be safely discarded from the search. This step is called **pruning**, and is usually implemented by maintaining a global variable m (shared among all nodes of the tree) that records the minimum upper bound seen among all subregions examined so far. Any node whose lower bound is greater than m can be discarded.

The recursion stops when the current candidate set S is reduced to a single element, or when the upper bound for set S matches the lower bound. Either way, any element of S will be a minimum of the function within S .

When x is a vector of \mathbb{R}^n , branch and bound algorithms can be combined with *interval analysis*^[5] and *contractor* techniques in order to provide guaranteed enclosures of the global minimum.^{[6][7]}

36.1.1 Generic version

The following is the skeleton of a generic branch and bound algorithm for minimizing an arbitrary objective function f .^[2] To obtain an actual algorithm from this, one requires a bounding function g , that computes lower bounds of f on nodes of the search tree, as well as a problem-specific branching rule.

- Using a *heuristic*, find a solution x_h to the optimization problem. Store its value, $B = f(x_h)$. (If no heuristic is available, set B to infinity.) B will denote the best solution found so far, and will be used as an upper bound on candidate solutions.
- Initialize a queue to hold a partial solution with none of the variables of the problem assigned.
- Loop until the queue is empty:
 - Take a node N off the queue.
 - If N represents a single candidate solution x and $f(x) < B$, then x is the best solution so far. Record it and set $B \leftarrow f(x)$.
 - Else, *branch* on N to produce new nodes N_i . For each of these:

- If $g(N_i) > B$, do nothing; since the lower bound on this node is greater than the upper bound of the problem, it will never lead to the optimal solution, and can be discarded.
- Else, store N_i on the queue.

Several different queue data structures can be used. A stack (LIFO queue) will yield a depth-first algorithm. A best-first branch and bound algorithm can be obtained by using a priority queue that sorts nodes on their g-value.^[2] The depth-first variant is recommended when no good heuristic is available for producing an initial solution, because it quickly produces full solutions, and therefore upper bounds.^[8]

36.2 Applications

This approach is used for a number of NP-hard problems

- Integer programming
- Nonlinear programming
- Travelling salesman problem (TSP)^{[3][9]}
- Quadratic assignment problem (QAP)
- Maximum satisfiability problem (MAX-SAT)
- Nearest neighbor search (NNS)
- Cutting stock problem
- False noise analysis (FNA)
- Computational phylogenetics
- Set inversion
- Parameter estimation
- 0/1 knapsack problem
- Feature selection in machine learning^[10]
- k-nearest neighbor search^[11]
- Structured prediction in computer vision^{[12]:267–276}

Branch-and-bound may also be a base of various heuristics. For example, one may wish to stop branching when the gap between the upper and lower bounds becomes smaller than a certain threshold. This is used when the solution is “good enough for practical purposes” and can greatly reduce the computations required. This type of solution is particularly applicable when the cost function used is *noisy* or is the result of statistical estimates and so is not known precisely but rather only known to lie within a range of values with a specific probability.

36.3 See also

- Alpha-beta pruning
- Backtracking
- Branch-and-cut, a hybrid between branch-and-bound and the cutting plane methods that is used extensively for solving integer linear programs.

36.4 References

- [1] A. H. Land and A. G. Doig (1960). “An automatic method of solving discrete programming problems”. *Econometrica* **28** (3). pp. 497–520. doi:10.2307/1910129.
- [2] Clausen, Jens (1999). *Branch and Bound Algorithms—Principles and Examples* (Technical report). University of Copenhagen.
- [3] Little, John D. C.; Murty, Katta G.; Sweeney, Dura W.; Karel, Caroline (1963). “An algorithm for the traveling salesman problem”. *Operations Research* **11** (6): 972–989. doi:10.1287/opre.11.6.972.
- [4] Balas, Egon; Toth, Paolo (1983). Branch and bound methods for the traveling salesman problem (Report) (Management Science Research Report MSRR-488). Carnegie Mellon University Graduate School of Industrial Administration.
- [5] Moore, R. E. (1966). *Interval Analysis*. Englewood Cliff, New Jersey: Prentice-Hall. ISBN 0-13-476853-1.
- [6] Jaulin, L.; Kieffer, M.; Didrit, O.; Walter, E. (2001). *Applied Interval Analysis*. Berlin: Springer. ISBN 1-85233-219-0.
- [7] Hansen, E.R. (1992). *Global Optimization using Interval Analysis*. New York: Marcel Dekker.
- [8] Mehlhorn, Kurt; Sanders, Peter (2008). *Algorithms and data structures: The basic toolbox*. Springer. p. 249.
- [9] Conway, Richard Walter; Maxwell, William L.; Miller, Louis W. (2003). *Theory of Scheduling*. Courier Dover Publications. pp. 56–61.
- [10] Narendra, Patrenahalli M.; Fukunaga, K. (1977). “A branch and bound algorithm for feature subset selection”. *IEEE Transactions on Computers* **C-26** (9): 917–922. doi:10.1109/TC.1977.1674939.
- [11] Fukunaga, Keinosuke; Narendra, Patrenahalli M. (1975). “A branch and bound algorithm for computing k-nearest neighbors”. *IEEE Transactions on Computers*: 750–753.
- [12] Nowozin, Sebastian; Lampert, Christoph H. (2011). “Structured Learning and Prediction in Computer Vision”. *Foundations and Trends in Computer Graphics and Vision* **6** (3–4): 185–365. doi:10.1561/0600000033. ISBN 978-1-60198-457-9.

Chapter 37

Branch and cut

Branch and cut^[1] (sometimes written as *branch-and-cut*) is a method of combinatorial optimization for solving integer linear programs (ILPs), that is, linear programming (LP) problems where some or all the unknowns are restricted to integer values.^[2] Branch and cut involves running a branch and bound algorithm and using cutting planes to tighten the linear programming relaxations. Note that if cuts are only used to tighten the initial LP relaxation, the algorithm is called **cut and branch**.

37.1 Description of the Algorithm

The method solves the linear program without the integer constraint using the regular simplex algorithm. When an optimal solution is obtained, and this solution has a non-integer value for a variable that is supposed to be integer, a cutting plane algorithm may be used to find further linear constraints which are satisfied by all feasible integer points but violated by the current fractional solution. These inequalities may be added to the linear program, such that resolving it will yield a different solution which is hopefully “less fractional”.

At this point, the branch and bound part of the algorithm is started. The problem is split into multiple (usually two) versions. The new linear programs are then solved using the simplex method and the process repeats. During the branch and bound process, non-integral solutions to LP relaxations serve as upper bounds and integral solutions serve as lower bounds. A node can be pruned if an upper bound is lower than an existing lower bound. Further, when solving the LP relaxations, additional cutting planes may be generated, which may be either *global cuts*, i.e., valid for all feasible integer solutions, or *local cuts*, meaning that they are satisfied by all solutions fulfilling the side constraints from the currently considered branch and bound subtree.

The algorithm is summarized below. The algorithm assumes the ILP is a maximization problem.

1. Add the initial ILP to L , the list of active problems
2. Set $x^* = \text{null}$ and $v^* = -\infty$
3. while the L is not empty

- (a) Select and remove a problem from L
- (b) Solve the LP relaxation of the problem.
- (c) If the solution is infeasible, go back to 3 (while). Otherwise denote the solution by x with objective value v .
- (d) If $v \leq v^*$, go back to 3.
- (e) If x is integer, set $v^* \leftarrow v, x^* \leftarrow x$ and go back to 3.
- (f) If desired, search for cutting planes that are violated by x . If any are found, add them to the LP relaxation and return to 3.2.
- (g) Branch to partition the problem into new problems with restricted feasible regions. Add these problem to L and go back to 3

4. return x^*

37.2 Branching Strategies

An important step in the branch and cut algorithm is the branching step. At this step, there are a variety of branching heuristics that can be used. The branching strategies described below all involve what is called **branching on a variable**.^[3] Branching on a variable involves choosing a variable, x_i , with a fractional value, x'_i , in the optimal solution to the current LP relaxation and then adding the constraints $x_i \leq \lfloor x'_i \rfloor$ and $x_i \geq \lceil x'_i \rceil$

- **Most Infeasible Branching** This branching strategy chooses the variable with the fractional part closest to 0.5.
- **Pseudo Cost Branching** The basic idea of this strategy is to keep track for each variable x_i the change in the objective function when this variable was previously chosen as the variable to branch on. The strategy then chooses the variable that is predicted to have the most change on the objective function based on past changes when it was chosen as the branching variable. Note that pseudo cost branching is initially uninformative in the search since few variables have been branched on.

- **Strong Branching** Strong branching involves testing which of the candidate variable gives the best improvement to the objective function before actually branching on them. **Full strong branching** tests all candidate variables and can be computationally expensive. The computational cost can be reduced by only considering a subset of the candidate variables and not solving each of the corresponding LP relaxations to completion.

There are also a large number of variations of these branching strategies, such as using strong branching early on when pseudo cost branching is relatively uninformative and then switching to pseudo cost branching later when there is enough branching history for pseudo cost to be informative.

37.3 External links

- Mixed Integer Programming
- BranchAndCut.org FAQ
- SCIP an open source framework for branch-cut-and-price and a mixed integer programming solver
- ABACUS - A Branch-And-CUT System - open source software
- COIN-OR Cbc - open source software

37.4 References

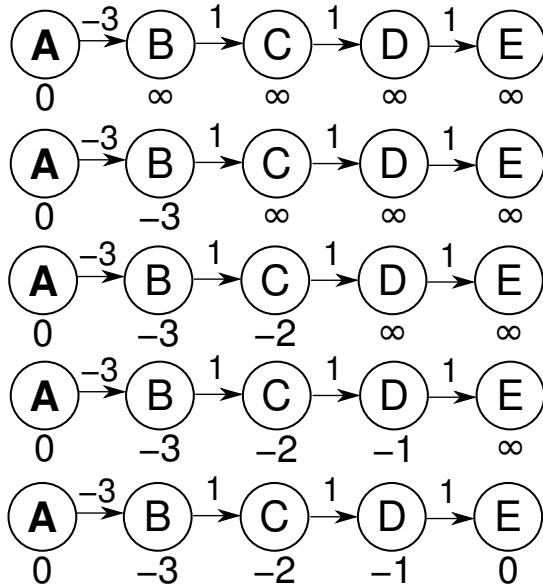
- [1] Padberg, M. and Rinaldi, G. (1991). “A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems”. *Siam Review*: 60–100. doi:10.1137/1033004.
- [2] John E., Mitchell (2002). “Branch-and-Cut Algorithms for Combinatorial Optimization Problems”. *Handbook of Applied Optimization*: 65–77.
- [3] Achterberg, T.; T. Koch; A. Martin (2005). “Branching rules revisited”. *Operations Research* **33** (1): 42–54. doi:10.1016/j.orl.2004.04.002.

Chapter 38

Bellman–Ford algorithm

The **Bellman–Ford algorithm** is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph.^[1] It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. The algorithm is usually named after two of its developers, Richard Bellman and Lester Ford, Jr., who published it in 1958 and 1956, respectively; however, Edward F. Moore also published the same algorithm in 1957, and for this reason it is also sometimes called the **Bellman–Ford–Moore algorithm**.^[1]

Negative edge weights are found in various applications of graphs, hence the usefulness of this algorithm.^[2] If a graph contains a “negative cycle” (i.e. a cycle whose edges sum to a negative value) that is reachable from the source, then there is no *cheapest* path: any path can be made cheaper by one more walk around the negative cycle. In such a case, the Bellman–Ford algorithm can detect negative cycles and report their existence.^{[3][1]}



–1 or 4 iterations for the distance estimates to converge. Conversely, if the edges are processed in the best order, from left to right, the algorithm converges in a single iteration.

38.1 Algorithm

Like Dijkstra's Algorithm, Bellman–Ford is based on the principle of relaxation, in which an approximation to the correct distance is gradually replaced by more accurate values until eventually reaching the optimum solution. In both algorithms, the approximate distance to each vertex is always an overestimate of the true distance, and is replaced by the minimum of its old value with the length of a newly found path. However, Dijkstra's algorithm greedily selects the minimum-weight node that has not yet been processed, and performs this relaxation process on all of its outgoing edges; by contrast, the Bellman–Ford algorithm simply relaxes *all* the edges, and does this $|V| - 1$ times, where $|V|$ is the number of vertices in the graph. In each of these repetitions, the number of vertices with correctly calculated distances grows, from which it follows that eventually all vertices will have their correct distances. This method allows the Bellman–Ford algorithm to be applied to a wider class of inputs than Dijkstra.

Bellman–Ford runs in $O(|V| \cdot |E|)$ time, where $|V|$ and $|E|$ are the number of vertices and edges respectively.

```
function BellmanFord(list vertices, list edges, vertex source) ::distance[],predecessor[] // This implementation takes in a graph, represented as // lists of vertices and edges, and fills two arrays // (distance and predecessor) with shortest-path // (less cost/distance/metric) information // Step 1: initialize graph for each vertex v in vertices: if v is source then distance[v] := 0 else distance[v] := inf predecessor[v] := null // Step 2: relax edges repeatedly for i from 1 to size(vertices)-1: for each edge (u, v) with weight w in edges: if distance[u] + w < distance[v]: distance[v] := distance[u] + w predecessor[v] := u // Step 3: check for negative-weight cycles for each edge (u, v) with weight w in edges: if distance[u] + w < distance[v]: error "Graph contains a negative-weight cycle" return distance[], predecessor[]
```

Simply put, the algorithm initializes the distance to the source to 0 and all other nodes to infinity. Then for all edges, if the distance to the destination can be shortened

by taking the edge, the distance is updated to the new lower value. At each iteration i that the edges are scanned, the algorithm finds all shortest paths of at most length i edges. Since the longest possible path without a cycle can be $|V| - 1$ edges, the edges must be scanned $|V| - 1$ times to ensure the shortest path has been found for all nodes. A final scan of all the edges is performed and if any distance is updated, then a path of length $|V|$ edges has been found which can only occur if at least one negative cycle exists in the graph.

38.2 Proof of correctness

The correctness of the algorithm can be shown by induction. The precise statement shown by induction is:

Lemma. After i repetitions of *for* loop:

- If $\text{Distance}(u)$ is not infinity, it is equal to the length of some path from s to u ;
- If there is a path from s to u with at most i edges, then $\text{Distance}(u)$ is at most the length of the shortest path from s to u with at most i edges.

Proof. For the base case of induction, consider $i=0$ and the moment before *for* loop is executed for the first time. Then, for the source vertex, $\text{source.distance} = 0$, which is correct. For other vertices u , $u.\text{distance} = \text{infinity}$, which is also correct because there is no path from *source* to u with 0 edges.

For the inductive case, we first prove the first part. Consider a moment when a vertex's distance is updated by $v.\text{distance} := u.\text{distance} + uv.\text{weight}$. By inductive assumption, $u.\text{distance}$ is the length of some path from *source* to u . Then $u.\text{distance} + uv.\text{weight}$ is the length of the path from *source* to v that follows the path from *source* to u and then goes to v .

For the second part, consider the shortest path from *source* to u with at most i edges. Let v be the last vertex before u on this path. Then, the part of the path from *source* to v is the shortest path from *source* to v with at most $i-1$ edges. By inductive assumption, $v.\text{distance}$ after $i-1$ iterations is at most the length of this path. Therefore, $uv.\text{weight} + v.\text{distance}$ is at most the length of the path from s to u . In the i^{th} iteration, $u.\text{distance}$ gets compared with $uv.\text{weight} + v.\text{distance}$, and is set equal to it if $uv.\text{weight} + v.\text{distance}$ was smaller. Therefore, after i iteration, $u.\text{distance}$ is at most the length of the shortest path from *source* to u that uses at most i edges.

If there are no negative-weight cycles, then every shortest path visits each vertex at most once, so at step 3 no further improvements can be made. Conversely, suppose no improvement can be made. Then for any cycle with vertices $v[0], \dots, v[k-1]$,

$$v[i].\text{distance} \leq v[(i-1) \bmod k].\text{distance} + v[(i-1) \bmod k]v[i].\text{weight}$$

Summing around the cycle, the $v[i].\text{distance}$ terms and the $v[i-1 \bmod k].\text{distance}$ terms cancel, leaving

$$0 \leq \sum_{i=1}^k v[i-1 \bmod k]v[i].\text{weight}$$

I.e., every cycle has nonnegative weight.

38.3 Finding negative cycles

When the algorithm is used to find shortest paths, the existence of negative cycles is a problem, preventing the algorithm from finding a correct answer. However, since it terminates upon finding a negative cycle, the Bellman–Ford algorithm can be used for applications in which this is the target to be sought – for example in cycle-cancelling techniques in network flow analysis.^[1]

38.4 Applications in routing

A distributed variant of the Bellman–Ford algorithm is used in **distance-vector routing protocols**, for example the **Routing Information Protocol (RIP)**. The algorithm is distributed because it involves a number of nodes (routers) within an Autonomous system, a collection of IP networks typically owned by an ISP. It consists of the following steps:

1. Each node calculates the distances between itself and all other nodes within the AS and stores this information as a table.
2. Each node sends its table to all neighboring nodes.
3. When a node receives distance tables from its neighbors, it calculates the shortest routes to all other nodes and updates its own table to reflect any changes.

The main disadvantages of the Bellman–Ford algorithm in this setting are as follows:

- It does not scale well.
- Changes in network topology are not reflected quickly since updates are spread node-by-node.
- Count to infinity if link or node failures render a node unreachable from some set of other nodes, those nodes may spend forever gradually increasing their estimates of the distance to it, and in the meantime there may be routing loops.

38.5 Improvements

The Bellman–Ford algorithm may be improved in practice (although not in the worst case) by the observation that, if an iteration of the main loop of the algorithm terminates without making any changes, the algorithm can be immediately terminated, as subsequent iterations will not make any more changes. With this early termination condition, the main loop may in some cases use many fewer than $|V| - 1$ iterations, even though the worst case of the algorithm remains unchanged.

Yen (1970) described two more improvements to the Bellman–Ford algorithm for a graph without negative-weight cycles; again, while making the algorithm faster in practice, they do not change its $O(|V|^2|E|)$ worst case time bound. His first improvement reduces the number of relaxation steps that need to be performed within each iteration of the algorithm. If a vertex v has a distance value that has not changed since the last time the edges out of v were relaxed, then there is no need to relax the edges out of v a second time. In this way, as the number of vertices with correct distance values grows, the number whose outgoing edges need to be relaxed in each iteration shrinks, leading to a constant-factor savings in time for dense graphs.

Yen's second improvement first assigns some arbitrary linear order on all vertices and then partitions the set of all edges into two subsets. The first subset, E_f , contains all edges (vi, vj) such that $i < j$; the second, E_b , contains edges (vi, vj) such that $i > j$. Each vertex is visited in the order $v_1, v_2, \dots, v_{|V|}$, relaxing each outgoing edge from that vertex in E_f . Each vertex is then visited in the order $v_{|V|}, v_{|V|-1}, \dots, v_1$, relaxing each outgoing edge from that vertex in E_b . Each iteration of the main loop of the algorithm, after the first one, adds at least two edges to the set of edges whose relaxed distances match the correct shortest path distances: one from E_f and one from E_b . This modification reduces the worst-case number of iterations of the main loop of the algorithm from $|V| - 1$ to $|V|/2$.^{[4][5]}

Another improvement, by Bannister & Eppstein (2012), replaces the arbitrary linear order of the vertices used in Yen's second improvement by a random permutation. This change makes the worst case for Yen's improvement (in which the edges of a shortest path strictly alternate between the two subsets E_f and E_b) very unlikely to happen. With a randomly permuted vertex ordering, the expected number of iterations needed in the main loop is at most $|V|/3$.^[5]

38.6 Notes

[1] Bang-Jensen & Gutin (2000)

[2] Sedgewick (2002).

- [3] Kleinberg & Tardos (2006).
- [4] Cormen et al., 2nd ed., Problem 24-1, pp. 614–615.
- [5] See Sedgewick's web exercises for *Algorithms*, 4th ed., exercises 5 and 11 (retrieved 2013-01-30).

38.7 References

38.7.1 Original sources

- Bellman, Richard (1958). “On a routing problem”. *Quarterly of Applied Mathematics* **16**: 87–90. MR 0102435.
- Ford Jr., Lester R. (August 14, 1956). *Network Flow Theory*. Paper P-923. Santa Monica, California: RAND Corporation.
- Moore, Edward F. (1959). *The shortest path through a maze*. Proc. Internat. Sympos. Switching Theory 1957, Part II. Cambridge, Mass.: Harvard Univ. Press. pp. 285–292. MR 0114710.
- Yen, Jin Y. (1970). “An algorithm for finding shortest routes from all source nodes to a given destination in general networks”. *Quarterly of Applied Mathematics* **27**: 526–530. MR 0253822.
- Bannister, M. J.; Eppstein, D. (2012). *Randomized speedup of the Bellman–Ford algorithm*. Analytic Algorithmics and Combinatorics (ANALCO12), Kyoto, Japan. pp. 41–47. arXiv:1111.5414.

38.7.2 Secondary sources

- Bang-Jensen, Jørgen; Gutin, Gregory (2000). “Section 2.3.4: The Bellman–Ford–Moore algorithm”. *Digraphs: Theory, Algorithms and Applications* (First ed.). ISBN 978-1-84800-997-4.
- Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L.. *Introduction to Algorithms*. MIT Press and McGraw-Hill., Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 24.1: The Bellman–Ford algorithm, pp. 588–592. Problem 24-1, pp. 614–615. Third Edition. MIT Press, 2009. ISBN 978-0-262-53305-8. Section 24.1: The Bellman–Ford algorithm, pp. 651–655.
- Heineman, George T.; Pollice, Gary; Selkow, Stanley (2008). “Chapter 6: Graph Algorithms”. *Algorithms in a Nutshell*. O'Reilly Media. pp. 160–164. ISBN 978-0-596-51624-6.
- Kleinberg, Jon; Tardos, Éva (2006). *Algorithm Design*. New York: Pearson Education, Inc.

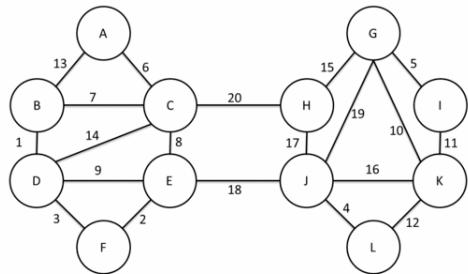
- Sedgewick, Robert (2002). “Section 21.7: Negative Edge Weights”. *Algorithms in Java* (3rd ed.). ISBN 0-201-36121-3.

38.8 External links

- C++ code example
- Open Source Java Graph package with Bellman–Ford Algorithms

Chapter 39

Borůvka's algorithm



An animation, describing Boruvka's (Sollin's) algorithm, for finding a minimum spanning tree in a graph - An example on the runtime of the algorithm

Borůvka's algorithm is an algorithm for finding a minimum spanning tree in a graph for which all edge weights are distinct.

It was first published in 1926 by Otakar Borůvka as a method of constructing an efficient electricity network for Moravia.^{[1][2][3]} The algorithm was rediscovered by Choquet in 1938;^[4] again by Florek, Łukasiewicz, Perkal, Steinhaus, and Zubrzycki^[5] in 1951; and again by Sollin^[6] in 1965. Because Sollin was the only computer scientist in this list living in an English speaking country, this algorithm is frequently called **Sollin's algorithm**, especially in the parallel computing literature.

The algorithm begins by first examining each vertex and adding the cheapest edge from that vertex to another in the graph, without regard to already added edges, and continues joining these groupings in a like manner until a tree spanning all vertices is completed.

39.1 Pseudocode

Designating each vertex or set of connected vertices a "component", pseudocode for Borůvka's algorithm is:

1. Input: A connected graph G whose edges have distinct weights
2. Initialize a forest T to be a set of one-vertex trees, one for each vertex of the graph.

3. While T has more than one component:
4. For each component C of T :
5. Begin with an empty set of edges S
6. For each vertex v in C :
7. Find the cheapest edge from v to a vertex outside of C , and add it to S
8. Add the cheapest edge in S to T
9. Output: T is the minimum spanning tree of G .

As in Kruskal's algorithm, tracking components of T can be done efficiently using a disjoint-set data structure. In graphs where edges have identical weights, edges with equal weights can be ordered based on the lexicographic order of their endpoints.

39.2 Complexity

Borůvka's algorithm can be shown to take $O(\log V)$ iterations of the outer loop until it terminates, and therefore to run in time $O(E \log V)$, where E is the number of edges, and V is the number of vertices in G . In planar graphs, and more generally in families of graphs closed under graph minor operations, it can be made to run in linear time, by removing all but the cheapest edge between each pair of components after each stage of the algorithm.^[7]

39.3 Example

39.4 Other algorithms

Other algorithms for this problem include Prim's algorithm and Kruskal's algorithm. Fast parallel algorithms can be obtained by combining Prim's algorithm with Borůvka's.^[8]

A faster randomized minimum spanning tree algorithm based in part on Borůvka's algorithm due to Karger, Klein, and Tarjan runs in expected $O(E)$ time.^[9] The

best known (deterministic) minimum spanning tree algorithm by Bernard Chazelle is also based in part on Borůvka's and runs in $O(E \alpha(E, V))$ time, where α is the inverse of the Ackermann function.^[10] These randomized and deterministic algorithms combine steps of Borůvka's algorithm, reducing the number of components that remain to be connected, with steps of a different type that reduce the number of edges between pairs of components.

39.5 Notes

- [1] Borůvka, Otakar (1926). "O jistém problému minimálním (About a certain minimal problem)". *Práce mor. přírodověd. spol. v Brně III* (in Czech, German summary) **3**: 37–58.
- [2] Borůvka, Otakar (1926). "Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí (Contribution to the solution of a problem of economical construction of electrical networks)". *Elektronický Obzor* (in Czech) **15**: 153–154.
- [3] Nešetřil, Jaroslav; Milková, Eva; Nešetřilová, Helena (2001). "Otakar Borůvka on minimum spanning tree problem: translation of both the 1926 papers, comments, history". *Discrete Mathematics* **233** (1–3): 3–36. doi:10.1016/S0012-365X(00)00224-7. MR 1825599.
- [4] Choquet, Gustave (1938). "Étude de certains réseaux de routes". *Comptes-rendus de l'Académie des Sciences* (in French) **206**: 310–313.
- [5] Florek, Kazimierz (1951). "Sur la liaison et la division des points d'un ensemble fini". *Colloquium Mathematicum* **2** (1951) (in French): 282–285.
- [6] Sollin, M. (1965). "Le tracé de canalisation". *Programming, Games, and Transportation Networks* (in French).
- [7] Eppstein, David (1999). "Spanning trees and spanners". In Sack, J.-R.; Urrutia, J.. *Handbook of Computational Geometry*. Elsevier. pp. 425–461.; Mareš, Martin (2004). "Two linear time algorithms for MST on minor closed graph classes". *Archivum mathematicum* **40** (3): 315–320..
- [8] Bader, David A.; Cong, Guojing (1 November 2006). "Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs". *Journal of Parallel and Distributed Computing* **66** (11): 1366–1378. doi:10.1016/j.jpdc.2006.06.001.
- [9] Karger, David R.; Klein, Philip N.; Tarjan, Robert E. (1 March 1995). "A randomized linear-time algorithm to find minimum spanning trees". *Journal of the ACM* **42** (2): 321–328. doi:10.1145/201019.201022.
- [10] Chazelle, Bernard (2000). "A minimum spanning tree algorithm with inverse-Ackermann type complexity". *J. ACM* **47** (6): 1028–1047. doi:10.1145/355541.355562.

Chapter 40

Dijkstra's algorithm

Not to be confused with Dykstra's projection algorithm.

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published in 1959.^{[1][2]} The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes,^[2] but a more common variant fixes a single node as the “source” node and finds shortest paths from the source to all other nodes in the graph, producing a shortest path tree.

For a given source node in the graph, the algorithm finds the shortest path between that node and every other.^{[3]:196–206} It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, if the nodes of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path algorithm is widely used in network routing protocols, most notably IS-IS and Open Shortest Path First (OSPF). It is also employed as a subroutine in other algorithms such as Johnson's.

Dijkstra's original algorithm does not use a min-priority queue and runs in time $O(|V|^2)$ (where $|V|$ is the number of nodes). The idea of this algorithm is also given in (Leyzorek et al. 1957). The implementation based on a min-priority queue implemented by a Fibonacci heap and running in $O(|E| + |V| \log |V|)$ (where $|E|$ is the number of edges) is due to (Fredman & Tarjan 1984). This is asymptotically the fastest known single-source shortest-path algorithm for arbitrary directed graphs with unbounded non-negative weights.

In some fields, artificial intelligence in particular, Dijkstra's algorithm or a variant of it is known as **uniform-cost search** and formulated as an instance of the more general idea of best-first search.^[4]

40.1 History

Dijkstra thought about the shortest path problem when working at the Mathematical Center in Amsterdam in 1956 as a programmer to demonstrate capabilities of a new computer called ARMAC. His objective was to choose both a problem as well as an answer (that would be produced by computer) that non-computing people could understand. He designed the shortest path algorithm in about 20 minutes without aid of paper and pen and later implemented it for ARMAC for a slightly simplified transportation map of 64 cities in Netherland (ARMAC was a 6-bit computer and hence could hold 64 cities comfortably).^[1] A year later, he came across another problem from hardware engineers working on the institute's next computer: minimize the amount of wire needed to connect the pins on the back panel of the machine. As a solution, he re-discovered the algorithm known as Prim's minimal spanning tree algorithm (known earlier to Jarník, and also rediscovered by Prim).^{[5][6]} Dijkstra published the algorithm in 1959, two years after Prim and 29 years after Jarník.^{[7][8]}

40.2 Algorithm

Let the node at which we are starting be called the **initial node**. Let the **distance of node Y** be the distance from the **initial node** to Y . Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.
3. For the current node, consider all of its unvisited neighbors and calculate their *tentative* distances. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one. For example, if the current node A is marked



Illustration of Dijkstra's algorithm search for finding path from a start node (lower left, red) to a goal node (upper right, green) in a robot motion planning problem. Open nodes represent the “tentative” set. Filled nodes are visited ones, with color representing the distance: the greener, the farther. Nodes in all the different directions are explored uniformly, appearing as a more-or-less circular wavefront as Dijkstra's algorithm uses a heuristic identically equal to 0.

with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B (through A) will be $6 + 2 = 8$. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.

4. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Select the unvisited node that is marked with the smallest tentative distance, and set it as the new “current node” then go back to step 3.

40.3 Description

Note: For ease of understanding, this discussion uses the terms **intersection**, **road** and **map** — however, in formal terminology these terms are **vertex**, **edge** and **graph**, respectively.

Suppose you would like to find the shortest path between two **intersections** on a city map, a starting point and a destination. The order is conceptually simple: to start, mark the distance to every intersection on the map with infinity. This is done not to imply there is an infinite distance, but to note that intersection has not yet been *visited*; some variants of this method simply leave the intersection unlabeled. Now, at each iteration, select a *current* intersection. For the first iteration, the current intersection will be the starting point and the distance to it (the intersection’s label) will be zero. For subsequent iterations (after the first), the current intersection will be the closest unvisited intersection to the starting point—this will be easy to find.

From the current intersection, update the distance to every unvisited intersection that is directly connected to it. This is done by determining the sum of the distance between an unvisited intersection and the value of the current intersection, and **relabeling** the unvisited intersection with this value if it is less than its current value. In effect, the intersection is relabeled if the path to it through the current intersection is shorter than the previously known paths. To facilitate shortest path identification, in pencil, mark the road with an arrow pointing to the relabeled intersection if you label/relabel it, and erase all others pointing to it. After you have updated the distances to each **neighboring** intersection, mark the current intersection as *visited* and select the unvisited intersection with lowest distance (from the starting point) – or the lowest label—as the current intersection. Nodes marked as visited are labeled with the shortest path from the starting point to it and will not be revisited or returned to.

Continue this process of updating the neighboring intersections with the shortest distances, then marking the current intersection as visited and moving onto the closest unvisited intersection until you have marked the destination as visited. Once you have marked the destination as visited (as is the case with any visited intersection) you have determined the shortest path to it, from the starting point, and can trace your way back, following the arrows in reverse.

This algorithm makes no attempt to direct “exploration” towards the destination as one might expect. Rather, the sole consideration in determining the next “current” intersection is its distance from the starting point. This algorithm therefore expands outward from the starting point, interactively considering every node that is closer in terms of shortest path distance until it reaches the destination. When understood in this way, it is clear how the algorithm necessarily finds the shortest path. However, it may also reveal one of the algorithm’s weaknesses: its relative slowness in some topologies.

40.4 Pseudocode

In the following algorithm, the code $u \leftarrow \text{vertex in } Q$ with $\min \text{dist}[u]$, searches for the vertex u in the vertex set Q that has the least $\text{dist}[u]$ value. $\text{length}(u, v)$ returns the length of the edge joining (i.e. the distance between) the two neighbor-nodes u and v . The variable alt on line 19 is the length of the path from the root node to the neighbor node v if it were to go through u . If this path is shorter than the current shortest path recorded for v , that current path is replaced with this alt path. The prev array is populated with a pointer to the “next-hop” node on the source graph to get the shortest route to the source.

```

1 function Dijkstra(Graph, source): 2 3  $\text{dist}[\text{source}] \leftarrow 0$  //Distance from source to source 4  $\text{prev}[\text{source}] \leftarrow \text{undefined}$  // Previous node in optimal path initialization 5 6
for each vertex  $v$  in Graph: //Initialization 7 if  $v \neq \text{source}$ 
// Where  $v$  has not yet been removed from  $Q$  (unvisited nodes) 8  $\text{dist}[v] \leftarrow \infty$  // Unknown distance function
from source to  $v$  9  $\text{prev}[v] \leftarrow \text{undefined}$  // Previous node
in optimal path from source 10 end if 11 add  $v$  to  $Q$  //
All nodes initially in  $Q$  (unvisited nodes) 12 end for 13
14 while  $Q$  is not empty: 15  $u \leftarrow \text{vertex in } Q$  with  $\min \text{dist}[u]$  //Source node in first case 16 remove  $u$  from  $Q$  17
18 for each neighbor  $v$  of  $u$ : // where  $v$  is still in  $Q$ . 19  $\text{alt} \leftarrow \text{dist}[u] + \text{length}(u, v)$  20 if  $\text{alt} < \text{dist}[v]$ : // A shorter
path to  $v$  has been found 21  $\text{dist}[v] \leftarrow \text{alt}$  22  $\text{prev}[v] \leftarrow u$ 
23 end if 24 end for 25 end while 26 27 return  $\text{dist}[], \text{prev}[]$  28 29 end function
```

If we are only interested in a shortest path between vertices source and target, we can terminate the search after line 15 if $u = \text{target}$. Now we can read the shortest path from source to target by reverse iteration:

```

1  $S \leftarrow \text{empty sequence}$  2  $u \leftarrow \text{target}$  3 while  $\text{prev}[u]$  is defined: // Construct the shortest path with a stack  $S$  4 insert
 $u$  at the beginning of  $S$  // Push the vertex onto the stack 5
 $u \leftarrow \text{prev}[u]$  // Traverse from target to source 6 end while
```

Now sequence S is the list of vertices constituting one of the shortest paths from source to target, or the empty sequence if no path exists.

A more general problem would be to find all the shortest paths between source and target (there might be several different ones of the same length). Then instead of storing only a single node in each entry of $\text{prev}[]$ we would store all nodes satisfying the relaxation condition. For example, if both r and source connect to target and both of them lie on different shortest paths through target (because the edge cost is the same in both cases), then we would add both r and source to $\text{prev}[\text{target}]$. When the algorithm completes, $\text{prev}[]$ data structure will actually describe a graph that is a subset of the original graph with some edges removed. Its key property will be that if the algorithm was run with some starting node, then every path from that node to any other node in the new graph will be the shortest path between those nodes in the original graph, and all paths of that length from the original

graph will be present in the new graph. Then to actually find all these shortest paths between two given nodes we would use a path finding algorithm on the new graph, such as **depth-first search**.

40.4.1 Using a priority queue

A **min-priority queue** is an abstract data structure that provides 3 basic operations : `add_with_priority()`, `decrease_priority()` and `extract_min()`. As mentioned earlier, using such a data structure can lead to faster computing times than using a basic queue. Notably, **Fibonacci heap** (Fredman & Tarjan 1984) or **Brodal queue** offer optimal implementations for those 3 operations. As the algorithm is slightly different, we mention it here, in pseudo-code as well :

```

1 function Dijkstra(Graph, source): 2  $\text{dist}[\text{source}] \leftarrow 0$  // Initialization 3 for each vertex  $v$  in Graph: 4 if  $v \neq \text{source}$ 
5  $\text{dist}[v] \leftarrow \infty$  // Unknown distance from source to
 $v$  6  $\text{prev}[v] \leftarrow \text{undefined}$  // Predecessor of  $v$  7 end if 8
 $Q.\text{add\_with\_priority}(v, \text{dist}[v])$  9 end for 10 11 while  $Q$ 
is not empty: // The main loop 12  $u \leftarrow Q.\text{extract\_min}()$ 
// Remove and return best vertex 13 for each neighbor  $v$ 
of  $u$ : 14  $\text{alt} = \text{dist}[u] + \text{length}(u, v)$  15 if  $\text{alt} < \text{dist}[v]$ : 16
 $\text{dist}[v] \leftarrow \text{alt}$  17  $\text{prev}[v] \leftarrow u$  18  $Q.\text{decrease\_priority}(v,$ 
 $\text{alt})$  19 end if 20 end for 21 end while 22 return  $\text{prev}[]$ 
```

Instead of filling the priority queue with all nodes in the initialization phase, it is also possible to initialize it to contain only *source*; then, inside the **if** $\text{alt} < \text{dist}[v]$ block, the node must be inserted if not already in the queue (instead of performing a `decrease_priority` operation).^{[3]:198}

Other data structures can be used to achieve even faster computing times in practice.^[9]

40.5 Running time

Bounds of the running time of Dijkstra's algorithm on a graph with edges E and vertices V can be expressed as a function of the number of edges, denoted $|E|$, and the number of vertices, denoted $|V|$, using **big-O** notation. How tight a bound is possible depends on the way the vertex set Q is implemented. In the following, upper bounds can be simplified because $|E| = O(|V|^2)$ for any graph, but that simplification disregards the fact that in some problems, other upper bounds on $|E|$ may hold.

For any implementation of the vertex set Q , the running time is in

$$O(|E| \cdot T_{dk} + |V| \cdot T_{em})$$

where T_{dk} and T_{em} are the complexities of the *decrease-key* and *extract-minimum* operations in Q , respectively. The simplest implementation of the Dijkstra's algorithm

stores the vertex set Q as an ordinary linked list or array, and extract-minimum is simply a linear search through all vertices in Q . In this case, the running time is $O(|E| + |V|^2) = O(|V|^2)$.

For sparse graphs, that is, graphs with far fewer than $|V|^2$ edges, Dijkstra's algorithm can be implemented more efficiently by storing the graph in the form of adjacency lists and using a self-balancing binary search tree, binary heap, pairing heap, or Fibonacci heap as a priority queue to implement extracting minimum efficiently. To perform decrease-key steps in a binary heap efficiently, it is necessary to use an auxiliary data structure that maps each vertex to its position in the heap, and to keep this structure up to date as the priority queue Q changes. With a self-balancing binary search tree or binary heap, the algorithm requires

$$\Theta((|E| + |V|) \log |V|)$$

time in the worst case; for connected graphs this time bound can be simplified to $\Theta(|E| \log |V|)$. The Fibonacci heap improves this to

$$O(|E| + |V| \log |V|)$$

When using binary heaps, the average case time complexity is lower than the worst-case: assuming edge costs are drawn independently from a common probability distribution, the expected number of decrease-key operations is bounded by $O(|V| \log(|E|/|V|))$, giving a total running time of^{[3]:199–200}

$$O(|E| + |V| \log \frac{|E|}{|V|} \log |V|)$$

40.5.1 Practical optimizations and infinite graphs

In common presentations of Dijkstra's algorithm, initially all nodes are entered into the priority queue. This is, however, not necessary: the algorithm can start with a priority queue that contains only one item, and insert new items as they are discovered (instead of doing a decrease-key, check whether the key is in the queue; if it is, decrease its key, otherwise insert it).^{[3]:198} This variant has the same worst-case bounds as the common variant, but maintains a smaller priority queue in practice, speeding up the queue operations.^[4]

Moreover, not inserting all nodes in a graph makes it possible to extend the algorithm to find the shortest path from a single source to the closest of a set of target nodes on infinite graphs or those too large to represent in memory. The resulting algorithm is called *uniform-cost search*

(UCS) in the artificial intelligence literature^{[4][10]} and can be expressed in pseudocode as

```
procedure UniformCostSearch(Graph, start, goal)
    node ← start
    cost ← 0
    frontier ← priority queue containing node only explored ← empty set
    do if frontier is empty return failure
    node ← frontier.pop()
    if node is goal return solution
    explored.add(node)
    for each of node's neighbors n
        if n is not in explored if n is not in frontier
            frontier.add(n)
        else if n is in frontier with higher cost
            replace existing node with n
```

The complexity of this algorithm can be expressed in an alternative way for very large graphs: when C^* is the length of the shortest path from the start node to any node satisfying the “goal” predicate, and each edge has cost at least ϵ , then the algorithm's worst-case time and space complexity are both in $O(b^{1+LC^*/\epsilon})$.^[10]

40.5.2 Specialized variants

When arc weights are integers and bounded by a constant C , the usage of a special priority queue structure by Van Emde Boas et al.(1977) (Ahuja et al. 1990) brings the complexity to $O(|E| \log \log |C|)$. Another interesting implementation based on a combination of a new radix heap and the well-known Fibonacci heap runs in time $O(|E| + |V| \sqrt{\log |C|})$ (Ahuja et al. 1990). Finally, the best algorithms in this special case are as follows. The algorithm given by (Thorup 2000) runs in $O(|E| \log \log |V|)$ time and the algorithm given by (Raman 1997) runs in $O(|E| + |V| \min\{(\log |V|)^{1/3+\epsilon}, (\log |C|)^{1/4+\epsilon}\})$ time.

Also, for directed acyclic graphs, it is possible to find shortest paths from a given starting vertex in linear $O(|E| + |V|)$ time, by processing the vertices in a topological order, and calculating the path length for each vertex to be the minimum length obtained via any of its incoming edges.^{[11][12]}

In the special case of integer weights and undirected graphs, the Dijkstra's algorithm can be completely countered with a linear $O(|V| + |E|)$ complexity algorithm, given by (Thorup 1999).

40.6 Related problems and algorithms

The functionality of Dijkstra's original algorithm can be extended with a variety of modifications. For example, sometimes it is desirable to present solutions which are less than mathematically optimal. To obtain a ranked list of less-than-optimal solutions, the optimal solution is first calculated. A single edge appearing in the optimal solution is removed from the graph, and the optimum solution to this new graph is calculated. Each edge of the original solution is suppressed in turn and a new shortest-path

calculated. The secondary solutions are then ranked and presented after the first optimal solution.

Dijkstra's algorithm is usually the working principle behind link-state routing protocols, OSPF and IS-IS being the most common ones.

Unlike Dijkstra's algorithm, the Bellman–Ford algorithm can be used on graphs with negative edge weights, as long as the graph contains no negative cycle reachable from the source vertex s . The presence of such cycles means there is no shortest path, since the total weight becomes lower each time the cycle is traversed. It is possible to adapt Dijkstra's algorithm to handle negative weight edges by combining it with the Bellman–Ford algorithm (to remove negative edges and detect negative cycles), such an algorithm is called Johnson's algorithm.

The A* algorithm is a generalization of Dijkstra's algorithm that cuts down on the size of the subgraph that must be explored, if additional information is available that provides a lower bound on the “distance” to the target. This approach can be viewed from the perspective of linear programming: there is a natural linear program for computing shortest paths, and solutions to its dual linear program are feasible if and only if they form a consistent heuristic (speaking roughly, since the sign conventions differ from place to place in the literature). This feasible dual / consistent heuristic defines a non-negative reduced cost and A* is essentially running Dijkstra's algorithm with these reduced costs. If the dual satisfies the weaker condition of admissibility, then A* is instead more akin to the Bellman–Ford algorithm.

The process that underlies Dijkstra's algorithm is similar to the greedy process used in Prim's algorithm. Prim's purpose is to find a minimum spanning tree that connects all nodes in the graph; Dijkstra is concerned with only two nodes. Prim's does not evaluate the total weight of the path from the starting node, only the individual path.

Breadth-first search can be viewed as a special-case of Dijkstra's algorithm on unweighted graphs, where the priority queue degenerates into a FIFO queue.

Fast marching method can be viewed as a continuous version of Dijkstra's algorithm which computes the geodesic distance on a triangle mesh.

40.6.1 Dynamic programming perspective

From a dynamic programming point of view, Dijkstra's algorithm is a successive approximation scheme that solves the dynamic programming functional equation for the shortest path problem by the Reaching method.^{[13][14][15]}

In fact, Dijkstra's explanation of the logic behind the algorithm,^[16] namely

Problem 2. Find the path of minimum total length between two given nodes P and Q

.

We use the fact that, if R is a node on the minimal path from P to Q , knowledge of the latter implies the knowledge of the minimal path from P to R .

is a paraphrasing of Bellman's famous Principle of Optimality in the context of the shortest path problem.

40.7 See also

- A* search algorithm
- Bellman–Ford algorithm
- Euclidean shortest path
- Flood fill
- Floyd–Warshall algorithm
- Johnson's algorithm
- Longest path problem

40.8 Notes

- [1] Dijkstra, Edsger; Thomas J. Misa, Editor (August 2010). “An Interview with Edsger W. Dijkstra”. *Communications of the ACM* **53** (8): 41–47. doi:10.1145/1787234.1787249. What is the shortest way to travel from Rotterdam to Groningen? It is the algorithm for the shortest path which I designed in about 20 minutes. One morning I was shopping with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path.
- [2] Dijkstra, E. W. (1959). “A note on two problems in connexion with graphs”. *Numerische Mathematik* **1**: 269–271. doi:10.1007/BF01386390.
- [3] Mehlhorn, Kurt; Sanders, Peter (2008). *Algorithms and Data Structures: The Basic Toolbox*. Springer.
- [4] Felner, Ariel (2011). *Position Paper: Dijkstra's Algorithm versus Uniform Cost Search or a Case Against Dijkstra's Algorithm*. Proc. 4th Int'l Symp. on Combinatorial Search. In a route-finding problem, Felner finds that the queue can be a factor 500–600 smaller, taking some 40% of the running time.
- [5] Dijkstra, Edward W., *Reflections on “A note on two problems in connexion with graphs”*
- [6] Tarjan, Robert Endre (1983), *Data Structures and Network Algorithms*, CBMS_NSF Regional Conference Series in Applied Mathematics **44**, Society for Industrial and Applied Mathematics, p. 75, The third classical minimum spanning tree algorithm was discovered by Jarník and rediscovered by Prim and Dijkstra; it is commonly known as Prim's algorithm.

- [7] R. C. Prim: *Shortest connection networks and some generalizations*. In: *Bell System Technical Journal*, 36 (1957), pp. 1389–1401.
 - [8] V. Jarník: *O jistém problému minimálním* [About a certain minimal problem], Práce Moravské Přírodovědecké Společnosti, 6, 1930, pp. 57–63. (in Czech)
 - [9] Chen, M.; Chowdhury, R. A.; Ramachandran, V.; Roche, D. L.; Tong, L. (2007). *Priority Queues and Dijkstra's Algorithm — UTCS Technical Report TR-07-54 — 12 October 2007*. Austin, Texas: The University of Texas at Austin, Department of Computer Sciences.
 - [10] Russell, Stuart; Norvig, Peter (2009) [1995]. *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall. pp. 75, 81. ISBN 978-0-13-604259-4.
 - [11] http://www.boost.org/doc/libs/1_44_0/libs/graph/doc/dag_shortest_paths.html
 - [12] Cormen et al, Introduction to Algorithms & 3ed, chapter-24 2009
 - [13] Sniedovich, M. (2006). “Dijkstra’s algorithm revisited: the dynamic programming connexion” (PDF). *Journal of Control and Cybernetics* 35 (3): 599–620. Online version of the paper with interactive computational modules.
 - [14] Denardo, E.V. (2003). *Dynamic Programming: Models and Applications*. Mineola, NY: Dover Publications. ISBN 978-0-486-42810-9.
 - [15] Sniedovich, M. (2010). *Dynamic Programming: Foundations and Principles*. Francis & Taylor. ISBN 978-0-8247-4099-3.
 - [16] Dijkstra 1959, p. 270
- Leyzorek, M.; Gray, R. S.; Johnson, A. A.; Ladew, W. C.; Meaker, Jr., S. R.; Petry, R. M.; Seitz, R. N. (1957). *Investigation of Model Techniques — First Annual Report — 6 June 1956 — 1 July 1957 — A Study of Model Techniques for Communication Systems*. Cleveland, Ohio: Case Institute of Technology.
 - Knuth, D.E. (1977). “A Generalization of Dijkstra’s Algorithm”. *Information Processing Letters* 6 (1): 1–5. doi:10.1016/0020-0190(77)90002-3.
 - Ahuja, Ravindra K.; Mehlhorn, Kurt; Orlin, James B.; Tarjan, Robert E. (April 1990). “Faster Algorithms for the Shortest Path Problem”. *Journal of Association for Computing Machinery (ACM)* 37 (2): 213–223. doi:10.1145/77600.77615.
 - Raman, Rajeev (1997). “Recent results on the single-source shortest paths problem”. *SIGACT News* 28 (2): 81–87. doi:10.1145/261342.261352.
 - Thorup, Mikkel (2000). “On RAM priority Queues”. *SIAM Journal on Computing* 30 (1): 86–109. doi:10.1137/S0097539795288246.
 - Thorup, Mikkel (1999). “Undirected single-source shortest paths with positive integer weights in linear time”. *Journal of the ACM* 46 (3): 362–394. doi:10.1145/316542.316548.

40.9 References

- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). “Section 24.3: Dijkstra’s algorithm”. *Introduction to Algorithms* (Second ed.). MIT Press and McGraw-Hill. pp. 595–601. ISBN 0-262-03293-7.
- Fredman, Michael Lawrence; Tarjan, Robert E. (1984). *Fibonacci heaps and their uses in improved network optimization algorithms*. 25th Annual Symposium on Foundations of Computer Science. IEEE. pp. 338–346. doi:10.1109/SFCS.1984.715934.
- Fredman, Michael Lawrence; Tarjan, Robert E. (1987). “Fibonacci heaps and their uses in improved network optimization algorithms”. *Journal of the Association for Computing Machinery* 34 (3): 596–615. doi:10.1145/28869.28874.
- Zhan, F. Benjamin; Noon, Charles E. (February 1998). “Shortest Path Algorithms: An Evaluation Using Real Road Networks”. *Transportation Science* 32 (1): 65–73. doi:10.1287/trsc.32.1.65.

40.10 External links

- Applet by Carla Laffra of Pace University
- Dijkstra’s Algorithm Applet
- Shortest Path Problem: Dijkstra’s Algorithm
- Dijkstra’s Algorithm Simulation
- Oral history interview with Edsger W. Dijkstra, Charles Babbage Institute University of Minnesota, Minneapolis.
- Animation of Dijkstra’s algorithm

Chapter 41

Floyd–Warshall algorithm

“Floyd’s algorithm” redirects here. For cycle detection, see [Floyd’s cycle-finding algorithm](#). For computer graphics, see [Floyd–Steinberg dithering](#).

In computer science, the **Floyd–Warshall algorithm** (also known as **Floyd’s algorithm**, **Roy–Warshall algorithm**, **Roy–Floyd algorithm**, or the **WFI algorithm**) is a graph analysis [algorithm](#) for finding [shortest paths](#) in a [weighted graph](#) with positive or negative edge weights (but with no negative cycles, see below) and also for finding [transitive closure](#) of a relation R . A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between *all* pairs of vertices, though it does not return details of the paths themselves.

The Floyd–Warshall algorithm was published in its currently recognized form by [Robert Floyd](#) in 1962. However, it is essentially the same as algorithms previously published by [Bernard Roy](#) in 1959 and also by [Stephen Warshall](#) in 1962 for finding the transitive closure of a graph.^[1] The modern formulation of Warshall’s algorithm as three nested for-loops was first described by Peter Ingerman, also in 1962.

The algorithm is an example of [dynamic programming](#).

41.1 Algorithm

The Floyd–Warshall algorithm compares all possible paths through the graph between each pair of vertices. It is able to do this with $\Theta(|V|^3)$ comparisons in a graph. This is remarkable considering that there may be up to $\Omega(|V|^2)$ edges in the graph, and every combination of edges is tested. It does so by incrementally improving an estimate on the shortest path between two vertices, until the estimate is optimal.

Consider a graph G with vertices V numbered 1 through N . Further consider a function $\text{shortestPath}(i, j, k)$ that returns the shortest possible path from i to j using vertices only from the set $\{1, 2, \dots, k\}$ as intermediate points along the way. Now, given this function, our goal is to find the shortest path from each i to each j using only vertices 1 to $k + 1$.

For each of these pairs of vertices, the true shortest path could be either (1) a path that only uses vertices in the set $\{1, \dots, k\}$ or (2) a path that goes from i to $k + 1$ and then from $k + 1$ to j . We know that the best path from i to j that only uses vertices 1 through k is defined by $\text{shortestPath}(i, j, k)$, and it is clear that if there were a better path from i to $k + 1$ to j , then the length of this path would be the concatenation of the shortest path from i to $k + 1$ (using vertices in $\{1, \dots, k\}$) and the shortest path from $k + 1$ to j (also using vertices in $\{1, \dots, k\}$).

If $w(i, j)$ is the weight of the edge between vertices i and j , we can define $\text{shortestPath}(i, j, k + 1)$ in terms of the following recursive formula: the base case is

$$\text{shortestPath}(i, j, 0) = w(i, j)$$

and the recursive case is

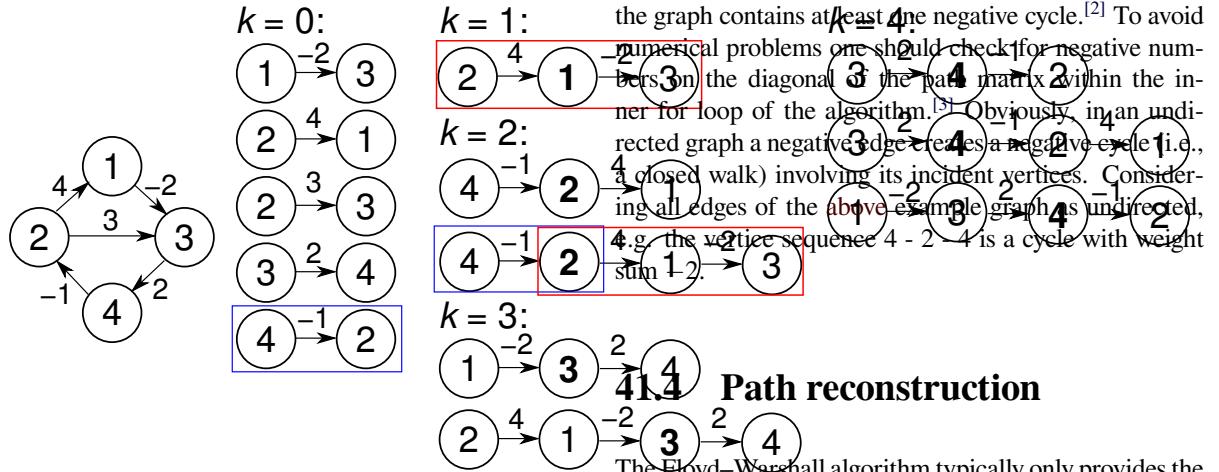
$$\text{shortestPath}(i, j, k+1) = \min(\text{shortestPath}(i, j, k), \text{shortestPath}(i, k+1, j))$$

This formula is the heart of the Floyd–Warshall algorithm. The algorithm works by first computing $\text{shortestPath}(i, j, k)$ for all (i, j) pairs for $k = 1$, then $k = 2$, etc. This process continues until $k = n$, and we have found the shortest path for all (i, j) pairs using any intermediate vertices. Pseudocode for this basic version follows:

```
1 let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
2 for each vertex v 3 dist[v][v] ← 0
4 for each edge (u,v) 5 dist[u][v] ← w(u,v) //the weight of the edge (u,v)
6 for k from 1 to |V| 7 for i from 1 to |V|
8 for j from 1 to |V| 9 if dist[i][j] > dist[i][k] + dist[k][j]
10 dist[i][j] ← dist[i][k] + dist[k][j] 11 end if
```

41.2 Example

The algorithm above is executed on the graph on the left below:



Prior to the first iteration of the outer loop, labeled $k=0$ above, the only known paths correspond to the single edges in the graph. At $k=1$, paths that go through the vertex 1 are found: in particular, the path $2 \rightarrow 1 \rightarrow 3$ is found, replacing the path $2 \rightarrow 3$ which has fewer edges but is longer. At $k=2$, paths going through the vertices $\{1,2\}$ are found. The red and blue boxes show how the path $4 \rightarrow 2 \rightarrow 1 \rightarrow 3$ is assembled from the two known paths $4 \rightarrow 2$ and $2 \rightarrow 1 \rightarrow 3$ encountered in previous iterations, with 2 in the intersection. The path $4 \rightarrow 2 \rightarrow 3$ is not considered, because $2 \rightarrow 1 \rightarrow 3$ is the shortest path encountered so far from 2 to 3. At $k=3$, paths going through the vertices $\{1,2,3\}$ are found. Finally, at $k=4$, all shortest paths are found.

41.3 Behavior with negative cycles

A negative cycle is a cycle whose edges sum to a negative value. There is no shortest path between any pair of vertices i, j which form part of a negative cycle, because path-lengths from i to j can be arbitrarily small (negative). For numerically meaningful output, the Floyd–Warshall algorithm assumes that there are no negative cycles. Nevertheless, if there are negative cycles, the Floyd–Warshall algorithm can be used to detect them. The intuition is as follows:

- The Floyd–Warshall algorithm iteratively revises path lengths between all pairs of vertices (i, j) , including where $i = j$;
- Initially, the length of the path (i, i) is zero;
- A path $\{(i, k), (k, i)\}$ can only improve upon this if it has length less than zero, i.e. denotes a negative cycle;
- Thus, after the algorithm, (i, i) will be negative if there exists a negative-length path from i back to i .

Hence, to detect negative cycles using the Floyd–Warshall algorithm, one can inspect the diagonal of the path matrix, and the presence of a negative number indicates that

the graph contains at least one negative cycle.^[2] To avoid numerical problems one should check for negative numbers on the diagonal of the path matrix within the inner loop of the algorithm.^[3] Obviously, in an undirected graph a negative edge creates a negative cycle (i.e., a closed walk) involving its incident vertices. Considering all edges of the above example graph as undirected, e.g. the vertex sequence $4 \rightarrow 2 \rightarrow 4$ is a cycle with weight sum $-1 + 1 = 0$.

41.4 Path reconstruction

The Floyd–Warshall algorithm typically only provides the lengths of the paths between all pairs of vertices. With simple modifications, it is possible to create a method to reconstruct the actual path between any two endpoint vertices. While one may be inclined to store the actual path from each vertex to each other vertex, this is not necessary, and in fact, is very costly in terms of memory. Instead, the Shortest-path tree can be calculated for each node in $\Theta(|E|)$ time using $\Theta(|V|)$ memory to store each tree which allows us to efficiently reconstruct a path from any two connected vertices.

```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
let next be a |V| × |V| array of vertex indices initialized to null
procedure FloydWarshallWithPathReconstruction()
    for each edge (u,v) dist[u][v] ← w(u,v) //the weight of the edge (u,v)
    next[u][v] ← v
    for k from 1 to |V| // standard Floyd-Warshall implementation
        for i from 1 to |V| for j from 1 to |V|
            if dist[i][k] + dist[k][j] < dist[i][j] then
                dist[i][j] ← dist[i][k] + dist[k][j]
                next[i][j] ← next[i][k]
procedure Path(u, v)
    if next[u][v] = null then return []
    path = [u]
    while u ≠ v u ← next[u][v]
    path.append(u)
    return path
```

41.5 Analysis

Let n be $|V|$, the number of vertices. To find all n^2 of $\text{shortestPath}(i,j,k)$ (for all i and j) from those of $\text{shortestPath}(i,j,k-1)$ requires $2n^2$ operations. Since we begin with $\text{shortestPath}(i,j,0) = \text{edgeCost}(i,j)$ and compute the sequence of n matrices $\text{shortestPath}(i,j,1)$, $\text{shortestPath}(i,j,2)$, ..., $\text{shortestPath}(i,j,n)$, the total number of operations used is $n \cdot 2n^2 = 2n^3$. Therefore, the complexity of the algorithm is $\Theta(n^3)$.

41.6 Applications and generalizations

The Floyd–Warshall algorithm can be used to solve the following problems, among others:

- Shortest paths in directed graphs (Floyd's algorithm).
- Transitive closure of directed graphs (Warshall's algorithm). In Warshall's original formulation of the algorithm, the graph is unweighted and represented by a Boolean adjacency matrix. Then the addition operation is replaced by logical conjunction (AND) and the minimum operation by logical disjunction (OR).
- Finding a regular expression denoting the regular language accepted by a finite automaton (Kleene's algorithm, a closely related generalization of the Floyd–Warshall algorithm)^[4]
- Inversion of real matrices (Gauss–Jordan algorithm) [5]
- Optimal routing. In this application one is interested in finding the path with the maximum flow between two vertices. This means that, rather than taking minima as in the pseudocode above, one instead takes maxima. The edge weights represent fixed constraints on flow. Path weights represent bottlenecks; so the addition operation above is replaced by the minimum operation.
- Fast computation of Pathfinder networks.
- Widest paths/Maximum bandwidth paths
- Computing canonical form of difference bound matrices (DBMs)

41.7 Implementations

Implementations are available for many programming languages.

- For C++, in the boost::graph library
- For C#, at QuickGraph
- For Java, in the Apache Commons Graph library
- For MATLAB, in the Matlab_bgl package
- For Perl, in the Graph module
- For Python, in the NetworkX library
- For R, in package e1071

41.8 See also

- Dijkstra's algorithm, an algorithm for finding single-source shortest paths in a more restrictive class of inputs, graphs in which all edge weights are non-negative

- Johnson's algorithm, an algorithm for solving the same problem as the Floyd–Warshall algorithm, all pairs shortest paths in graphs with some edge weights negative. Compared to the Floyd–Warshall algorithm, Johnson's algorithm is more efficient for sparse graphs.

41.9 References

- [1] Weisstein, Eric. “Floyd–Warshall Algorithm”. *Wolfram MathWorld*. Retrieved 13 November 2009.
- [2] Dorit Hochbaum (2014). “Section 8.9: Floyd–Warshall algorithm for all pairs shortest paths” (PDF). *Lecture Notes for IEOR 266: Graph Algorithms and Network Flows*. Department of Industrial Engineering and Operations Research, University of California, Berkeley.
- [3] Stefan Hougardy (April 2010). “The Floyd–Warshall algorithm on graphs with negative cycles”. *Information Processing Letters* **110** (8–9): 279–281. doi:10.1016/j.ipl.2010.02.001.
- [4] Gross, Jonathan L.; Yellen, Jay (2003), *Handbook of Graph Theory*, Discrete Mathematics and Its Applications, CRC Press, p. 65, ISBN 9780203490204.
- [5] Penaloza, Rafael. “Algebraic Structures for Transitive Closure”.
- Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L. (1990). *Introduction to Algorithms* (1st ed.). MIT Press and McGraw-Hill. ISBN 0-262-03141-8.
 - Section 26.2, “The Floyd–Warshall algorithm”, pp. 558–565;
 - Section 26.4, “A general framework for solving path problems in directed graphs”, pp. 570–576.
- Floyd, Robert W. (June 1962). “Algorithm 97: Shortest Path”. *Communications of the ACM* **5** (6): 345. doi:10.1145/367766.368168.
- Ingerman, Peter Z. (November 1962). “Algorithm 141: Path Matrix”. *Communications of the ACM* **5** (11): 556. doi:10.1145/368996.369016.
- Kleene, S. C. (1956). “Representation of events in nerve nets and finite automata”. In C. E. Shannon and J. McCarthy. *Automata Studies*. Princeton University Press. pp. 3–42.
- Warshall, Stephen (January 1962). “A theorem on Boolean matrices”. *Journal of the ACM* **9** (1): 11–12. doi:10.1145/321105.321107.
- Kenneth H. Rosen (2003). *Discrete Mathematics and Its Applications, 5th Edition*. Addison Wesley. ISBN 0-07-119881-4.
- Roy, Bernard (1959). “Transitivité et connexité.”. *C. R. Acad. Sci. Paris* **249**: 216–218.

41.10 External links

- Interactive animation of the Floyd–Warshall algorithm
- The Floyd–Warshall algorithm in C#, as part of QuickGraph
- Visualization of Floyd’s algorithm

Chapter 42

Johnson's algorithm

For the scheduling algorithm of the same name, see Job Shop Scheduling.

Johnson's algorithm is a way to find the shortest paths between all pairs of vertices in a sparse, edge weighted, directed graph. It allows some of the edge weights to be negative numbers, but no negative-weight cycles may exist. It works by using the Bellman–Ford algorithm to compute a transformation of the input graph that removes all negative weights, allowing Dijkstra's algorithm to be used on the transformed graph.^{[1][2]} It is named after Donald B. Johnson, who first published the technique in 1977.^[3]

A similar reweighting technique is also used in Suurballe's algorithm for finding two disjoint paths of minimum total length between the same two vertices in a graph with non-negative edge weights.^[4]

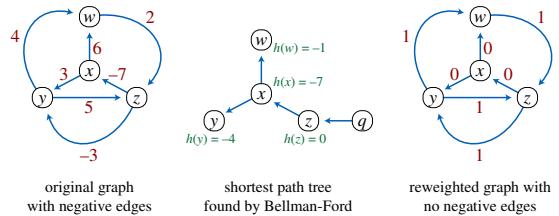
42.1 Algorithm description

Johnson's algorithm consists of the following steps:^{[1][2]}

1. First, a new node q is added to the graph, connected by zero-weight edges to each of the other nodes.
2. Second, the Bellman–Ford algorithm is used, starting from the new vertex q , to find for each vertex v the minimum weight $h(v)$ of a path from q to v . If this step detects a negative cycle, the algorithm is terminated.
3. Next the edges of the original graph are reweighted using the values computed by the Bellman–Ford algorithm: an edge from u to v , having length $w(u, v)$, is given the new length $w(u, v) + h(u) - h(v)$.
4. Finally, q is removed, and Dijkstra's algorithm is used to find the shortest paths from each node s to every other vertex in the reweighted graph.

42.2 Example

The first three stages of Johnson's algorithm are depicted in the illustration below.



The graph on the left of the illustration has two negative edges, but no negative cycles. At the center is shown the new vertex q , a shortest path tree as computed by the Bellman–Ford algorithm with q as starting vertex, and the values $h(v)$ computed at each other node as the length of the shortest path from q to that node. Note that these values are all non-positive, because q has a length-zero edge to each vertex and the shortest path can be no longer than that edge. On the right is shown the reweighted graph, formed by replacing each edge weight $w(u, v)$ by $w(u, v) + h(u) - h(v)$. In this reweighted graph, all edge weights are non-negative, but the shortest path between any two nodes uses the same sequence of edges as the shortest path between the same two nodes in the original graph. The algorithm concludes by applying Dijkstra's algorithm to each of the four starting nodes in the reweighted graph.

42.3 Correctness

In the reweighted graph, all paths between a pair s and t of nodes have the same quantity $h(s) - h(t)$ added to them. The previous statement can be proven as follows: Let p be an s - t path. Its weight W in the reweighted graph is given by the following expression:

$$(w(s, p_1) + h(s) - h(p_1)) + (w(p_1, p_2) + h(p_1) - h(p_2)) + \dots + (w(p_n, t) + h(p_n) - h(t))$$

Every $+h(p_i)$ is cancelled by $-h(p_i)$ in the previous bracketed expression; therefore, we are left with the fol-

lowing expression for W :

42.6 External links

- Boost: All Pairs Shortest Paths

$$(w(s, p1) + w(p1, p2) + \dots + w(p_n, t)) + h(s) - h(t)$$

The bracketed expression is the weight of p in the original weighting.

Since the reweighting adds the same amount to the weight of every s-t path, a path is a shortest path in the original weighting if and only if it is a shortest path after reweighting. The weight of edges that belong to a shortest path from q to any node is zero, and therefore the lengths of the shortest paths from q to every node become zero in the reweighted graph; however, they still remain shortest paths. Therefore, there can be no negative edges: if edge uv had a negative weight after the reweighting, then the zero-length path from q to u together with this edge would form a negative-length path from q to v , contradicting the fact that all vertices have zero distance from q . The non-existence of negative edges ensures the optimality of the paths found by Dijkstra's algorithm. The distances in the original graph may be calculated from the distances calculated by Dijkstra's algorithm in the reweighted graph by reversing the reweighting transformation.^[1]

42.4 Analysis

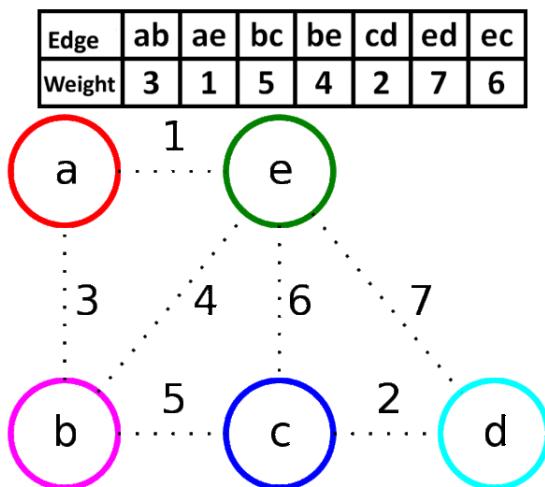
The time complexity of this algorithm, using Fibonacci heaps in the implementation of Dijkstra's algorithm, is $O(V^2 \log V + VE)$: the algorithm uses $O(VE)$ time for the Bellman–Ford stage of the algorithm, and $O(V \log V + E)$ for each of V instantiations of Dijkstra's algorithm. Thus, when the graph is sparse, the total time can be faster than the Floyd–Warshall algorithm, which solves the same problem in time $O(V^3)$.^[1]

42.5 References

- [1] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001), *Introduction to Algorithms*, MIT Press and McGraw-Hill, ISBN 978-0-262-03293-3. Section 25.3, “Johnson’s algorithm for sparse graphs”, pp. 636–640.
- [2] Black, Paul E. (2004), “Johnson’s Algorithm”, *Dictionary of Algorithms and Data Structures*, National Institute of Standards and Technology.
- [3] Johnson, Donald B. (1977), “Efficient algorithms for shortest paths in sparse networks”, *Journal of the ACM* **24** (1): 1–13, doi:10.1145/321992.321993.
- [4] Suurballe, J. W. (1974), “Disjoint paths in a network”, *Networks* **14** (2): 125–145, doi:10.1002/net.3230040204.

Chapter 43

Kruskal's algorithm



Visualization of Kruskal's algorithm

Kruskal's algorithm is a minimum-spanning-tree algorithm where the algorithm finds an edge of the least possible weight that connects any two trees in the forest.^[1] It is a greedy algorithm in graph theory as it finds a minimum spanning tree for a connected weighted graph at each step.^[1] This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a *minimum spanning forest* (a minimum spanning tree for each connected component).

This algorithm first appeared in *Proceedings of the American Mathematical Society*, pp. 48–50 in 1956, and was written by Joseph Kruskal.^[2]

Other algorithms for this problem include Prim's algorithm, Reverse-delete algorithm, and Borůvka's algorithm.

43.1 Description

- create a forest F (a set of trees), where each vertex in the graph is a separate tree
- create a set S containing all the edges in the graph

- while S is nonempty and F is not yet spanning
 - remove an edge with minimum weight from S
 - if that edge connects two different trees, then add it to the forest F , combining two trees into a single tree

At the termination of the algorithm, the forest forms a minimum spanning forest of the graph. If the graph is connected, the forest has a single component and forms a minimum spanning tree.

43.2 Pseudocode

The following code is implemented with disjoint-set data structure:

```
KRUSKAL(G): 1 A =  $\emptyset$  2 foreach v  $\in$  G.V: 3 MAKE-SET(v) 4 foreach (u, v) ordered by weight(u, v), increasing: 5 if FIND-SET(u)  $\neq$  FIND-SET(v): 6 A = A  $\cup$  {(u, v)} 7 UNION(u, v) 8 return A
```

43.3 Complexity

Where E is the number of edges in the graph and V is the number of vertices, Kruskal's algorithm can be shown to run in $O(E \log E)$ time, or equivalently, $O(E \log V)$ time, all with simple data structures. These running times are equivalent because:

- E is at most V^2 and $\log V^2 = 2 \log V$ is $O(\log V)$.
- Each isolated vertex is a separate component of the minimum spanning forest. If we ignore isolated vertices we obtain $V \leq E+1$, so $\log V$ is $O(\log E)$.

We can achieve this bound as follows: first sort the edges by weight using a comparison sort in $O(E \log E)$ time; this allows the step “remove an edge with minimum weight from S ” to operate in constant time. Next, we use a disjoint-set data structure (Union&Find) to keep track of which vertices are in which components. We need to perform $O(V)$ operations, as in each iteration we connect a

vertex to the spanning tree, two 'find' operations and possibly one union for each edge. Even a simple disjoint-set data structure such as disjoint-set forests with union by rank can perform $O(V)$ operations in $O(V \log V)$ time. Thus the total time is $O(E \log E) = O(E \log V)$.

Provided that the edges are either already sorted or can be sorted in linear time (for example with [counting sort](#) or [radix sort](#)), the algorithm can use more sophisticated disjoint-set data structure to run in $O(E \alpha(V))$ time, where α is the extremely slowly growing inverse of the single-valued [Ackermann function](#).

43.4 Example

43.5 Proof of correctness

The proof consists of two parts. First, it is proved that the algorithm produces a [spanning tree](#). Second, it is proved that the constructed spanning tree is of minimal weight.

43.5.1 Spanning tree

Let P be a connected, weighted graph and let Y be the subgraph of P produced by the algorithm. Y cannot have a cycle, being within one subtree and not between two different trees. Y cannot be disconnected, since the first encountered edge that joins two components of Y would have been added by the algorithm. Thus, Y is a spanning tree of P .

43.5.2 Minimality

We show that the following proposition \mathbf{P} is true by induction: If F is the set of edges chosen at any stage of the algorithm, then there is some minimum spanning tree that contains F .

- Clearly \mathbf{P} is true at the beginning, when F is empty: any minimum spanning tree will do, and there exists one because a weighted connected graph always has a minimum spanning tree.
- Now assume \mathbf{P} is true for some non-final edge set F and let T be a minimum spanning tree that contains F . If the next chosen edge e is also in T , then \mathbf{P} is true for $F + e$. Otherwise, $T + e$ has a cycle C and there is another edge f that is in C but not F . (If there were no such edge f , then e could not have been added to F , since doing so would have created the cycle C .) Then $T - f + e$ is a tree, and it has the same weight as T , since T has minimum weight and the weight of f cannot be less than the weight of e , otherwise the algorithm would have chosen f instead of e . So $T - f + e$ is a minimum spanning tree containing $F + e$ and again \mathbf{P} holds.

- Therefore, by the principle of induction, \mathbf{P} holds when F has become a spanning tree, which is only possible if F is a minimum spanning tree itself.

43.6 See also

- Dijkstra's algorithm
- Prim's algorithm
- Borůvka's algorithm
- Reverse-delete algorithm
- Single-linkage clustering

43.7 References

- [1] Cormen, Thomas; Charles E Leiserson, Ronald L Rivest, Clifford Stein (2009). *Introduction To Algorithms* (Third ed.). MIT Press. p. 631. ISBN 0262258102.
- [2] Kruskal, J. B. (1956). "On the shortest spanning subtree of a graph and the traveling salesman problem". *Proceedings of the American Mathematical Society* 7: 48–50. doi:10.1090/S0002-9939-1956-0078686-7. JSTOR 2033241.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 23.2: The algorithms of Kruskal and Prim, pp. 567–574.
- Michael T. Goodrich and Roberto Tamassia. *Data Structures and Algorithms in Java*, Fourth Edition. John Wiley & Sons, Inc., 2006. ISBN 0-471-73884-0. Section 13.7.1: Kruskal's Algorithm, pp. 632..

43.8 External links

- Kruskal's algorithm explanation and example with c implementation
- Download the example minimum spanning tree data.
- Animation of Kruskal's algorithm (Requires Java plugin)
- download kruskal algorithm Implement with C++ and java(graphical)(Requires java 7+)
- C# Implementation
- Open source java graph library with implementation of Kruskal's algorithm
- Auto-generated PowerPoint Slides for Teaching and Learning

Chapter 44

Dinic's algorithm

Dinitz's algorithm is a strongly polynomial algorithm for computing the maximum flow in a flow network, conceived in 1970 by Israeli (formerly Soviet) computer scientist Yefim (Chaim) A. Dinitz.^[1] The algorithm runs in $O(V^2E)$ time and is similar to the Edmonds–Karp algorithm, which runs in $O(VE^2)$ time, in that it uses shortest augmenting paths. The introduction of the concepts of the *level graph* and *blocking flow* enable Dinic's algorithm to achieve its performance.

44.1 History

Yefim Dinitz invented this algorithm in response to a pre-class exercise in Adel'son-Vel'sky's (co-inventor of AVL trees) Algorithm class. At the time he was not aware of the basic facts regarding Ford–Fulkerson algorithm.^[2]

Dinitz mentions inventing his algorithm in January, 1969 which was published in 1970 in journal Doklady. In 1974, Shimon Even and (his then Ph.D. student) Alon Itai at the Technion, Haifa were very curious and intrigued by the Dinitz's algorithm as well as Alexander Karzanov's idea of blocking flow. However it was hard to decipher these two papers for them, each being limited to four pages to meet the restrictions of journal Doklady. However Even did not give up and after three days of effort managed to understand both papers except for the layered network maintenance issue. Over the next couple of years, Even gave lectures on “Dinic's algorithm” mispronouncing the name of the author while popularizing it. Even and Itai also contributed to this algorithm by combining BFS and DFS which is the current version of algorithm^[3]

For about 10 years of time after Ford–Fulkerson algorithm was invented, it was unknown if it can be made to terminate in polynomial time in the generic case of irrational edge capacities. This caused lack of any known polynomial time algorithm that solved max flow problem in generic case. Dinitz algorithm and the Edmonds–Karp algorithm, which was published in 1972, independently showed that in the Ford–Fulkerson algorithm, if each augmenting path is the shortest one, the length of the augmenting paths is non-decreasing and it always terminated.

44.2 Definition

Let $G = ((V, E), c, s, t)$ be a network with $c(u, v)$ and $f(u, v)$ the capacity and the flow of the edge (u, v) respectively.

The **residual capacity** is a mapping $c_f: V \times V \rightarrow R^+$ defined as,

1. if $(u, v) \in E$,

$$c_f(u, v) = c(u, v) - f(u, v)$$

$$c_f(v, u) = f(u, v)$$

2. $c_f(u, v) = 0$ otherwise.

The **residual graph** is the graph $G_f = ((V, E_f), c_f|_{E_f}, s, t)$, where

$$E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$$

An **augmenting path** is an $s - t$ path in the residual graph G_f .

Define $\text{dist}(v)$ to be the length of the shortest path from s to v in G_f . Then the **level graph** of G_f is the graph $G_L = (V, E_L, c_f|_{E_L}, s, t)$, where

$$E_L = \{(u, v) \in E_f : \text{dist}(v) = \text{dist}(u) + 1\}$$

A **blocking flow** is an $s - t$ flow f such that the graph $G' = (V, E'_L, s, t)$ with $E'_L = \{(u, v) : f(u, v) < c_f|_{E_L}(u, v)\}$ contains no $s - t$ path.

44.3 Algorithm

Dinic's Algorithm

Input: A network $G = ((V, E), c, s, t)$.

Output: An $s - t$ flow f of maximum value.

1. Set $f(e) = 0$ for each $e \in E$.

2. Construct G_L from G_f of G . If $\text{dist}(t) = \infty$, stop and output f .
3. Find a blocking flow f' in G_L .
4. Augment flow f by f' and go back to step 2.

44.4 Analysis

It can be shown that the number of edges in each blocking flow increases by at least 1 each time and thus there are at most $n - 1$ blocking flows in the algorithm, where n is the number of vertices in the network. The level graph G_L can be constructed by Breadth-first search in $O(E)$ time and a blocking flow in each level graph can be found in $O(VE)$ time. Hence, the running time of Dinic's algorithm is $O(V^2E)$.

Using a data structure called **dynamic trees**, the running time of finding a blocking flow in each phase can be reduced to $O(E \log V)$ and therefore the running time of Dinic's algorithm can be improved to $O(VE \log V)$.

44.4.1 Special cases

In networks with unit capacities, a much stronger time bound holds. Each blocking flow can be found in $O(E)$ time, and it can be shown that the number of phases does not exceed $O(\sqrt{E})$ and $O(V^{2/3})$. Thus the algorithm runs in $O(\min\{V^{2/3}, E^{1/2}\}E)$ time.

In networks arising during the solution of **bipartite matching** problem, the number of phases is bounded by $O(\sqrt{V})$, therefore leading to the $O(\sqrt{V}E)$ time bound. The resulting algorithm is also known as **Hopcroft–Karp algorithm**. More generally, this bound holds for any **unit network** — a network in which each vertex, except for source and sink, either has a single entering edge of capacity one, or a single outgoing edge of capacity one, and all other capacities are arbitrary integers.^[4]

44.5 Example

The following is a simulation of the Dinic's algorithm. In the level graph G_L , the vertices with labels in red are the values $\text{dist}(v)$. The paths in blue form a blocking flow.

44.6 See also

- Ford–Fulkerson algorithm
- Maximum flow problem

44.7 Notes

- [1] Yefim Dinitz (1970). “Algorithm for solution of a problem of maximum flow in a network with power estimation”. *Doklady Akademii nauk SSSR* **11**: 1277–1280.
- [2] Ilan Kadar, Sivan Albagli (2009), *Dinitz's algorithm for finding a maximum flow in a network*
- [3] Yefim Dinitz, *Dinitz's Algorithm: The Original Version and Even's Version*
- [4] Tarjan 1983, p. 102.

44.8 References

- Yefim Dinitz (2006). “Dinitz' Algorithm: The Original Version and Even's Version”. In Oded Goldreich, Arnold L. Rosenberg, and Alan L. Selman. *Theoretical Computer Science: Essays in Memory of Shimon Even*. Springer. pp. 218–240. ISBN 978-3-540-32880-3.
- Tarjan, R. E. (1983). *Data structures and network algorithms*.
- B. H. Korte, Jens Vygen (2008). “8.4 Blocking Flows and Fujishige's Algorithm”. *Combinatorial Optimization: Theory and Algorithms (Algorithms and Combinatorics, 21)*. Springer Berlin Heidelberg. pp. 174–176. ISBN 978-3-540-71844-4.

Chapter 45

Edmonds–Karp algorithm

In computer science, the **Edmonds–Karp algorithm** is an implementation of the Ford–Fulkerson method for computing the maximum flow in a flow network in $O(V E^2)$ time. The algorithm was first published by Yefim (Chaim) Dinic in 1970^[1] and independently published by Jack Edmonds and Richard Karp in 1972.^[2] Dinic's algorithm includes additional techniques that reduce the running time to $O(V^2E)$.

45.1 Algorithm

The algorithm is identical to the Ford–Fulkerson algorithm, except that the search order when finding the augmenting path is defined. The path found must be a shortest path that has available capacity. This can be found by a breadth-first search, as we let edges have unit length. The running time of $O(V E^2)$ is found by showing that each augmenting path can be found in $O(E)$ time, that every time at least one of the E edges becomes saturated (an edge which has the maximum possible flow), that the distance from the saturated edge to the source along the augmenting path must be longer than last time it was saturated, and that the length is at most V . Another property of this algorithm is that the length of the shortest augmenting path increases monotonically. There is an accessible proof in *Introduction to Algorithms*.^[3]

45.2 Pseudocode

For a more high level description, see Ford–Fulkerson algorithm.

algorithm EdmondsKarp **input:** $C[1..n, 1..n]$ (*Capacity matrix*) $E[1..n, 1..?]$ (*Neighbour lists*) s (*Source*) t (*Sink*) **output:** f (*Value of maximum flow*) F (*A matrix giving a legal flow with the maximum value*) $f := 0$ (*Initial flow is zero*) $F := \text{array}(1..n, 1..n)$ (*Residual capacity from u to v is $C[u,v] - F[u,v]$*) **forever** $m, P := \text{BreadthFirstSearch}(C, E, s, t, F)$ **if** $m = 0$ **break** $f := f + m$ (*Backtrack search, and write flow*) $v := t$ **while** $v \neq s$ $u := P[v]$ $F[u,v] := F[u,v] + m$ $F[v,u] := F[v,u] - m$ $v := u$ **return** (f, F) **algorithm**

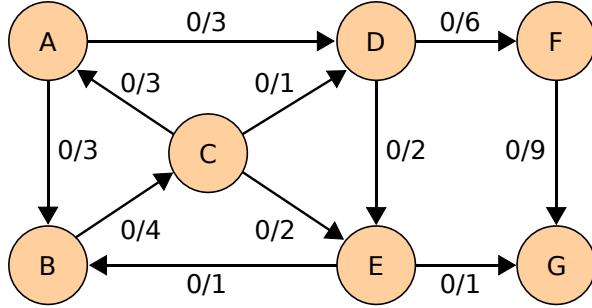
BreadthFirstSearch **input:** C, E, s, t, F **output:** $M[t]$ (*Capacity of path found*) P (*Parent table*) $P := \text{array}(1..n)$ **for** $u \in 1..n$ $P[u] := -1$ $P[s] := -2$ (*make sure source is not rediscovered*) $M := \text{array}(1..n)$ (*Capacity of found path to node*) $M[s] := \infty$ $Q := \text{queue}()$ $Q.offer(s)$ **while** $Q.size() > 0$ $u := Q.poll()$ **for** $v \in E[u]$ (*If there is available capacity, and v is not seen before in search*) **if** $C[u,v] - F[u,v] > 0$ **and** $P[v] = -1$ $P[v] := u$ $M[v] := \min(M[u], C[u,v] - F[u,v])$ **if** $v \neq t$ $Q.offer(v)$ **else return** $M[t], P$ **return** $0, P$

EdmondsKarp pseudo code using Adjacency nodes.

algorithm EdmondsKarp **input:** graph (*Graph with list of Adjacency nodes with capacities,flow,reverse and destinations*) s (*Source*) t (*Sink*) **output:** flow (*Value of maximum flow*) $flow := 0$ (*Initial flow to zero*) $q := \text{array}(1..n)$ (*Initialize q to graph length*) **while** true $qt := 0$ (*Variable to iterate over all the corresponding edges for a source*) $q[qt++] := s$ (*initialize source array*) $pred := \text{array}(q.length)$ (*Initialize predecessor List with the graph length*) **for** $qh=0; qh < qt \ \&\& \ pred[t] == \text{null}$ $cur := q[qh]$ **for** (*graph[cur]*) (*Iterate over list of Edges*) $Edge[] e := \text{graph}[cur]$ (*Each edge should be associated with Capacity*) **if** $pred[e.t] == \text{null} \ \&\& \ e.cap > e.f$ $pred[e.t] := e$ $q[qt++] := e.t$ **if** $pred[t] == \text{null}$ **break** **int** $df := \text{MAX VALUE}$ (*Initialize to max integer value*) **for** $u = t; u != s; u = pred[u].s$ $df := \min(df, pred[u].cap - pred[u].f)$ **for** $u = t; u != s; u = pred[u].s$ $pred[u].f := pred[u].f + df$ $pEdge := \text{array}(\text{PredEdge})$ $pEdge := \text{graph}[pred[u].t]$ $pEdge[pred[u].rev].f := pEdge[pred[u].rev].f - df$; $flow := flow + df$ **return** $flow$

45.3 Example

Given a network of seven nodes, source A, sink G, and capacities as shown below:



In the pairs f/c written on the edges, f is the current flow, and c is the capacity. The residual capacity from u to v is $c_f(u, v) = c(u, v) - f(u, v)$, the total capacity, minus the flow that is already used. If the net flow from u to v is negative, it *contributes* to the residual capacity.

Notice how the length of the augmenting path found by the algorithm (in red) never decreases. The paths found are the shortest possible. The flow found is equal to the capacity across the **minimum cut** in the graph separating the source and the sink. There is only one minimal cut in this graph, partitioning the nodes into the sets $\{A, B, C, E\}$ and $\{D, F, G\}$, with the capacity

$$c(A, D) + c(C, D) + c(E, G) = 3 + 1 + 1 = 5.$$

45.4 Notes

- [1] Dinic, E. A. (1970). “Algorithm for solution of a problem of maximum flow in a network with power estimation”. *Soviet Math. Doklady* (Doklady) **11**: 1277–1280.
- [2] Edmonds, Jack; Karp, Richard M. (1972). “Theoretical improvements in algorithmic efficiency for network flow problems”. *Journal of the ACM* (Association for Computing Machinery) **19** (2): 248–264. doi:10.1145/321694.321699.
- [3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein (2009). “26.2”. *Introduction to Algorithms* (third ed.). MIT Press. pp. 727–730. ISBN 978-0-262-03384-8.

45.5 References

1. Algorithms and Complexity (see pages 63–69).
[http://www.cis.upenn.edu/~{ }wilf/AlgComp3.html](http://www.cis.upenn.edu/~wilf/AlgComp3.html)

Chapter 46

Ford–Fulkerson algorithm

The **Ford–Fulkerson method** or **Ford–Fulkerson algorithm (FFA)** is an algorithm which computes the maximum flow in a flow network. It is called a “method” instead of an “algorithm” as the approach to finding augmenting paths in a residual graph is not fully specified^[1] or it is specified in several implementations with different running times.^[2] It was published in 1956 by L. R. Ford, Jr. and D. R. Fulkerson.^[3] The name “Ford–Fulkerson” is often also used for the **Edmonds–Karp algorithm**, which is a specialization of Ford–Fulkerson.

The idea behind the algorithm is as follows: As long as there is a path from the source (start node) to the sink (end node), with available capacity on all edges in the path, we send flow along one of these paths. Then we find another path, and so on. A path with available capacity is called an augmenting path.

46.1 Algorithm

Let $G(V, E)$ be a graph, and for each edge from u to v , let $c(u, v)$ be the capacity and $f(u, v)$ be the flow. We want to find the maximum flow from the source s to the sink t . After every step in the algorithm the following is maintained:

This means that the flow through the network is a *legal flow* after each round in the algorithm. We define the **residual network** $G_f(V, E_f)$ to be the network with capacity $c_f(u, v) = c(u, v) - f(u, v)$ and no flow. Notice that it can happen that a flow from v to u is allowed in the residual network, though disallowed in the original network: if $f(u, v) > 0$ and $c(v, u) = 0$ then $c_f(v, u) = c(v, u) - f(v, u) = f(u, v) > 0$.

Algorithm Ford–Fulkerson

Inputs Given a Network $G = (V, E)$ with flow capacity c , a source node s , and a sink node t

Output Compute a flow f from s to t of maximum value

1. $f(u, v) \leftarrow 0$ for all edges (u, v)

2. While there is a path p from s to t in G_f , such that $c_f(u, v) > 0$ for all edges $(u, v) \in p$:
 - (a) Find $c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$
 - (b) For each edge $(u, v) \in p$
 - i. $f(u, v) \leftarrow f(u, v) + c_f(p)$
(Send flow along the path)
 - ii. $f(v, u) \leftarrow f(v, u) - c_f(p)$
(The flow might be “returned” later)

The path in step 2 can be found with for example a breadth-first search or a depth-first search in $G_f(V, E_f)$. If you use the former, the algorithm is called **Edmonds–Karp**.

When no more paths in step 2 can be found, s will not be able to reach t in the residual network. If S is the set of nodes reachable by s in the residual network, then the total capacity in the original network of edges from S to the remainder of V is on the one hand equal to the total flow we found from s to t , and on the other hand serves as an upper bound for all such flows. This proves that the flow we found is maximal. See also **Max-flow Min-cut theorem**.

If the graph $G(V, E)$ has multiple sources and sinks, we act as follows: Suppose that $T = \{t | t \text{ sink a is}\}$ and $S = \{s | s \text{ source a is}\}$. Add a new source s^* with an edge (s^*, s) from s^* to every node $s \in S$, with capacity $c(s^*, s) = d_s$ ($d_s = \sum_{(s,u) \in E} c(s, u)$). And add a new sink t^* with an edge (t, t^*) from every node $t \in T$ to t^* , with capacity $c(t, t^*) = d_t$ ($d_t = \sum_{(v,t) \in E} c(v, t)$). Then apply the Ford–Fulkerson algorithm.

Also, if a node u has capacity constraint d_u , we replace this node with two nodes u_{in}, u_{out} , and an edge (u_{in}, u_{out}) , with capacity $c(u_{in}, u_{out}) = d_u$. Then apply the Ford–Fulkerson algorithm.

46.2 Complexity

By adding the flow augmenting path to the flow already established in the graph, the maximum flow will be reached

when no more flow augmenting paths can be found in the graph. However, there is no certainty that this situation will ever be reached, so the best that can be guaranteed is that the answer will be correct if the algorithm terminates. In the case that the algorithm runs forever, the flow might not even converge towards the maximum flow. However, this situation only occurs with irrational flow values. When the capacities are integers, the runtime of Ford–Fulkerson is bounded by $O(Ef)$ (see big O notation), where E is the number of edges in the graph and f is the maximum flow in the graph. This is because each augmenting path can be found in $O(E)$ time and increases the flow by an integer amount which is at least 1.

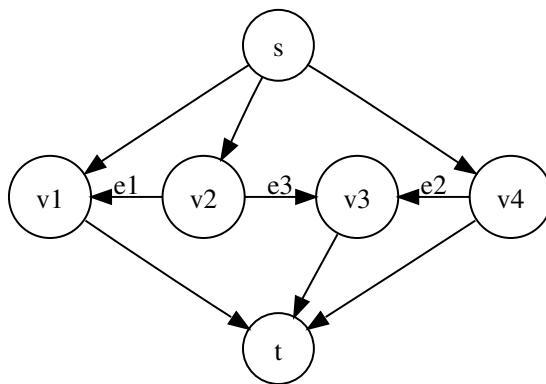
A variation of the Ford–Fulkerson algorithm with guaranteed termination and a runtime independent of the maximum flow value is the Edmonds–Karp algorithm, which runs in $O(VE^2)$ time.

46.3 Integral example

The following example shows the first steps of Ford–Fulkerson in a flow network with 4 nodes, source A and sink D . This example shows the worst-case behaviour of the algorithm. In each step, only a flow of 1 is sent across the network. If breadth-first-search were used instead, only two steps would be needed.

Notice how flow is “pushed back” from C to B when finding the path A, C, B, D .

46.4 Non-terminating example



Consider the flow network shown on the right, with source s , sink t , capacities of edges e_1 , e_2 and e_3 respectively 1 , $r = (\sqrt{5} - 1)/2$ and 1 and the capacity of all other edges some integer $M \geq 2$. The constant r was chosen so, that $r^2 = 1 - r$. We use augmenting paths according to the following table, where $p_1 = \{s, v_4, v_3, v_2, v_1, t\}$, $p_2 = \{s, v_2, v_3, v_4, t\}$ and $p_3 = \{s, v_1, v_2, v_3, t\}$.

Note that after step 1 as well as after step 5, the residual capacities of edges e_1 , e_2 and e_3 are in the form r^n , r^{n+1} and 0 , respectively, for some $n \in \mathbb{N}$. This means that we can use augmenting paths p_1 , p_2 , p_1 and p_3 infinitely many times and residual capacities of these edges will always be in the same form. Total flow in the network after step 5 is $1 + 2(r^1 + r^2)$. If we continue to use augmenting paths as above, the total flow converges to $1 + 2 \sum_{i=1}^{\infty} r^i = 3 + 2r$, while the maximum flow is $2M + 1$. In this case, the algorithm never terminates and the flow doesn't even converge to the maximum flow.^[4]

46.5 Python implementation

```

class Edge(object):
    def __init__(self, u, v, w):
        self.source = u
        self.sink = v
        self.capacity = w
    def __repr__(self):
        return "%s->%s:%s" % (self.source, self.sink, self.capacity)
class FlowNetwork(object):
    def __init__(self):
        self.adj = {}
        self.flow = {}
    def add_vertex(self, vertex):
        self.adj[vertex] = []
    def get_edges(self, v):
        return self.adj[v]
    def add_edge(self, u, v, w=0):
        if u == v:
            raise ValueError("u == v")
        edge = Edge(u, v, w)
        edge.redge = edge
        edge.redge.redge = edge
        self.adj[u].append(edge)
        self.adj[v].append(edge.redge)
        self.flow[edge] = 0
        self.flow[edge.redge] = 0
    def find_path(self, source, sink, path):
        if source == sink:
            return path
        if source not in self.get_edges(source):
            return None
        result = self.find_path(source, sink, path + [source])
        if result != None:
            return result
        for edge in self.get_edges(source):
            if edge not in path:
                residual = edge.capacity - self.flow[edge]
                if residual > 0:
                    result = self.find_path(edge.sink, sink, path + [edge])
                    if result != None:
                        return result
    def max_flow(self, source, sink):
        path = self.find_path(source, sink, [])
        while path != None:
            residuals = [edge.capacity - self.flow[edge] for edge in path]
            flow = min(residuals)
            for edge in path:
                self.flow[edge] += flow
                self.flow[edge.redge] -= flow
            path = self.find_path(source, sink, [])
        return sum(self.flow[edge] for edge in self.get_edges(source))
  
```

46.5.1 Usage example

For the example flow network in maximum flow problem we do the following:

```

>>> g = FlowNetwork()
>>> [g.add_vertex(v) for v in "sopqrt"]
[None, None, None, None, None, None]
>>> g.add_edge('s','o',3)
>>> g.add_edge('s','p',3)
>>> g.add_edge('o','p',2)
>>> g.add_edge('o','q',3)
>>> g.add_edge('p','r',2)
>>> g.add_edge('r','t',3)
>>> g.add_edge('q','r',4)
>>> g.add_edge('q','t',2)
>>> print(g.max_flow('s','t'))
5
  
```

46.6 Notes

- [1] Laung-Terng Wang, Yao-Wen Chang, Kwang-Ting (Tim) Cheng (2009). *Electronic Design Automation: Synthesis, Verification, and Test*. Morgan Kaufmann. p. 204. ISBN 0080922007.
- [2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein (2009). *Introduction to Algorithms*. MIT Press. p. 714. ISBN 0262258102.
- [3] Ford, L. R.; Fulkerson, D. R. (1956). “Maximal flow through a network”. *Canadian Journal of Mathematics* **8**: 399. doi:10.4153/CJM-1956-045-5.
- [4] Zwick, Uri (21 August 1995). “The smallest networks on which the Ford–Fulkerson maximum flow procedure may fail to terminate”. *Theoretical Computer Science* **148** (1): 165–170. doi:10.1016/0304-3975(95)00022-O.

46.7 References

- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). “Section 26.2: The Ford–Fulkerson method”. *Introduction to Algorithms* (Second ed.). MIT Press and McGraw–Hill. pp. 651–664. ISBN 0-262-03293-7.
- George T. Heineman, Gary Pollice, and Stanley Selkow (2008). “Chapter 8:Network Flow Algorithms”. *Algorithms in a Nutshell*. O'Reilly Media. pp. 226–250. ISBN 978-0-596-51624-6.
- Jon Kleinberg and Éva Tardos (2006). “Chapter 7:Extensions to the Maximum-Flow Problem”. *Algorithm Design*. Pearson Education. pp. 378–384. ISBN 0-321-29535-8.

46.8 See also

- Approximate max-flow min-cut theorem

46.9 External links

- A tutorial explaining the Ford–Flukerson method to solve the max-flow problem
- Another Java animation
- Java Web Start application

Media related to Ford–Fulkerson algorithm at Wikimedia Commons

Chapter 47

Push–relabel maximum flow algorithm

In mathematical optimization, the **push–relabel algorithm** (alternatively, **preflow–push algorithm**) is an algorithm for computing **maximum flows**. The name “push–relabel” comes from the two basic operations used in the algorithm. Throughout its execution, the algorithm maintains a “preflow” and gradually converts it into a maximum flow by moving flow locally between neighboring vertices using *push* operations under the guidance of an admissible network maintained by *relabel* operations. In comparison, the **Ford–Fulkerson algorithm** performs global augmentations that send flow following paths from the source all the way to the sink.^[1]

The push–relabel algorithm is considered one of the most efficient maximum flow algorithms. The generic algorithm has a **strongly polynomial** $O(V^2E)$ time complexity, which is asymptotically more efficient than the $O(VE^2)$ Edmonds–Karp algorithm.^[2] Specific variants of the algorithms achieve even lower time complexities. The variant based on the highest label vertex selection rule has $O(V^2\sqrt{E})$ time complexity and is generally regarded as the benchmark for maximum flow algorithms.^{[3][4]} Subcubic $O(VE \log (V^2/E))$ time complexity can be achieved using **dynamic trees**, although in practice it is less efficient.^[2]

The push–relabel algorithm has been extended to compute **minimum cost flows**.^[5] The idea of distance labels has led to a more efficient augmenting path algorithm, which in turn can be incorporated back into the push–relabel algorithm to create a variant with even higher empirical performance.^{[4][6]}

47.1 History

The concept of a preflow was originally designed by Alexander V. Karzanov and was published in 1974 in Soviet Mathematical Dokladi 15. This preflow algorithm also used a push operation; however, it used distances in the auxiliary network to determine where to push the flow instead of a labeling system.^{[2][7]}

The push–relabel algorithm was designed by Andrew V. Goldberg and Robert Tarjan. The algorithm was initially present in November 1986 in STOC ‘86: Proceed-

ings of the eighteenth annual ACM symposium on Theory of computing, and then officially in October 1988 as an article in the Journal of the ACM. Both papers detail a generic form of the algorithm terminating in $O(V^2E)$ along with a $O(V^3)$ sequential implementation, a $O(VE \log (V^2/E))$ implementation using dynamic trees, and parallel/distributed implementation.^{[2][8]}

47.2 Concepts

47.2.1 Definitions and notations

Main article: **Flow network**

Consider a flow network $G = (V, E)$ with a pair of distinct vertices s and t designated as the source and the sink, respectively. The $c(u, v) \geq 0$ relation denotes the capacity of each edge $(u, v) \in E$. If $(u, v) \notin E$, then we assume that $c(u, v) = 0$. A flow on G is a function **real** function $f : V \times V \rightarrow \mathbb{R}$ satisfying the following conditions:

The push–relabel algorithm introduces the concept of **preflows**. A preflow is a function with a definition almost identical to that of a flow except that it relaxes the flow conservation condition. Instead of requiring strict flow balance at vertices other than s and t , it allows them to carry positive excesses. This means that in a preflow the total flow into a vertex can exceed the flow out of the vertex. Put symbolically:

A vertex v is called *active* if $e(v) > 0$ for $v \in V - \{s, t\}$.

For each $(u, v) \in V \times V$, denote its *residual capacity* by $c_f(u, v) = c(u, v) - f(u, v)$. The residual network of G with respect to a preflow f is defined as $G_f(V, E_f)$ where the residual edges are defined as $E_f = \{(u, v) | u, v \in$

$V \wedge c_f(u, v) > 0\}$. If there is no path from any active vertex to t in G_f , then preflow is called *maximum*. In a maximum preflow, $e(t)$ is equal to the value of a maximum flow; if T is the set of vertices from which t is reachable in G_f , and $S = V \setminus T$, then (S, T) is a minimum *s-t cut*.

The push-relabel algorithm uses a nonnegative integer *valid labeling* function which makes use of *distance labels*, or *heights*, on vertices to determine which vertex pair should be selected for the push operation. This labeling function is denoted by $h(v), v \in V$. This function must satisfy the following conditions in order to be considered valid:

In the algorithm, the height values of s and t are fixed. $h(u)$ is a lower bound of the unweighted distance from u to t in G_f if t is reachable from u . If u has been disconnected from t , then $h(u) - |V|$ is a lower bound of the unweighted distance from u to s . As a result, if a valid height function exists, there are no *s-t* paths in G_f because no such paths can be longer than $|V| - 1$.

An edge $(u, v) \in E_f$ is called *admissible* if $h(u) = h(v) + 1$. The network $\tilde{G}_f(V, \tilde{E}_f)$ when $\tilde{E}_f = \{(u, v) | (u, v) \in E_f \wedge h(u) = h(v) + 1\}$ is called the *admissible network*. The admissible network is acyclic.

47.2.2 Operations

Initialization

The algorithm starts by creating a residual graph, initializing the preflow values to zero and performing a set of saturating push operations on residual edges exiting the source, (s, v) where $v \in V \setminus \{s\}$. Similarly, the label heights are initialized such that the height at the source is in the number of vertices in the graph, $h(s) = |V|$, and all other vertices are given a height of zero. Once the initialization is complete, the algorithm repeatedly performs either the push or relabel operations against active vertices until no applicable operation can be performed.

Push

The push operation applies on an admissible out-edge (u, v) of an active vertex u in G_f . It moves $\min\{e(u), c_f(u, v)\}$ units of flow from u to v .

push(u, v): assert $e[u] > 0$ and $h[u] == h[v] + 1 \Delta = \min(e[u], c[u][v] - f[u][v]) f[u][v] += \Delta f[v][u] -= \Delta e[u] -= \Delta e[v] += \Delta$

A push operation that causes $f(u, v)$ to reach $c(u, v)$ is called a *saturating* push since it uses up all the available capacity of the residual edge. Otherwise, all of the excess

at the vertex is pushed across the residual edge. This is called an *unsaturating* or *non-saturating* push.

Relabel

The relabel operation applies on an active vertex u without any admissible out-edges in G_f . It modifies $h(u)$ to the minimum value such that an admissible out-edge is created. Note that this always increases $h(u)$ and never creates a steep edge, which is an edge (u, v) such that $c_f(u, v) > 0$, and $h(u) > h(v) + 1$.

relabel(u): assert $e[u] > 0$ and $h[u] <= h[v] \forall v$ such that $f[u][v] < c[u][v]$ $h[u] = \min(h[v]) \forall v$ such that $f[u][v] < c[u][v] + 1$

Effects of push and relabel

After a push or relabel operation, h remains a valid height function with respect to f .

For a push operation on an admissible edge (u, v) , it may add an edge (v, u) to E_f , where $h(v) = h(u) - 1 \leq h(u) + 1$; it may also remove the edge (u, v) from E_f , where it effectively removes the constraint $h(u) \leq h(v) + 1$.

To see that a relabel operation on vertex u preserves the validity of $h(u)$, notice that this is trivially guaranteed by definition for the out-edges of u in G_f . For the in-edges of u in the G_f , the increased $h(u)$ can only satisfy the constraints less tightly, not violate them.

47.3 The generic push-relabel algorithm

47.3.1 Description

The following algorithm is a generic version of the push-relabel algorithm. It is used as a proof of concept and does not contain implementation details on how to select an active vertex for the push and relabel operations. This generic version of the algorithm will terminate in $O(V^2E)$.

Since $h(s) = |V|$, $h(t) = 0$, and there are no paths longer than $|V| - 1$ in G_f , in order for $h(s)$ to satisfy the valid labeling condition, s must be disconnected from t . At initialization, the algorithm fulfills this requirement by creating a preflow f that saturates all out-edges of s , after which $h(v) = 0$ is trivially valid for all $v \in V \setminus \{s, t\}$. After initialization, the algorithm repeatedly executes an applicable push or relabel operation until no such operations apply, at which point the preflow has been converted into a maximum flow.

generic-push-relabel($G(V, E), s, t$): create a preflow f that saturates all out-edges of s let $h[s] = |V|$ let $h[v] = 0 \forall v \in V$

$\setminus \{s\}$ while there is an applicable push or relabel operation execute the operation

47.3.2 Correctness

The algorithm maintains the condition that h is a valid labeling during its execution. This can be proven true by examining the effects of the push and relabel operations on the label function h . The relabel operation increases the label value by the associated minimum plus one which will always satisfy the $h(u) \leq h(v) + 1$ constraint. The push operation can send flow from u to v if $h(u) = h(v) + 1$. This may add (v, u) to G_f and may delete (u, v) from G_f . The addition of (v, u) to G_f will not affect the valid labeling since $h(v) = h(u) - 1$. The deletion of (u, v) from G_f removes corresponding constraint since the valid labeling property $h(u) \leq h(v) + 1$ only applies to residual edges in G_f .^[8]

If a preflow f and a valid labeling h for f exists then there is no augmenting path from s to t in the residual graph G_f . This can be proven by contradiction based on inequalities which arise in the labeling function when supposing that an augmenting path does exist. If the algorithm terminates, then all vertices in $V - \{s, t\}$ are not active. This means all $v \in V - \{s, t\}$ have no excess flow, and with no excess the preflow f obeys the flow conservation constraint and can be considered a normal flow. This flow is the maximum flow according to the max-flow min-cut theorem since there is no augmenting path from s to t .^[8]

Therefore, the algorithm will return the maximum flow upon termination.

47.3.3 Time complexity

In order to bind the time complexity of the algorithm, we must analyze the number of push and relabel operations which occur within the main loop. The numbers of relabel, saturating push and nonsaturating push operations are analyzed separately.

In the algorithm, the relabel operation can be performed at most $(2|V| - 1)(|V| - 2) < 2|V|^2$ times. This is because the labeling $h(u)$ value for any vertex u can never decrease, and the maximum label value is at most $2|V| - 1$ for all vertices. This means the relabel operation could potentially be performed $2|V| - 1$ times for all vertices $V \setminus \{s, t\}$ (i.e. $|V| - 2$). This results in a bound of $O(V^2)$ for the relabel operation.

Each saturating push on an admissible edge (u, v) removes the edge from G_f . For the edge to be reinserted into G_f for another saturating push, v must be first relabeled, followed by a push on edge (v, u) , then u must be relabeled. In the process, $h(u)$ increases by at least two. Therefore, there are $O(V)$ saturating pushes on (u, v) , and the total number of saturating pushes is at most $2|V||E|$. This results in a time bound of $O(VE)$ for the saturating push

operations.

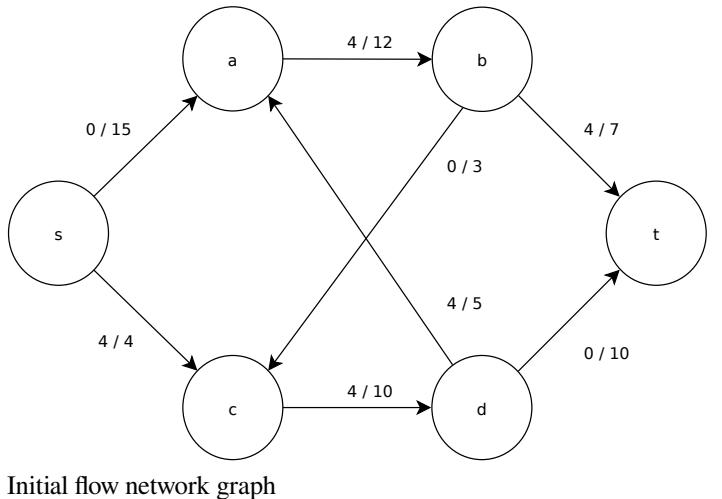
Bounding the number of nonsaturating pushes can be achieved via a potential argument. We use the potential function $\Phi = \sum_{u \in V \wedge e(u) > 0} h(u)$ (i.e. Φ is the sum of the heights of all active vertices). It is obvious that Φ is $|V|$ initially and stays nonnegative throughout the execution of the algorithm. Both relabels and saturating pushes can increase Φ . However, the value of Φ must be equal 0 at termination since there cannot be any remaining active vertices at the end of the algorithm's execution. This means that over the execution of the algorithm, the nonsaturating pushes must make up the difference of the relabel and saturating push operations in order for Φ to terminate with a value of 0.

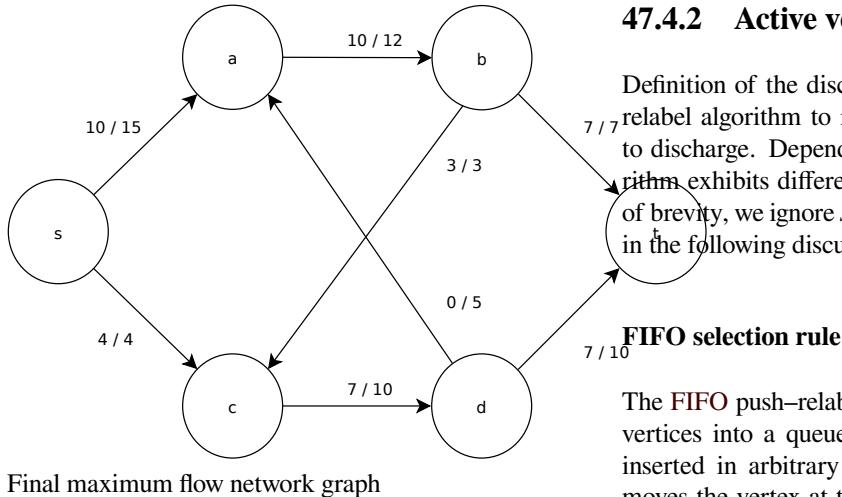
The relabel operation can increase Φ by at most $(2|V| - 1)(|V| - 2)$. A saturating push on (u, v) activates v if it was inactive before the push, increasing Φ by at most $2|V| - 1$. Hence, the total contribution of all saturating pushes operations to Φ is at most $(2|V| - 1)(2|V||E|)$. A nonsaturating push on (u, v) always deactivates u , but it can also activate v as in a saturating push. As a result, it decreases Φ by at least $h(u) - h(v) = 1$. Since relabels and saturating pushes increase Φ , the total number of nonsaturating pushes must make up the difference of $(2|V| - 1)(|V| - 2) + (2|V| - 1)(2|V||E|) \leq 4|V|^2|E|$. This results in a time bound of $O(V^2E)$ for the nonsaturating push operations.

In sum, the algorithm executes $O(V^2)$ relabels, $O(VE)$ saturating pushes and $O(V^2E)$ nonsaturating pushes. Data structures can be designed to pick and execute an applicable operation in $O(1)$ time. Therefore, the time complexity of the algorithm is $O(V^2E)$.^{[1][8]}

47.3.4 Example

The following is a sample execution of the generic push-relabel algorithm, as defined above, on the following simple network flow graph diagram.





In the example, the h and e values denote the height and excess, respectively, of the vertex during the execution of the algorithm. Each residual graph in the example only contains the residual edges with a capacity larger than zero. Each residual graph may contain multiple iterations of the *perform operation* loop.

The example (but with initial flow of 0) can be run [here](#) interactively.

47.4 Practical implementations

While the generic push–relabel algorithm has $O(V^2E)$ time complexity, efficient implementations achieve $O(V^3)$ or lower time complexity by enforcing appropriate rules in selecting applicable push and relabel operations. The empirical performance can be further improved by heuristics.

47.4.1 “Current-edge” data structure and discharge operation

The “current-edge” data structure is a mechanism for visiting the in- and out-neighbors of a vertex in the flow network in a static circular order. If a singly linked list of neighbors is created for a vertex, the data structure can be as simple as a pointer into the list that steps through the list and rewinds to the head when it runs off the end.

Based on the “current-edge” data structure, the discharge operation can be defined. A discharge operation applies on an active node and repeatedly pushes flow from the node until it becomes inactive, relabeling it as necessary to create admissible edges in the process.

`discharge(u): while e[u] > 0 if current-edge[u] has run off the end of neighbors[u] relabel(u) rewind current-edge[u] else let (u, v) = current-edge[u] if (u, v) is admissible push(u, v) else let current-edge[u] point to the next neighbor of u`

47.4.2 Active vertex selection rules

Definition of the discharge operation reduces the push–relabel algorithm to repeatedly selecting an active node to discharge. Depending on the selection rule, the algorithm exhibits different time complexities. For the sake of brevity, we ignore s and t when referring to the vertices in the following discussion.

FIFO selection rule

The **FIFO** push–relabel algorithm^[2] organizes the active vertices into a queue. The initial active nodes can be inserted in arbitrary order. The algorithm always removes the vertex at the front of the queue for discharging. Whenever an inactive vertex becomes active, it is appended to the back of the queue.

The algorithm has $O(V^3)$ time complexity.

Relabel-to-front selection rule

The relabel-to-front push–relabel algorithm^[1] organizes all vertices into a linked list and maintains the invariant that the list is topologically sorted with respect to the admissible network. The algorithm scans the list from front to back and performs a discharge operation on the current vertex if it is active. If the node is relabeled, it is moved to the front of the list, and the scan is restarted from the front.

The algorithm also has $O(V^3)$ time complexity.

Highest label selection rule

The highest-label push–relabel algorithm^[9] organizes all vertices into buckets indexed by their heights. The algorithm always selects an active vertex with the largest height to discharge.

The algorithm has $O(V^2\sqrt{E})$ time complexity. If the lowest-label selection rule is used instead, the time complexity becomes $O(V^2E)$.^[3]

47.4.3 Implementation techniques

Although in the description of the generic push–relabel algorithm above, $h(u)$ is set to zero for each vertex u other than s and t at the beginning, it is preferable to perform a backward **breadth-first search** from t to compute the exact heights.^[2]

The algorithm is typically separated into two phases. Phase one computes a maximum preflow by discharging only active vertices whose heights are below n . Phase two converts the maximum preflow into a maximum flow by returning excess flow that cannot reach t to s . It can be

shown that phase two has $O(VE)$ time complexity regardless of the order of push and relabel operations and is therefore dominated by phase one. Alternatively, it can be implemented using flow decomposition.^[10]

Heuristics are crucial to improving the empirical performance of the algorithm.^[11] Two commonly used heuristics are the gap heuristic and the global relabeling heuristic.^{[2][12]} The gap heuristic detects gaps in the height function. If there is a height $0 < \tilde{h} < |V|$ for which there is no vertex u such that $h(u) = \tilde{h}$, then any vertex u with $\tilde{h} < h(u) < |V|$ has been disconnected from t and can be relabeled to $(|V| + 1)$ immediately. The global relabeling heuristic periodically performs backward breadth-first search from t in G_f to compute the exact heights of the vertices. Both heuristics skip unhelpful relabel operations, which are a bottleneck of the algorithm and contribute to the ineffectiveness of dynamic trees.^[4]

47.5 Sample implementations

C implementation

```
#include <stdlib.h> #include <stdio.h> #define NODES 6 #define MIN(X,Y) ((X) < (Y) ? (X) : (Y)) #define INFINITE 10000000 void push(const int * const * C, int ** F, int *excess, int u, int v) { int send = MIN(excess[u], C[u][v] - F[u][v]); F[u][v] += send; F[v][u] -= send; excess[u] -= send; excess[v] += send; } void relabel(const int * const * C, const int * const * F, int *height, int u) { int v; int min_height = INFINITE; for (v = 0; v < NODES; v++) { if (C[u][v] - F[u][v] > 0) { min_height = MIN(min_height, height[v]); height[u] = min_height + 1; } } } void discharge(const int * const * C, int ** F, int *excess, int *height, int *seen, int u) { while (excess[u] > 0) { if (seen[u] < NODES) { int v = seen[u]; if ((C[u][v] - F[u][v] > 0) && (height[u] > height[v])){ push(C, F, excess, u, v); } else seen[u] += 1; } else { relabel(C, F, height, u); seen[u] = 0; } } } void moveToFront(int i, int *A) { int temp = A[i]; int n; for (n = i; n > 0; n--){ A[n] = A[n-1]; } A[0] = temp; } int pushRelabel(const int * const * C, int ** F, int source, int sink) { int *excess, *height, *list, *seen, i, p; excess = (int *) calloc(NODES, sizeof(int)); height = (int *) calloc(NODES, sizeof(int)); seen = (int *) calloc(NODES, sizeof(int)); list = (int *) calloc((NODES-2), sizeof(int)); for (i = 0, p = 0; i < NODES; i++){ if((i != source) && (i != sink)) { list[p] = i; p++; } } height[source] = NODES; excess[source] = INFINITE; for (i = 0; i < NODES; i++) push(C, F, excess, source, i); p = 0; while (p < NODES - 2) { int u = list[p]; int old_height = height[u]; discharge(C, F, excess, height, seen, u); if (height[u] > old_height) { moveToFront(p,list); p=0; } else p += 1; } int maxflow = 0; for (i = 0; i < NODES; i++) maxflow += F[source][i]; free(list); free(seen); free(height); free(excess); return maxflow; } void printMatrix(const int * const * M) { int i,j; for (i = 0; i < NODES; i++) { for (j = 0; j < NODES; j++) printf("%d\t",M[i][j]); printf("\n"); } }
```

```
int main(void) { int **flow, **capacities, i; flow = (int **) malloc(NODES, sizeof(int*)); capacities = (int **) malloc(NODES, sizeof(int*)); for (i = 0; i < NODES; i++) { flow[i] = (int *) malloc(NODES, sizeof(int)); capacities[i] = (int *) malloc(NODES, sizeof(int)); } //Sample graph capacities[0][1] = 2; capacities[0][2] = 9; capacities[1][2] = 1; capacities[1][3] = 0; capacities[1][4] = 0; capacities[2][4] = 7; capacities[3][5] = 7; capacities[4][5] = 4; printf("Capacity:\n"); printMatrix(capacities); printf("Max Flow:\n%d\n", pushRelabel(capacities, flow, 0, 5)); printf("Flows:\n"); printMatrix(flow); return 0; }
```

Python implementation

```
def relabel_to_front(C, source, sink): n = len(C) # C is the capacity matrix F = [[0] * n for _ in xrange(n)] # residual capacity from u to v is C[u][v] - F[u][v] height = [0] * n # height of node excess = [0] * n # flow into node minus flow from node seen = [0] * n # neighbours seen since last relabel # node "queue" nodelist = [i for i in xrange(n) if i != source and i != sink] def push(u, v): send = min(excess[u], C[u][v] - F[u][v]) F[u][v] += send F[v][u] -= send excess[u] -= send excess[v] += send def relabel(u): # find smallest new height making a push possible, # if such a push is possible at all min_height = infinity for v in xrange(n): if C[u][v] - F[u][v] > 0: min_height = min(min_height, height[v]) height[u] = min_height + 1 def discharge(u): while excess[u] > 0: if seen[u] < n: # check next neighbour v = seen[u] if C[u][v] - F[u][v] > 0 and height[u] > height[v]: push(u, v) else: seen[u] += 1 else: # we have checked all neighbours. must relabel relabel(u) seen[u] = 0 height[source] = n # longest path from source to sink is less than n long excess[source] = infinity # send as much flow as possible to neighbours of source for v in xrange(n): push(source, v) p = 0 while p < len(nodelist): u = nodelist[p] old_height = height[u] discharge(u) if height[u] > old_height: nodelist.insert(0, nodelist.pop(p)) # move to front of list p = 0 # start from front of list else: p += 1 return sum(F[source])
```

47.6 References

- [1] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. (2001). “§26 Maximum flow”. *Introduction to Algorithms* (2nd ed.). The MIT Press. pp. 643–698. ISBN 0262032937.
- [2] Goldberg, A V; Tarjan, R E (1986). “A new approach to the maximum flow problem”. *Proceedings of the eighteenth annual ACM symposium on Theory of computing - STOC '86*. p. 136. doi:10.1145/12130.12144. ISBN 0897911938.
- [3] Ahuja, Ravindra K.; Kodialam, Murali; Mishra, Ajay K.; Orlin, James B. (1997). “Computational investigations of maximum flow algorithms”. *European Journal of Operational Research* 97 (3): 509. doi:10.1016/S0377-2217(96)00269-X.

- [4] Goldberg, Andrew V. (2008). “The Partial Augment-Relabel Algorithm for the Maximum Flow Problem”. *Algorithms - ESA 2008*. Lecture Notes in Computer Science **5193**. p. 466. doi:10.1007/978-3-540-87744-8_39. ISBN 978-3-540-87743-1.
- [5] Goldberg, Andrew V (1997). “An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm”. *Journal of Algorithms* **22**: 1. doi:10.1006/jagm.1995.0805.
- [6] Ahuja, Ravindra K.; Orlin, James B. (1991). “Distance-directed augmenting path algorithms for maximum flow and parametric maximum flow problems”. *Naval Research Logistics* **38** (3): 413. doi:10.1002/1520-6750(199106)38:3<413::AID-NAV3220380310>3.0.CO;2-J.
- [7] Goldberg, Andrew V.; Tarjan, Robert E. (2014). “Efficient maximum flow algorithms”. *Communications of the ACM* **57** (8): 82. doi:10.1145/2628036.
- [8] Goldberg, Andrew V.; Tarjan, Robert E. (1988). “A new approach to the maximum-flow problem”. *Journal of the ACM* **35** (4): 921. doi:10.1145/48014.61051.
- [9] Cherian, J.; Maheshwari, S. N. (1988). “Analysis of pre-flow push algorithms for maximum network flow”. *Foundations of Software Technology and Theoretical Computer Science*. Lecture Notes in Computer Science **338**. p. 30. doi:10.1007/3-540-50517-2_69. ISBN 978-3-540-50517-4.
- [10] Ahuja, R. K.; Magnanti, T. L.; Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications* (1st ed.). Prentice Hall. ISBN 013617549X.
- [11] Cherkassky, Boris V.; Goldberg, Andrew V. (1995). “On implementing push-relabel method for the maximum flow problem”. *Integer Programming and Combinatorial Optimization*. Lecture Notes in Computer Science **920**. p. 157. doi:10.1007/3-540-59408-6_49. ISBN 978-3-540-59408-6.
- [12] Derigs, U.; Meier, W. (1989). “Implementing Goldberg’s max-flow-algorithm ? A computational investigation”. *ZOR Zeitschrift für Operations Research Methods and Models of Operations Research* **33** (6): 383. doi:10.1007/BF01415937.

Chapter 48

Evolutionary algorithm

In artificial intelligence, an **evolutionary algorithm** (EA) is a subset of evolutionary computation, a generic population-based metaheuristic optimization algorithm. An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the solutions (see also loss function). Evolution of the population then takes place after the repeated application of the above operators. *Artificial evolution* (AE) describes a process involving individual *evolutionary algorithms*; EAs are individual components that participate in an AE.

Evolutionary algorithms often perform well approximating solutions to all types of problems because they ideally do not make any assumption about the underlying fitness landscape; this generality is shown by successes in fields as diverse as engineering, art, biology, economics, marketing, genetics, operations research, robotics, social sciences, physics, politics and chemistry.

Techniques from evolutionary algorithms applied to the modeling of biological evolution are generally limited to explorations of microevolutionary processes and planning models based upon cellular processes. The computer simulations *Tierra* and *Avida* attempt to model macroevolutionary dynamics.

In most real applications of EAs, computational complexity is a prohibiting factor. In fact, this computational complexity is due to fitness function evaluation. Fitness approximation is one of the solutions to overcome this difficulty. However, seemingly simple EA can solve often complex problems; therefore, there may be no direct link between algorithm complexity and problem complexity.

A possible limitation of many evolutionary algorithms is their lack of a clear genotype-phenotype distinction. In nature, the fertilized egg cell undergoes a complex process known as **embryogenesis** to become a mature phenotype. This indirect encoding is believed to make the genetic search more robust (i.e. reduce the probability of fatal mutations), and also may improve the **evolvability** of the organism.^{[1][2]} Such indirect (aka generative or developmental) encodings also enable evolution to exploit the regularity in the environment.^[3] Recent work in the

field of **artificial embryogeny**, or **artificial developmental systems**, seeks to address these concerns. And **gene expression programming** successfully explores a genotype-phenotype system, where the genotype consists of linear multigenic chromosomes of fixed length and the phenotype consists of multiple expression trees or computer programs of different sizes and shapes.^[4]

48.1 Implementation of biological processes

1. Generate the initial population of individuals randomly - first generation
2. Evaluate the fitness of each individual in that population
3. Repeat on this generation until termination (time limit, sufficient fitness achieved, etc.):
 - (a) Select the best-fit individuals for reproduction - parents
 - (b) Breed new individuals through crossover and mutation operations to give birth to offspring
 - (c) Evaluate the individual fitness of new individuals
 - (d) Replace least-fit population with new individuals

48.2 Evolutionary algorithm types

Similar techniques differ in the implementation details and the nature of the particular applied problem.

- **Genetic algorithm** - This is the most popular type of EA. One seeks the solution of a problem in the form of strings of numbers (traditionally binary, although the best representations are usually those that reflect something about the problem being solved), by applying operators such as recombination and mutation (sometimes one, sometimes both). This type of EA is often used in optimization problems.

- **Genetic programming** - Here the solutions are in the form of computer programs, and their fitness is determined by their ability to solve a computational problem.
- **Evolutionary programming** - Similar to genetic programming, but the structure of the program is fixed and its numerical parameters are allowed to evolve.
- **Gene expression programming** - Like genetic programming, GEP also evolves computer programs but it explores a genotype-phenotype system, where computer programs of different sizes are encoded in linear chromosomes of fixed length.
- **Evolution strategy** - Works with vectors of real numbers as representations of solutions, and typically uses self-adaptive mutation rates.
- **Differential evolution** - Based on vector differences and is therefore primarily suited for numerical optimization problems.
- **Neuroevolution** - Similar to genetic programming but the genomes represent artificial neural networks by describing structure and connection weights. The genome encoding can be direct or indirect.
- **Learning classifier system** - Here the solutions are classifiers (rules or conditions). A Michigan-LCS works with individual classifiers whereas a Pittsburgh-LCS uses populations of classifier-sets. Initially, classifiers were only binary, but now include real, neural net, or S-expression types. Fitness is determined with either a strength or accuracy based reinforcement learning approach.
- **Particle swarm optimization** - Based on the ideas of animal flocking behaviour. Also primarily suited for numerical optimization problems.

48.3 Related techniques

Swarm algorithms, including:

- **Ant colony optimization** - Based on the ideas of ant foraging by pheromone communication to form paths. Primarily suited for combinatorial optimization and graph problems.
- **Artificial Bee Colony Algorithm** - Based on the honey bee foraging behaviour. Primarily proposed for numerical optimization and extended to solve combinatorial, constrained and multi-objective optimization problems.
- **Bees algorithm** is based on the foraging behaviour of honey bees. It has been applied in many applications such as routing and scheduling.
- **Cuckoo search** is inspired by the brooding parasitism of the cuckoo species. It also uses Lévy flights, and thus it suits for global optimization problems.

48.4 Other population-based metaheuristic method

Firefly algorithm is inspired by the behavior of fireflies, attracting each other by flashing light. This is especially useful for multimodal optimization.

- **Harmony search** - Based on the ideas of musicians' behavior in searching for better harmonies. This algorithm is suitable for combinatorial optimization as well as parameter optimization.
- **Gaussian adaptation** - Based on information theory. Used for maximization of manufacturing yield, mean fitness or average information. See for instance Entropy in thermodynamics and information theory.
- **Memetic algorithm** - It is the hybrid form of population based methods. Inspired by Dawkins' notion of a meme, it commonly takes the form of a population-based algorithm coupled with individual learning procedures capable of performing local refinements. Emphasizes the exploitation of problem-specific knowledge, and tries to orchestrate local and global search in a synergic way.

48.5 See also

- Artificial development
- Developmental biology
- Digital organism
- Estimation of distribution algorithm
- Evolutionary computation
- Evolutionary robotics
- Fitness function
- Fitness landscape
- Fitness approximation
- Genetic operators
- Interactive evolutionary computation
- List of digital organism simulators
- Map of Evolutionary Algorithms
- No free lunch in search and optimization
- Program synthesis
- Test functions for optimization

48.6 Gallery [5]

- A two-population EA search of a bounded optima in 2D
- A two-population EA search over a constrained Rosenbrock function with bounded global optimum.
- A two-population EA search over a constrained Rosenbrock function. Global optimum is not bounded.
- Estimation of Distribution Algorithm over Keane's function

48.7 References

- [1] G.S. Hornby and J.B. Pollack. Creating high-level components with a generative representation for body-brain evolution. *Artificial Life*, 8(3):223–246, 2002.
- [2] Jeff Clune, Benjamin Beckmann, Charles Ofria, and Robert Pennock. “Evolving Coordinated Quadruped Gaits with the HyperNEAT Generative Encoding”. *Proceedings of the IEEE Congress on Evolutionary Computing Special Section on Evolutionary Robotics*, 2009. Trondheim, Norway.
- [3] J. Clune, C. Ofria, and R. T. Pennock, “How a generative encoding fares as problem-regularity decreases,” in PPSN (G. Rudolph, T. Jansen, S. M. Lucas, C. Poloni, and N. Beume, eds.), vol. 5199 of Lecture Notes in Computer Science, pp. 358–367, Springer, 2008.
- [4] Ferreira, C., 2001. Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. *Complex Systems*, Vol. 13, issue 2: 87-129.
- [5] Simionescu, P.A. (2014). *Computer Aided Graphing and Simulation Tools for AutoCAD Users* (1st ed.). Boca Raton, FL: CRC Press. ISBN 9-781-48225290-3.

48.8 Bibliography

- Ashlock, D. (2006), *Evolutionary Computation for Modeling and Optimization*, Springer, ISBN 0-387-22196-4.
- Bäck, T. (1996), *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford Univ. Press.
- Bäck, T., Fogel, D., Michalewicz, Z. (1997), *Handbook of Evolutionary Computation*, Oxford Univ. Press.
- Banzhaf, W., Nordin, P., Keller, R., Francone, F. (1998), *Genetic Programming - An Introduction*, Morgan Kaufmann, San Francisco

- Eiben, A.E., Smith, J.E. (2003), *Introduction to Evolutionary Computing*, Springer.
- Holland, J. H. (1975), *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor
- Michalewicz Z., Fogel D.B. (2004). How To Solve It: Modern Heuristics, Springer.
- Poli, R., Langdon, W. B., McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. Lulu.com, freely available from the internet. ISBN 978-1-4092-0073-4.
- Price, K., Storn, R.M., Lampinen, J.A., (2005). “Differential Evolution: A Practical Approach to Global Optimization”, Springer.
- Ingo Rechenberg (1971): Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution (PhD thesis). Reprinted by Fromman-Holzboog (1973).
- Hans-Paul Schwefel (1974): Numerische Optimierung von Computer-Modellen (PhD thesis). Reprinted by Birkhäuser (1977).
- Simon, D. (2013): *Evolutionary Optimization Algorithms*, Wiley.
- *Computational Intelligence: A Methodological Introduction* by Kruse, Borgelt, Klawonn, Moewes, Steinbrecher, Held, 2013, Springer, ISBN 9781447150121

Chapter 49

Hill climbing

This article is about the mathematical algorithm. For other meanings such as the branch of motorsport, see [Hillclimbing \(disambiguation\)](#).

In computer science, **hill climbing** is a mathematical optimization technique which belongs to the family of **local search**. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.

For example, hill climbing can be applied to the **travelling salesman problem**. It is easy to find an initial solution that visits all the cities but will be very poor compared to the optimal solution. The algorithm starts with such a solution and makes small improvements to it, such as switching the order in which two cities are visited. Eventually, a much shorter route is likely to be obtained.

Hill climbing is good for finding a **local optimum** (a solution that cannot be improved by considering a neighbouring configuration) but it is not necessarily guaranteed to find the best possible solution (the **global optimum**) out of all possible solutions (the **search space**). In convex problems, hill-climbing *is* optimal. Examples of algorithms that solve convex problems by hill-climbing include the **simplex algorithm** for linear programming and **binary search**.^{[1]:253}

The characteristic that only local optima are guaranteed can be cured by using restarts (repeated local search), or more complex schemes based on iterations, like **iterated local search**, on memory, like **reactive search optimization** and **tabu search**, or memory-less stochastic modifications, like **simulated annealing**.

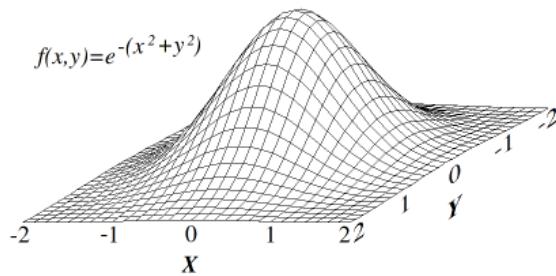
The relative simplicity of the algorithm makes it a popular first choice amongst optimizing algorithms. It is used widely in **artificial intelligence**, for reaching a goal state from a starting node. Choice of next node and starting node can be varied to give a list of related algorithms. Although more advanced algorithms such as **simulated annealing** or **tabu search** may give better results, in some situations hill climbing works just as well. Hill climbing can often produce a better result than other algorithms

when the amount of time available to perform a search is limited, such as with real-time systems. It is an **anytime algorithm**: it can return a valid solution even if it's interrupted at any time before it ends.

49.1 Mathematical description

Hill climbing attempts to maximize (or minimize) a target function $f(\mathbf{x})$, where \mathbf{x} is a vector of continuous and/or discrete values. At each iteration, hill climbing will adjust a single element in \mathbf{x} and determine whether the change improves the value of $f(\mathbf{x})$. (Note that this differs from **gradient descent** methods, which adjust all of the values in \mathbf{x} at each iteration according to the gradient of the hill.) With hill climbing, any change that improves $f(\mathbf{x})$ is accepted, and the process continues until no change can be found to improve the value of $f(\mathbf{x})$. \mathbf{x} is then said to be “locally optimal”.

In discrete vector spaces, each possible value for \mathbf{x} may be visualized as a **vertex** in a **graph**. Hill climbing will follow the graph from vertex to vertex, always locally increasing (or decreasing) the value of $f(\mathbf{x})$, until a local maximum (or local minimum) x_m is reached.



A convex surface. Hill-climbers are well-suited for optimizing over such surfaces, and will converge to the global maximum.

49.2 Variants

In **simple hill climbing**, the first closer node is chosen, whereas in **steepest ascent hill climbing** all successors are compared and the closest to the solution is chosen.

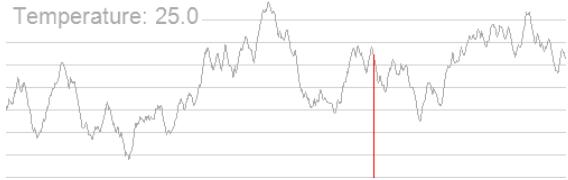
Both forms fail if there is no closer node, which may happen if there are local maxima in the search space which are not solutions. Steepest ascent hill climbing is similar to best-first search, which tries all possible extensions of the current path instead of only one.

Stochastic hill climbing does not examine all neighbors before deciding how to move. Rather, it selects a neighbor at random, and decides (based on the amount of improvement in that neighbor) whether to move to that neighbor or to examine another.

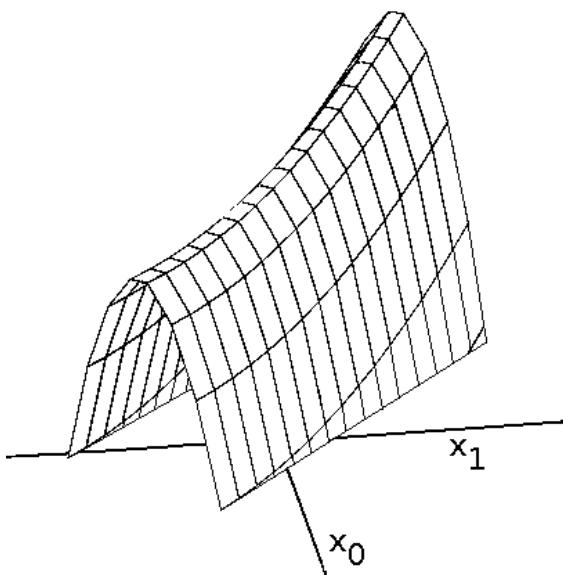
Coordinate descent does a line search along one coordinate direction at the current point in each iteration. Some versions of coordinate descent randomly pick a different coordinate direction each iteration.

Random-restart hill climbing is a meta-algorithm built on top of the hill climbing algorithm. It is also known as **Shotgun hill climbing**. It iteratively does hill-climbing, each time with a random initial condition x_0 . The best x_m is kept: if a new run of hill climbing produces a better x_m than the stored state, it replaces the stored state.

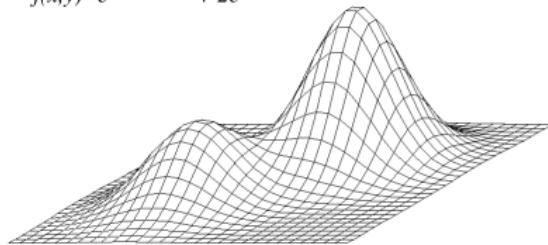
Random-restart hill climbing is a surprisingly effective algorithm in many cases. It turns out that it is often better to spend CPU time exploring the space, than carefully optimizing from an initial condition.



Despite the many local maxima in this graph, the global maximum can still be found using simulated annealing. Unfortunately, the applicability of simulated annealing is problem-specific because it relies on finding lucky jumps that improve the position. In problems that involve more dimensions, the cost of finding such a jump may increase exponentially with dimensionality. Consequently, there remain many problems for which hill climbers will efficiently find good results while simulated annealing will seemingly run forever without making progress. This depiction shows an extreme case involving only one dimension.



$$f(x,y)=e^{-(x^2+y^2)} + 2e^{-((x-1.7)^2+(y-1.7)^2)}$$



A surface with two local maxima. (Only one of them is the global maximum.) If a hill-climber begins in a poor location, it may converge to the lower maximum.

Hill climbing will not necessarily find the global maximum, but may instead converge on a **local maximum**. This problem does not occur if the heuristic is convex. However, as many functions are not convex hill climbing may often fail to reach a global maximum. Other local search algorithms try to overcome this problem such as stochastic hill climbing, random walks and simulated annealing.

49.3.2 Ridges and alleys

Ridges are a challenging problem for hill climbers that optimize in continuous spaces. Because hill climbers only adjust one element in the vector at a time, each step will move in an axis-aligned direction. If the target function creates a narrow ridge that ascends in a non-axis-aligned direction (or if the goal is to minimize, a narrow alley that descends in a non-axis-aligned direction), then the hill climber can only ascend the ridge (or descend the alley) by zig-zagging. If the sides of the ridge (or alley) are very steep, then the hill climber may be forced to take very tiny steps as it zig-zags toward a better position. Thus, it may take an unreasonable length of time for it to ascend the ridge (or descend the alley).

By contrast, gradient descent methods can move in any direction that the ridge or alley may ascend or descend. Hence, gradient descent or conjugate gradient method is generally preferred over hill climbing when the target

function is differentiable. Hill climbers, however, have the advantage of not requiring the target function to be differentiable, so hill climbers may be preferred when the target function is complex.

49.3.3 Plateau

Another problem that sometimes occurs with hill climbing is that of a plateau. A plateau is encountered when the search space is flat, or sufficiently flat that the value returned by the target function is indistinguishable from the value returned for nearby regions due to the precision used by the machine to represent its value. In such cases, the hill climber may not be able to determine in which direction it should step, and may wander in a direction that never leads to improvement.

49.4 Pseudocode

```
Discrete Space Hill Climbing Algorithm
currentNode = startNode; loop do
L = NEIGHBORS(currentNode);
nextEval = -INF; nextNode = NULL; for all x in L
if (EVAL(x) > nextEval) nextNode = x; nextEval =
EVAL(x); if nextEval <= EVAL(currentNode) //Return
current node since no better neighbors exist return
currentNode; currentNode = nextNode;
Continuous Space
Hill Climbing Algorithm
currentPoint = initialPoint; // the zero-magnitude vector is common
stepSize = initialStepSizes; // a vector of all 1's is common
acceleration = someAcceleration; // a value such as 1.2 is
common candidate[0] = -acceleration; candidate[1] =
-1 / acceleration; candidate[2] = 0; candidate[3] = 1
/ acceleration; candidate[4] = acceleration; loop do
before = EVAL(currentPoint); for each element i in
currentPoint do best = -1; bestScore = -INF; for j from
0 to 4 // try each of 5 candidate locations
currentPoint[i] = currentPoint[i] + stepSize[i] * candidate[j];
temp = EVAL(currentPoint); currentPoint[i] = current-
Point[i] - stepSize[i] * candidate[j]; if(temp > bestScore)
bestScore = temp; best = j; if candidate[best] is not 0
currentPoint[i] = currentPoint[i] + stepSize[i] * candi-
date[best]; stepSize[i] = stepSize[i] * candidate[best]; // accelerate if
(EVAL(currentPoint) - before) < epsilon
return currentPoint;
```

Contrast genetic algorithm; random optimization.

49.5 See also

- Gradient descent
- Greedy algorithm
- Tâtonnement
- Mean-shift

49.6 References

- Russell, Stuart J.; Norvig, Peter (2003), *Artificial Intelligence: A Modern Approach* (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, pp. 111–114, ISBN 0-13-790395-2
- [1] Skiena, Steven (2010). *The Algorithm Design Manual* (2nd ed.). Springer Science+Business Media. ISBN 1-849-96720-2.

This article is based on material taken from the [Free Online Dictionary of Computing](#) prior to 1 November 2008 and incorporated under the “relicensing” terms of the GFDL, version 1.3 or later.

49.7 External links

- [Hill Climbing visualization](#) A visualization of a hill-climbing (greedy) solution to the N-Queens puzzle by Yuval Baror.

Chapter 50

Local search (optimization)

In computer science, **local search** is a metaheuristic method for solving computationally hard optimization problems. Local search can be used on problems that can be formulated as finding a solution maximizing a criterion among a number of candidate solutions. Local search algorithms move from solution to solution in the space of candidate solutions (the *search space*) by applying local changes, until a solution deemed optimal is found or a time bound is elapsed.

Local search algorithms are widely applied to numerous hard computational problems, including problems from computer science (particularly artificial intelligence), mathematics, operations research, engineering, and bioinformatics. Examples of local search algorithms are WalkSAT and the 2-opt algorithm for the Traveling Salesman Problem.

50.1 Examples

Some problems where local search has been applied are:

1. The **vertex cover problem**, in which a solution is a vertex cover of a graph, and the target is to find a solution with a minimal number of nodes
2. The **travelling salesman problem**, in which a solution is a cycle containing all nodes of the graph and the target is to minimize the total length of the cycle
3. The **boolean satisfiability problem**, in which a candidate solution is a truth assignment, and the target is to maximize the number of clauses satisfied by the assignment; in this case, the final solution is of use only if it satisfies *all* clauses
4. The **nurse scheduling problem** where a solution is an assignment of nurses to shifts which satisfies all established constraints
5. The **k-medoid clustering problem** and other related facility location problems for which local search offers the best known approximation ratios from a worst-case perspective

50.2 Description

Most problems can be formulated in terms of search space and target in several different manners. For example, for the travelling salesman problem a solution can be a cycle and the criterion to maximize is a combination of the number of nodes and the length of the cycle. But a solution can also be a path, and being a cycle is part of the target.

A local search algorithm starts from a candidate solution and then iteratively moves to a neighbor solution. This is only possible if a neighborhood relation is defined on the search space. As an example, the neighborhood of a vertex cover is another vertex cover only differing by one node. For boolean satisfiability, the neighbors of a truth assignment are usually the truth assignments only differing from it by the evaluation of a variable. The same problem may have multiple different neighborhoods defined on it; local optimization with neighborhoods that involve changing up to k components of the solution is often referred to as **k-opt**.

Typically, every candidate solution has more than one neighbor solution; the choice of which one to move to is taken using only information about the solutions in the neighborhood of the current one, hence the name *local* search. When the choice of the neighbor solution is done by taking the one locally maximizing the criterion, the metaheuristic takes the name **hill climbing**. When no improving configurations are present in the neighborhood, local search is stuck at a **locally optimal point**. This local-optima problem can be cured by using restarts (repeated local search with different initial conditions), or more complex schemes based on iterations, like **iterated local search**, or memory, like **reactive search optimization**, or memory-less stochastic modifications, like **simulated annealing**.

Termination of local search can be based on a time bound. Another common choice is to terminate when the best solution found by the algorithm has not been improved in a given number of steps. Local search is an **anytime algorithm**: it can return a valid solution even if it's interrupted at any time before it ends. Local search algorithms are typically **approximation** or **incomplete** algorithms, as the

search may stop even if the best solution found by the algorithm is not optimal. This can happen even if termination is due to the impossibility of improving the solution, as the optimal solution can lie far from the neighborhood of the solutions crossed by the algorithms.

For specific problems it is possible to devise neighborhoods which are very large, possibly exponentially sized. If the best solution within the neighborhood can be found efficiently, such algorithms are referred to as very large-scale neighborhood search algorithms.

50.3 See also

Local search is a sub-field of:

- Metaheuristics
- Stochastic optimization
- Optimization

Fields within local search include:

- Hill climbing
- Simulated annealing (suited for either local or global search)
- Tabu search
- Reactive search optimization (combining machine learning and local search heuristics)

50.3.1 Real-valued search-spaces

Several methods exist for performing local search of real-valued search-spaces:

- Luus–Jaakola searches locally using a uniform distribution and an exponentially decreasing search-range.
- Random optimization searches locally using a normal distribution.
- Random search searches locally by sampling a hypersphere surrounding the current position.
- Pattern search takes steps along the axes of the search-space using exponentially decreasing step sizes.

50.4 Bibliography

- Battiti, Roberto; Mauro Brunato; Franco Mascia (2008). *Reactive Search and Intelligent Optimization*. Springer Verlag. ISBN 978-0-387-09623-0.

- Hoos, H.H. and Stutzle, T. (2005) Stochastic Local Search: Foundations and Applications, Morgan Kaufmann.
- Vijay Arya and Naveen Garg and Rohit Khandekar and Adam Meyerson and Kamesh Munagala and Vinayaka Pandit, (2004): *Local Search Heuristics for k -Median and Facility Location Problems*, Siam Journal of Computing 33(3).
- Juraj Hromkovič: *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics* (Springer)
- Wil Michiels, Emile Aarts, Jan Korst: *Theoretical Aspects of Local Search* (Springer)

Chapter 51

Simulated annealing

Simulated annealing (SA) is a generic probabilistic metaheuristic for the global optimization problem of locating a good approximation to the global optimum of a given function in a large search space. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities). For certain problems, simulated annealing may be more efficient than exhaustive enumeration — provided that the goal is merely to find an acceptably good solution in a fixed amount of time, rather than the best possible solution.

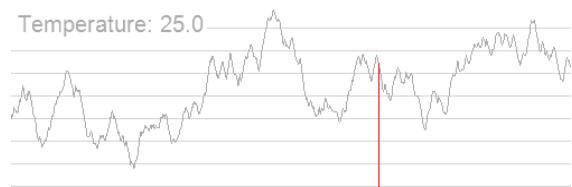
The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. Both are attributes of the material that depend on its thermodynamic free energy. Heating and cooling the material affects both the temperature and the thermodynamic free energy. While the same amount of cooling brings the same amount of decrease in temperature it will bring a bigger or smaller decrease in the thermodynamic free energy depending on the rate that it occurs, with a slower rate producing a bigger decrease.

This notion of slow cooling is implemented in the Simulated Annealing algorithm as a slow decrease in the probability of accepting worse solutions as it explores the solution space. Accepting worse solutions is a fundamental property of metaheuristics because it allows for a more extensive search for the optimal solution.

The method was independently described by Scott Kirkpatrick, C. Daniel Gelatt and Mario P. Vecchi in 1983,^[1] and by Vlado Černý in 1985.^[2] The method is an adaptation of the Metropolis-Hastings algorithm, a Monte Carlo method to generate sample states of a thermodynamic system, invented by M.N. Rosenbluth and published in a paper by N. Metropolis et al. in 1953.^[3]

51.1 Overview

The state of some physical systems, and the function $E(s)$ to be minimized is analogous to the internal energy of the system in that state. The goal is to bring the system, from an arbitrary *initial state*, to a state with the minimum possible energy.



Simulated annealing searching for a maximum. The objective here is to get to the highest point, however, it is not enough to use a simple hill climb algorithm, as there are many local maxima. By cooling the temperature slowly the global maximum is found.

51.1.1 The basic iteration

At each step, the SA heuristic considers some neighbouring state s' of the current state s , and probabilistically decides between moving the system to state s' or staying in state s . These probabilities ultimately lead the system to move to states of lower energy. Typically this step is repeated until the system reaches a state that is good enough for the application, or until a given computation budget has been exhausted.

51.1.2 The neighbours of a state

The neighbours of a state are new states of the problem that are produced after altering a given state in some well-defined way. For example, in the traveling salesman problem each state is typically defined as a permutation of the cities to be visited. The neighbours of a state are the set of permutations that are produced, for example, by reversing the order of any two successive cities. The well-defined way in which the states are altered in order to find neighbouring states is called a “move” and different moves give different sets of neighbouring states. These moves usually result in minimal alterations of the last state, as the previous example depicts, in order to help the algorithm keep the better parts of the solution and change only the worse parts. In the traveling salesman problem, the parts of the solution are the city connections.

Searching for neighbours of a state is fundamental to optimization because the final solution will come after a tour of successive neighbours. Simple heuristics move

by finding best neighbour after best neighbour and stop when they have reached a solution which has no neighbours that are better solutions. The problem with this approach is that the neighbours of a state are not guaranteed to contain any of the existing better solutions which means that failure to find a better solution among them does not guarantee that no better solution exists. This is why the best solution found by such algorithms is called a **local optimum** in contrast with the actual best solution which is called a **global optimum**. Metaheuristics use the neighbours of a solution as a way to explore the solutions space and although they prefer better neighbours they also accept worse neighbours in order to avoid getting stuck in local optima. As a result, if the algorithm is run for an infinite amount of time, the global optimum will be found.

51.1.3 Acceptance probabilities

The probability of making the **transition** from the current state s to a candidate new state s' is specified by an *acceptance probability function* $P(e, e', T)$, that depends on the energies $e = E(s)$ and $e' = E(s')$ of the two states, and on a global time-varying parameter T called the *temperature*. States with a smaller energy are better than those with a greater energy. The probability function P must be positive even when e' is greater than e . This feature prevents the method from becoming stuck at a local minimum that is worse than the global one.

When T tends to zero, the probability $P(e, e', T)$ must tend to zero if $e' > e$ and to a positive value otherwise. For sufficiently small values of T , the system will then increasingly favor moves that go “downhill” (i.e., to lower energy values), and avoid those that go “uphill.” With $T = 0$ the procedure reduces to the **greedy** algorithm, which makes only the downhill transitions.

In the original description of SA, the probability $P(e, e', T)$ was equal to 1 when $e' < e$ — i.e., the procedure always moved downhill when it found a way to do so, irrespective of the temperature. Many descriptions and implementations of SA still take this condition as part of the method’s definition. However, this condition is not essential for the method to work.

The P function is usually chosen so that the probability of accepting a move decreases when the difference $e' - e$ increases—that is, small uphill moves are more likely than large ones. However, this requirement is not strictly necessary, provided that the above requirements are met.

Given these properties, the temperature T plays a crucial role in controlling the evolution of the state s of the system with regard to its sensitivity to the variations of system energies. To be precise, for a large T , the evolution of s is sensitive to coarser energy variations, while it is sensitive to finer energy variations when T is small.

51.1.4 The annealing schedule

The name and inspiration of the algorithm demand an interesting feature related to the temperature variation to be embedded in the operational characteristics of the algorithm. This necessitates a gradual reduction of the temperature as the simulation proceeds. The algorithm starts initially with T set to a high value (or infinity), and then it is decreased at each step following some *annealing schedule*—which may be specified by the user, but must end with $T = 0$ towards the end of the allotted time budget. In this way, the system is expected to wander initially towards a broad region of the search space containing good solutions, ignoring small features of the energy function; then drift towards low-energy regions that become narrower and narrower; and finally move downhill according to the **steepest descent** heuristic.

For any given finite problem, the probability that the simulated annealing algorithm terminates with a **global optimum** solution approaches 1 as the annealing schedule is extended.^[4] This theoretical result, however, is not particularly helpful, since the time required to ensure a significant probability of success will usually exceed the time required for a **complete search** of the solution space.

51.2 Pseudocode

The following **pseudocode** presents the simulated annealing heuristic as described above. It starts from a state s_0 and continues to either a maximum of k_{\max} steps or until a state with an energy of e_{\min} or less is found. In the process, the call `neighbour(s)` should generate a randomly chosen neighbour of a given state s ; the call `random(0, 1)` should pick and return a value in the range [0, 1], uniformly at random. The annealing schedule is defined by the call `temperature(r)`, which should yield the temperature to use, given the fraction r of the time budget that has been expended so far.

- Let $s = s_0$
- For $k = 0$ through k_{\max} (exclusive):
 - $T \leftarrow \text{temperature}(k / k_{\max})$
 - Pick a random neighbour, $s_{\text{new}} \leftarrow \text{neighbour}(s)$
 - If $P(E(s), E(s_{\text{new}}), T) > \text{random}(0, 1)$, move to the new state:
 - $s \leftarrow s_{\text{new}}$
 - Output: the final state s

51.3 Selecting the parameters

In order to apply the SA method to a specific problem, one must specify the following parameters: the state

space, the energy (goal) function $E()$, the candidate generator procedure $\text{neighbour}()$, the acceptance probability function $P()$, and the annealing schedule $\text{temperature}()$ AND initial temperature $\langle \text{init temp} \rangle$. These choices can have a significant impact on the method's effectiveness. Unfortunately, there are no choices of these parameters that will be good for all problems, and there is no general way to find the best choices for a given problem. The following sections give some general guidelines.

51.3.1 Diameter of the search graph

Simulated annealing may be modeled as a random walk on a *search graph*, whose vertices are all possible states, and whose edges are the candidate moves. An essential requirement for the $\text{neighbour}()$ function is that it must provide a sufficiently short path on this graph from the initial state to any state which may be the global optimum. (In other words, the **diameter** of the search graph must be small.) In the traveling salesman example above, for instance, the search space for $n = 20$ cities has $n! = 2,432,902,008,176,640,000$ (2.4 quintillion) states; yet the neighbour generator function that swaps two consecutive cities can get from any state (tour) to any other state in at most $n(n - 1)/2 = 190$ steps (this is equivalent to $\sum_{i=1}^{n-1} i$).

51.3.2 Transition probabilities

To investigate the behavior of simulated annealing on a particular problem, it can be useful to consider the *transition probabilities* that result from the various design choices made in the implementation of the algorithm. For each edge (s, s') of the search graph, the transition probability is defined as the probability that the SA algorithm will move to state s' when its current state is s . This probability depends on the current temperature as specified by $\text{temp}()$, on the order in which the candidate moves are generated by the $\text{neighbour}()$ function, and on the acceptance probability function $P()$. (Note that the transition probability is **not** simply $P(e, e', T)$, because the candidates are tested serially.)

51.3.3 Acceptance probabilities

The specification of $\text{neighbour}()$, $P()$, and $\text{temperature}()$ is partially redundant. In practice, it's common to use the same acceptance function $P()$ for many problems, and adjust the other two functions according to the specific problem.

In the formulation of the method by Kirkpatrick et al., the acceptance probability function $P(e, e', T)$ was defined as 1 if $e' < e$, and $\exp(-(e' - e)/T)$ otherwise. This formula was superficially justified by analogy with the transitions of a physical system; it corresponds to the

Metropolis-Hastings algorithm, in the case where $T=1$ and the proposal distribution of Metropolis-Hastings is symmetric. However, this acceptance probability is often used for simulated annealing even when the $\text{neighbour}()$ function, which is analogous to the proposal distribution in Metropolis-Hastings, is not symmetric, or not probabilistic at all. As a result, the transition probabilities of the simulated annealing algorithm do not correspond to the transitions of the analogous physical system, and the long-term distribution of states at a constant temperature T need not bear any resemblance to the thermodynamic equilibrium distribution over states of that physical system, at any temperature. Nevertheless, most descriptions of SA assume the original acceptance function, which is probably hard-coded in many implementations of SA.

51.3.4 Efficient candidate generation

When choosing the candidate generator $\text{neighbour}()$, one must consider that after a few iterations of the SA algorithm, the current state is expected to have much lower energy than a random state. Therefore, as a general rule, one should skew the generator towards candidate moves where the energy of the destination state s' is likely to be similar to that of the current state. This heuristic (which is the main principle of the **Metropolis-Hastings algorithm**) tends to exclude “very good” candidate moves as well as “very bad” ones; however, the former are usually much less common than the latter, so the heuristic is generally quite effective.

In the traveling salesman problem above, for example, swapping two *consecutive* cities in a low-energy tour is expected to have a modest effect on its energy (length); whereas swapping two *arbitrary* cities is far more likely to increase its length than to decrease it. Thus, the consecutive-swap neighbour generator is expected to perform better than the arbitrary-swap one, even though the latter could provide a somewhat shorter path to the optimum (with $n - 1$ swaps, instead of $n(n - 1)/2$).

A more precise statement of the heuristic is that one should try first candidate states s' for which $P(E(s), E(s'), T)$ is large. For the “standard” acceptance function P above, it means that $E(s') - E(s)$ is on the order of T or less. Thus, in the traveling salesman example above, one could use a neighbour() function that swaps two random cities, where the probability of choosing a city pair vanishes as their distance increases beyond T .

51.3.5 Barrier avoidance

When choosing the candidate generator $\text{neighbour}()$ one must also try to reduce the number of “deep” local minima — states (or sets of connected states) that have much lower energy than all its neighbouring states. Such “closed catchment basins” of the energy function may

trap the SA algorithm with high probability (roughly proportional to the number of states in the basin) and for a very long time (roughly exponential on the energy difference between the surrounding states and the bottom of the basin).

As a rule, it is impossible to design a candidate generator that will satisfy this goal and also prioritize candidates with similar energy. On the other hand, one can often vastly improve the efficiency of SA by relatively simple changes to the generator. In the traveling salesman problem, for instance, it is not hard to exhibit two tours A , B , with nearly equal lengths, such that (1) A is optimal, (2) every sequence of city-pair swaps that converts A to B goes through tours that are much longer than both, and (3) A can be transformed into B by flipping (reversing the order of) a set of consecutive cities. In this example, A and B lie in different “deep basins” if the generator performs only random pair-swaps; but they will be in the same basin if the generator performs random segment-flips.

51.3.6 Cooling schedule

The physical analogy that is used to justify SA assumes that the cooling rate is low enough for the probability distribution of the current state to be near **thermodynamic equilibrium** at all times. Unfortunately, the *relaxation time*—the time one must wait for the equilibrium to be restored after a change in temperature—strongly depends on the “topography” of the energy function and on the current temperature. In the SA algorithm, the relaxation time also depends on the candidate generator, in a very complicated way. Note that all these parameters are usually provided as **black box** functions to the SA algorithm. Therefore, the ideal cooling rate cannot be determined beforehand, and should be empirically adjusted for each problem. Adaptive simulated annealing algorithms address this problem by connecting the cooling schedule to the search progress.

51.4 Restarts

Sometimes it is better to move back to a solution that was significantly better rather than always moving from the current state. This process is called *restarting* of simulated annealing. To do this we set s and e to s_{best} and e_{best} and perhaps restart the annealing schedule. The decision to restart could be based on several criteria. Notable among these include restarting based on a fixed number of steps, based on whether the current energy is too high compared to the best energy obtained so far, restarting randomly, etc.

51.5 Related methods

- **Quantum annealing** uses “quantum fluctuations” instead of thermal fluctuations to get through high but thin barriers in the target function.
- **Stochastic tunneling** attempts to overcome the increasing difficulty simulated annealing runs have in escaping from local minima as the temperature decreases, by ‘tunneling’ through barriers.
- **Tabu search** normally moves to neighbouring states of lower energy, but will take uphill moves when it finds itself stuck in a local minimum; and avoids cycles by keeping a “taboo list” of solutions already seen.
- **Reactive search optimization** focuses on combining machine learning with optimization, by adding an internal feedback loop to self-tune the free parameters of an algorithm to the characteristics of the problem, of the instance, and of the local situation around the current solution.
- **Stochastic gradient descent** runs many greedy searches from random initial locations.
- **Genetic algorithms** maintain a pool of solutions rather than just one. New candidate solutions are generated not only by “mutation” (as in SA), but also by “recombination” of two solutions from the pool. Probabilistic criteria, similar to those used in SA, are used to select the candidates for mutation or combination, and for discarding excess solutions from the pool.
- **Graduated optimization** digressively “smooths” the target function while optimizing.
- **Ant colony optimization (ACO)** uses many ants (or agents) to traverse the solution space and find locally productive areas.
- The **cross-entropy method (CE)** generates candidates solutions via a parameterized probability distribution. The parameters are updated via cross-entropy minimization, so as to generate better samples in the next iteration.
- **Harmony search** mimics musicians in improvisation process where each musician plays a note for finding a best harmony all together.
- **Stochastic optimization** is an umbrella set of methods that includes simulated annealing and numerous other approaches.

- Particle swarm optimization is an algorithm modelled on swarm intelligence that finds a solution to an optimization problem in a search space, or model and predict social behavior in the presence of objectives.
- Intelligent Water Drops (IWD) which mimics the behavior of natural water drops to solve optimization problems
- Parallel tempering is a simulation of model copies at different temperatures (or Hamiltonians) to overcome the potential barriers.

51.6 See also

- Adaptive simulated annealing
- Markov chain
- Combinatorial optimization
- Automatic label placement
- Multidisciplinary optimization
- Place and route
- Molecular dynamics
- Traveling salesman problem
- Reactive search optimization
- Graph cuts in computer vision
- Particle swarm optimization
- Intelligent Water Drops

51.7 References

- [1] Kirkpatrick, S.; Gelatt Jr, C. D.; Vecchi, M. P. (1983). "Optimization by Simulated Annealing". *Science* **220** (4598): 671–680. Bibcode:1983Sci...220..671K. doi:10.1126/science.220.4598.671. JSTOR 1690046. PMID 17813860.
- [2] Černý, V. (1985). "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm". *Journal of Optimization Theory and Applications* **45**: 41–51. doi:10.1007/BF00940812.
- [3] Metropolis, Nicholas; Rosenbluth, Arianna W.; Rosenbluth, Marshall N.; Teller, Augusta H.; Teller, Edward (1953). "Equation of State Calculations by Fast Computing Machines". *The Journal of Chemical Physics* **21** (6): 1087. Bibcode:1953JChPh..21.1087M. doi:10.1063/1.1699114.
- [4] Granville, V.; Krivanek, M.; Rasson, J.-P. (1994). "Simulated annealing: A proof of convergence". *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16** (6): 652–656. doi:10.1109/34.295910.

51.8 Further reading

- A. Das and B. K. Chakrabarti (Eds.), *Quantum Annealing and Related Optimization Methods*, Lecture Note in Physics, Vol. 679, Springer, Heidelberg (2005)
- Weinberger, E. (1990). "Correlated and uncorrelated fitness landscapes and how to tell the difference". *Biological Cybernetics* **63** (5): 325–336. doi:10.1007/BF00202749.
- De Vicente, Juan; Lanchares, Juan; Hermida, Román (2003). "Placement by thermodynamic simulated annealing". *Physics Letters A* **317** (5–6): 415–423. Bibcode:2003PhLA..317..415D. doi:10.1016/j.physleta.2003.08.070.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Section 10.12. Simulated Annealing Methods". *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8

51.9 External links

- Simulated Annealing visualization A visualization of a simulated annealing solution to the N-Queens puzzle by Yuval Baror.
- Global optimization algorithms for MATLAB
- Simulated Annealing A Java applet that allows you to experiment with simulated annealing. Source code included.
- "General Simulated Annealing Algorithm" An open-source MATLAB program for general simulated annealing exercises.
- Self-Guided Lesson on Simulated Annealing A Wikiversity project.
- Google in superposition of using, not using quantum computer Ars Technica discusses the possibility that the D-Wave computer being used by google may, in fact, be an efficient SA co-processor
- Minimizing Multimodal Functions of Continuous Variables with Simulated Annealing A Fortran 77 simulated annealing code.
- Simulated Annealing Tutorial and Code in Python and MATLAB
- Simulated Annealing in R via optim or GenSA

Chapter 52

Tabu search

Tabu search, created by Fred W. Glover in 1986^[1] and formalized in 1989,^{[2][3]} is a metaheuristic search method employing local search methods used for mathematical optimization.

Local (neighborhood) searches take a potential solution to a problem and check its immediate neighbors (that is, solutions that are similar except for one or two minor details) in the hope of finding an improved solution. Local search methods have a tendency to become stuck in sub-optimal regions or on plateaus where many solutions are equally fit.

Tabu search enhances the performance of local search by relaxing its basic rule. First, at each step *worsening* moves can be accepted if no improving move is available (like when the search is stuck at a strict local minimum). In addition, *prohibitions* (henceforth the term *tabu*) are introduced to discourage the search from coming back to previously-visited solutions.

The implementation of tabu search uses memory structures that describe the visited solutions or user-provided sets of rules.^[2] If a potential solution has been previously visited within a certain short-term period or if it has violated a rule, it is marked as "tabu" (forbidden) so that the algorithm does not consider that possibility repeatedly.

52.1 Background

The word *tabu* comes from *Tongan*, a language of Polynesia, used by the aborigines of Tonga to indicate things that cannot be touched because they are sacred.^[4]

Tabu search (TS) is a metaheuristic algorithm that can be used for solving *combinatorial optimization* problems (problems where an optimal ordering and selection of options is desired).

Current applications of TS span the areas of *resource planning*, *telecommunications*, *VLSI design*, *financial analysis*, *scheduling*, *space planning*, *energy distribution*, *molecular engineering*, *logistics*, *pattern classification*, *flexible manufacturing*, *waste management*, *mineral exploration*, *biomedical analysis*, *environmental conservation* and scores of others. In recent years, journals in a wide variety of fields have published tutorial articles

and computational studies documenting successes by tabu search in extending the frontier of problems that can be handled effectively — yielding solutions whose quality often significantly surpasses that obtained by methods previously applied. A comprehensive list of applications, including summary descriptions of gains achieved from practical implementations, can be found in^[5] Recent TS developments and applications can also be found in Tabu Search Vignettes.

52.2 Basic Description

Tabu search uses a local or neighborhood search procedure to iteratively move from one potential solution x to an improved solution x' in the neighborhood of x , until some stopping criterion has been satisfied (generally, an attempt limit or a score threshold). Local search procedures often become stuck in poor-scoring areas or areas where scores plateau. In order to avoid these pitfalls and explore regions of the *search space* that would be left unexplored by other local search procedures, tabu search carefully explores the neighborhood of each solution as the search progresses. The solutions admitted to the new neighborhood, $N^*(x)$, are determined through the use of memory structures. Using these memory structures, the search progresses by iteratively moving from the current solution x to an improved solution x' in $N^*(x)$.

These memory structures form what is known as the tabu list, a set of rules and banned solutions used to filter which solutions will be admitted to the neighborhood $N^*(x)$ to be explored by the search. In its simplest form, a tabu list is a short-term set of the solutions that have been visited in the recent past (less than n iterations ago, where n is the number of previous solutions to be stored - is also called the tabu tenure). More commonly, a tabu list consists of solutions that have changed by the process of moving from one solution to another. It is convenient, for ease of description, to understand a "solution" to be coded and represented by such attributes.

52.3 Types of Memory

The memory structures used in tabu search can roughly be divided into three categories:^[6]

- Short-term: The list of solutions recently considered. If a potential solution appears on the tabu list, it cannot be revisited until it reaches an expiration point.
- Intermediate-term: Intensification rules intended to bias the search towards promising areas of the search space.
- Long-term: Diversification rules that drive the search into new regions (i.e. regarding resets when the search becomes stuck in a plateau or a suboptimal dead-end).

Short-term, intermediate-term and long-term memories can overlap in practice. Within these categories, memory can further be differentiated by measures such as frequency and impact of changes made. One example of an intermediate-term memory structure is one that prohibits or encourages solutions that contain certain attributes (e.g., solutions which include undesirable or desirable values for certain variables) or a memory structure that prevents or induces certain moves (e.g. based on frequency memory applied to solutions sharing features in common with unattractive or attractive solutions found in the past). In short-term memory, selected attributes in solutions recently visited are labeled “tabu-active.” Solutions that contain tabu-active elements are banned. Aspiration criteria are employed that override a solution’s tabu state, thereby including the otherwise-excluded solution in the allowed set (provided the solution is “good enough” according to a measure of quality or diversity). A simple and commonly used aspiration criterion is to allow solutions which are better than the currently-known best solution.

Short-term memory alone may be enough to achieve solution superior to those found by conventional local search methods, but intermediate and long-term structures are often necessary for solving harder problems.^[7] Tabu search is often benchmarked against other metaheuristic methods - such as Simulated annealing, genetic algorithms, Ant colony optimization algorithms, Reactive search optimization, Guided Local Search, or greedy randomized adaptive search. In addition, tabu search is sometimes combined with other metaheuristics to create hybrid methods. The most common tabu search hybrid arises by joining TS with Scatter Search,^{[8][9]} a class of population-based procedures which has roots in common with tabu search, and is often employed in solving large non-linear optimization problems.

52.4 Pseudocode

The following pseudocode presents a simplified version of the tabu search algorithm as described above. This implementation has a rudimentary short-term memory,

but contains no intermediate or long-term memory structures. The term “fitness” refers to an evaluation of the candidate solution, as embodied in an objective function for mathematical optimization.

```

01: s ← s0
02: sBest ← s
03: tabuList ← []
04: while (not stoppingCondition())
05:   candidateList ← []
06:   bestCandidate ← null
07:   for (sCandidate in sNeighborhood)
08:     if ( (not tabuList.contains(sCandidate)) and (fitness(sCandidate) > fitness(bestCandidate)) )
09:       bestCandidate ← sCandidate
10:   end
11: end
12: s ← bestCandidate
13: if (fitness(sCandidate) > fitness(sBest))
14:   sBest ← sCandidate
15: end
16: tabuList.push(sCandidate);
17: if (tabuList.size > maxTabuSize)
18:   tabuList.removeFirst()
19: end
20: end
21: return sBest

```

Lines 1-3 represent some initial setup, respectively creating an initial solution (possibly chosen at random), setting that initial solution as the best seen to date, and initializing an empty tabu list. In this example, the tabu list is simply a short term memory structure that will contain a record of the elements of the states visited.

The proper algorithm starts in line 4. This loop will continue searching for an optimal solution until a user-specified stopping condition is met (two examples of such conditions are a simple time limit or a threshold on the fitness score). In line 5, an empty candidate list is initialized. The neighboring solutions are checked for tabu elements in line 7. In line 8 we look for the best solution in the neighborhood, that is not tabu.

The fitness function is generally a mathematical function, which returns a score or the aspiration criteria is satisfied - for example, a aspiration criteria could be considered as a new search space is found^[10]). If the best local candidate has a higher fitness value than the current best (line 13), it is set as the new best (line 14). The local best candidate is always added to the tabu list (line 16) and if the tabu list is full (line 17), some elements will be allowed to expire (line 18). Generally, elements expire from the list in the same order they are added. The procedure will select the best local candidate (although it has worse fitness than the sBest) in order to escape the local optimal.

This process continues until the user specified stopping criterion is met, at which point, the best solution seen during the search process is returned (line 21).

52.5 Example: Traveling salesman problem

The traveling salesman problem (TSP) is sometimes used to show the functionality of tabu search.^[7] This problem poses a straightforward question – given a list of cities, what is the shortest route that visits every city? For example, if city A and city B are next to each other, while city C is farther away, the total distance traveled will be shorter if cities A and B are visited one after the other before visiting city C. Since finding an optimal solution is NP-hard, heuristic-based approximation methods (such as local searches) are useful for devising close-to-optimal solutions. To obtain good TSP solutions, it is essential to exploit the graph structure. The value of exploiting problem structure is a recurring theme in metaheuristic methods, and tabu search is well-suited to this. A class of strategies associated with tabu search called ejection chain methods has made it possible to obtain high-quality TSP solutions efficiently^[11]

On the other hand, a simple tabu search can be used to find a **satisficing** solution for the traveling salesman problem (that is, a solution that satisfies an adequacy criterion, although not with the high quality obtained by exploiting the graph structure). The search starts with an initial solution, which can be generated randomly or according to some sort of **nearest neighbor algorithm**. To create new solutions, the order that two cities are visited in a potential solution is swapped. The total traveling distance between all the cities is used to judge how ideal one solution is compared to another. To prevent cycles – i.e., repeatedly visiting a particular set of solutions – and to avoid becoming stuck in local optima, a solution is added to the tabu list if it is accepted into the solution neighborhood, $N^*(x)$.

New solutions are created until some stopping criterion, such as an arbitrary number of iterations, is met. Once the simple tabu search stops, it returns the best solution found during its execution.

52.6 References

- [1] Fred Glover (1986). “Future Paths for Integer Programming and Links to Artificial Intelligence”. *Computers and Operations Research* **13** (5): 533–549. doi:10.1016/0305-0548(86)90048-1.
- [2] Fred Glover (1989). “Tabu Search - Part 1”. *ORSA Journal on Computing* **1** (2): 190–206. doi:10.1287/ijoc.1.3.190.
- [3] Fred Glover (1990). “Tabu Search - Part 2”. *ORSA Journal on Computing* **2** (1): 4–32. doi:10.1287/ijoc.2.1.4.
- [4] http://www.ise.ncsu.edu/fangroup/ie789.dir/IE789F_tabu.pdf

- [5] F. Glover, M. Laguna (1997). *Tabu Search*. Kluwer Academic Publishers.
- [6] Fred Glover (1990). “Tabu Search: A Tutorial”. *Interfaces*.
- [7] M. Malek, M. Huruswamy, H. Owens, M. Pandya (1989). *Serial and parallel search techniques for the traveling salesman problem*. *Annals of OR: Linkages with Artificial Intelligence*.
- [8] F. Glover, M. Laguna and R. Marti (2000). *Fundamentals of Scatter Search and Path Relinking*. *Control and Cybernetics* **29** (3). pp. 653–684.
- [9] M. Laguna and R. Marti (2003). *Scatter Search: Methodology and Implementations in C*. Kluwer Academic Publishers, Boston.
- [10] http://www.ise.ncsu.edu/fangroup/ie789.dir/IE789F_tabu.pdf
- [11] D. Gamboa, C. Rego and F. Glover (2005). “Data Structures and Ejection Chains for Solving Large Scale Traveling Salesman Problems”. *European Journal of Operational Research* **160** (1): 154–171. doi:10.1016/j.ejor.2004.04.023.

52.7 External links

- Visualization of the Tabu search algorithm (Applet)
- Metaheuristic International Conference (MIC 2011) - Udine
- The Reactive Search Community
- LION Conference on Learning and Intelligent Optimization techniques
-

Chapter 53

Active set method

“Active Set” redirects here. For the Wikipedia article on the band, see [The Active Set](#).

In mathematical optimization, a problem is defined using an objective function to minimize or maximize, and a set of constraints

$$g_1(x) \geq 0, \dots, g_k(x) \geq 0$$

that define the **feasible region**, that is, the set of all x to search for the optimal solution. Given a point x in the feasible region, a constraint

$$g_i(x) \geq 0$$

is called **active** at x if $g_i(x) = 0$ and **inactive** at x if $g_i(x) > 0$. Equality constraints are always active. The **active set** at x is made up of those constraints $g_i(x)$ that are active at the current point (Nocedal & Wright 2006, p. 308).

The active set is particularly important in optimization theory as it determines which constraints will influence the final result of optimization. For example, in solving the linear programming problem, the active set gives the hyperplanes that intersect at the solution point. In quadratic programming, as the solution is not necessarily on one of the edges of the bounding polygon, an estimation of the active set gives us a subset of inequalities to watch while searching the solution, which reduces the complexity of the search.

53.1 Active set methods

In general an active set algorithm has the following structure:

Find a feasible starting point

repeat until “optimal enough”

solve the equality problem defined by the active set (approximately)

compute the Lagrange multipliers of the active set

remove a subset of the constraints with negative Lagrange multipliers

search for infeasible constraints

end repeat

Methods that can be described as **active set methods** include:

- Successive linear programming (SLP)
- Sequential quadratic programming (SQP)
- Sequential linear-quadratic programming (SLQP)
- Reduced gradient method (RG)
- Generalized reduced gradient method (GRG)

53.2 References

- Murty, K. G. (1988). *Linear complementarity, linear and nonlinear programming*. Sigma Series in Applied Mathematics 3. Berlin: Heldermann Verlag. pp. xlvi+629 pp. ISBN 3-88538-403-5. MR 949214
- Nocedal, Jorge; Wright, Stephen J. (2006). *Numerical Optimization* (2nd ed.). Berlin, New York: Springer-Verlag. ISBN 978-0-387-30303-1..

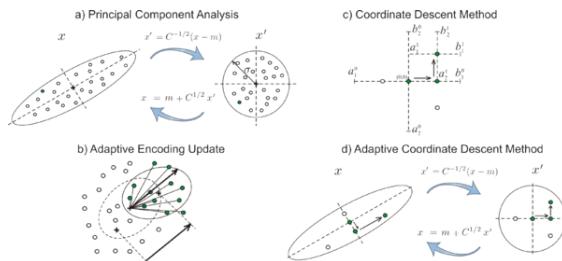
Chapter 54

Adaptive coordinate descent

Adaptive coordinate descent is an extension of the coordinate descent algorithm to non-separable optimization. The adaptive coordinate descent approach gradually builds a transformation of the coordinate system such that the new coordinates are as decorrelated as possible with respect to the objective function. The adaptive coordinate descent was shown to be competitive to the state-of-the-art **evolutionary algorithms** and has the following invariance properties.^[1]

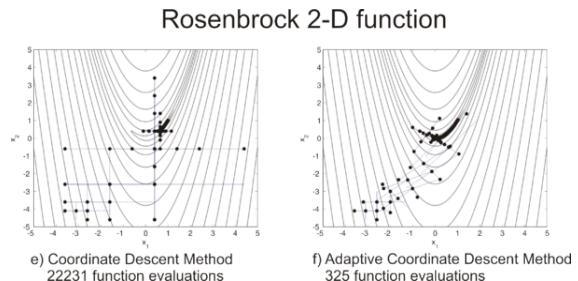
1. Invariance with respect to monotonous transformations of the function (scaling)
2. Invariance with respect to orthogonal transformations of the search space (rotation).

CMA-like Adaptive Encoding Update (b) mostly based on **principal component analysis** (a) is used to extend the coordinate descent method (c) to the optimization of non-separable problems (d).



The adaptation of an appropriate coordinate system allows adaptive coordinate descent to outperform coordinate descent on non-separable functions. The following figure illustrates the convergence of both algorithms on 2-dimensional Rosenbrock function up to a target function value 10^{-10} , starting from the initial point $x_0 = (-3, -4)$.

The adaptive coordinate descent method reaches the target value after only 325 function evaluations (about 70 times faster than coordinate descent), that is comparable to **gradient-based methods**. The algorithm has linear time complexity if update coordinate system every D iterations, it is also suitable for large-scale ($D \gg 100$) non-linear optimization.



54.1 Relevant approaches

First approaches to optimization using adaptive coordinate system were proposed already in the 1960s (see, e.g., Rosenbrock's method). PRincipal Axis (PRAXIS) algorithm, also referred to as Brent's algorithm, is an derivative-free algorithm which assumes quadratic form of the optimized function and repeatedly updates a set of conjugate search directions.^[2] The algorithm, however, is not invariant to scaling of the objective function and may fail under its certain rank-preserving transformations (e.g., will lead to a non-quadratic shape of the objective function). A recent analysis of PRAXIS can be found in .^[3] For practical applications see,^[4] where an adaptive coordinate descent approach with step-size adaptation and local coordinate system rotation was proposed for robot-manipulator path planning in 3D space with static polygonal obstacles.

54.2 See also

- Coordinate descent
- CMA-ES
- Rosenbrock methods
- Mathematical optimization

54.3 References

- [1] Loshchilov, I.; M. Schoenauer; M. Sebag (2011). “Adaptive Coordinate Descent”. *Genetic and Evolution-*

- ary Computation Conference (GECCO). ACM Press. pp. 885—892.
- [2] Brent, R.P. (1972). *Algorithms for minimization without derivatives*. Prentice-Hall.
- [3] Ali, U.; Kickmeier-Rust, M.D. (2008). “Implementation and Applications of a Three-Round User Strategy for Improved Principal Axis Minimization”. *Journal of Applied Quantitative Methods*. pp. 505—513.
- [4] Pavlov, D. (2006). “Manipulator path planning in 3-dimensional space”. *Computer Science--Theory and Applications*. Springer. pp. 505—513.

54.4 External links

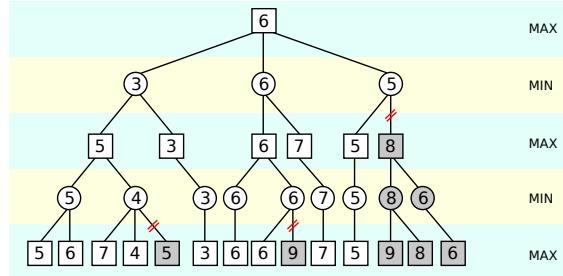
- **SOURCE CODE ACD** ACD is a MATLAB source code for Adaptive Coordinate Descent

Chapter 55

Alpha–beta pruning

For other uses, see [Alphabeta \(disambiguation\)](#).

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess, Go, etc.). It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.^[1]



An illustration of alpha–beta pruning. The grayed-out subtrees need not be explored (when moves are evaluated from left to right), since we know the group of subtrees as a whole yields the value of an equivalent subtree or worse, and as such cannot influence the final result. The max and min levels represent the turn of the player and the adversary, respectively.

55.1 History

Allen Newell and Herbert A. Simon who used what John McCarthy calls an “approximation”^[2] in 1958 wrote that alpha–beta “appears to have been reinvented a number of times”.^[3] Arthur Samuel had an early version and Richards, Hart, Levine and/or Edwards found alpha–beta independently in the United States.^[4] McCarthy proposed similar ideas during the Dartmouth Conference in 1956 and suggested it to a group of his students including Alan Kotok at MIT in 1961.^[5] Alexander Brudno independently discovered the alpha–beta algorithm, publishing his results in 1963.^[6] Donald Knuth and Ronald W. Moore refined the algorithm in 1975^{[7][8]} and Judea Pearl proved its optimality in 1982.^[9]

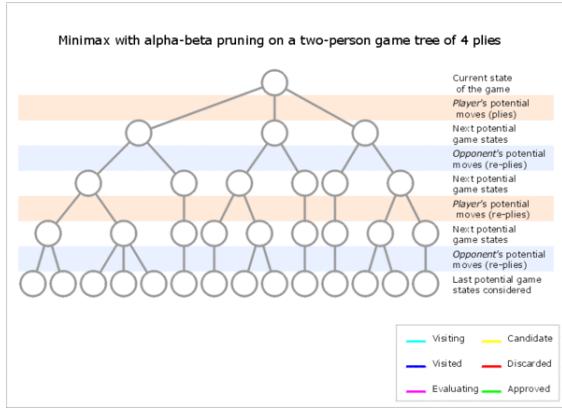
55.2 Improvements over naive minimax

The benefit of alpha–beta pruning lies in the fact that branches of the search tree can be eliminated. This way, the search time can be limited to the ‘more promising’ subtree, and a deeper search can be performed in the same time. Like its predecessor, it belongs to the branch and bound class of algorithms. The optimization reduces the effective depth to slightly more than half that of sim-

ple minimax if the nodes are evaluated in an optimal or near optimal order (best choice for side on move ordered first at each node).

With an (average or constant) branching factor of b , and a search depth of d plies, the maximum number of leaf node positions evaluated (when the move ordering is pessimistic) is $O(b*b*...*b) = O(b^d)$ – the same as a simple minimax search. If the move ordering for the search is optimal (meaning the best moves are always searched first), the number of leaf node positions evaluated is about $O(b*1*b*1*...*b)$ for odd depth and $O(b*1*b*1*...*1)$ for even depth, or $O(b^{d/2}) = O(\sqrt{b^d})$. In the latter case, where the ply of a search is even, the effective branching factor is reduced to its square root, or, equivalently, the search can go twice as deep with the same amount of computation.^[10] The explanation of $b*1*b*1*...$ is that all the first player’s moves must be studied to find the best one, but for each, only the best second player’s move is needed to refute all but the first (and best) first player move—alpha–beta ensures no other second player moves need be considered. When nodes are ordered at random, the average number of nodes evaluated is roughly $O(b^{3d/4})$.^[2]

Normally during alpha–beta, the subtrees are temporarily dominated by either a first player advantage (when many first player moves are good, and at each search depth the first move checked by the first player is adequate, but all



An animated pedagogical example that attempts to be human-friendly by substituting initial infinite (or arbitrarily large) values for emptiness and by avoiding using the negamax coding simplifications.

second player responses are required to try to find a refutation), or vice versa. This advantage can switch sides many times during the search if the move ordering is incorrect, each time leading to inefficiency. As the number of positions searched decreases exponentially each move nearer the current position, it is worth spending considerable effort on sorting early moves. An improved sort at any depth will exponentially reduce the total number of positions searched, but sorting all positions at depths near the root node is relatively cheap as there are so few of them. In practice, the move ordering is often determined by the results of earlier, smaller searches, such as through iterative deepening.

The algorithm maintains two values, alpha and beta, which represent the maximum score that the maximizing player is assured of and the minimum score that the minimizing player is assured of respectively. Initially alpha is negative infinity and beta is positive infinity, i.e. both players start with their lowest possible score. It can happen that when choosing a certain branch of a certain node the minimum score that the minimizing player is assured of becomes less than the maximum score that the maximizing player is assured of ($\text{beta} \leq \text{alpha}$). If this is the case, the parent node should not choose this node, because it will make the score for the parent node worse. Therefore, the other branches of the node do not have to be explored.

Additionally, this algorithm can be trivially modified to return an entire principal variation in addition to the score. Some more aggressive algorithms such as MTD(f) do not easily permit such a modification.

55.3 Pseudocode

The pseudo-code for the fail-soft variation of alpha-beta pruning is as follows:^[10]

```

01 function alphabeta(node, depth,  $\alpha$ ,  $\beta$ , maximizingPlayer)
02   if depth = 0 or node is a terminal node return the heuristic value of node
03   else  $v := -\infty$ 
04   for each child of node do
05      $v := \max(v, \text{alphabeta(child, depth - 1, } \alpha, \beta, \text{ FALSE}))$ 
06      $\alpha := \max(\alpha, v)$ 
07     if  $\beta \leq \alpha$  break (*  $\beta$  cut-off *)
08   return  $v$ 
09   else  $v := \infty$ 
10   for each child of node do
11      $v := \min(v, \text{alphabeta(child, depth - 1, } \alpha, \beta, \text{ TRUE}))$ 
12      $\beta := \min(\beta, v)$ 
13     if  $\beta \leq \alpha$  break (*  $\alpha$  cut-off *)
14   return  $v$  (* Initial call *)
15 Initial call alphabeta(origin, depth,  $-\infty, +\infty$ , TRUE)

```

With fail-soft alpha-beta, the alphabeta function may return values (v) that exceed ($v < \alpha$ or $v > \beta$) the α and β bounds set by its function call arguments. In comparison, fail-hard alpha-beta limits its function return value into the inclusive range of α and β .

55.4 Heuristic improvements

Further improvement can be achieved without sacrificing accuracy, by using ordering heuristics to search parts of the tree that are likely to force alpha-beta cutoffs early. For example, in chess, moves that take pieces may be examined before moves that do not, or moves that have scored highly in earlier passes through the game-tree analysis may be evaluated before others. Another common, and very cheap, heuristic is the killer heuristic, where the last move that caused a beta-cutoff at the same level in the tree search is always examined first. This idea can also be generalized into a set of refutation tables.

Alpha-beta search can be made even faster by considering only a narrow search window (generally determined by guesswork based on experience). This is known as *aspiration search*. In the extreme case, the search is performed with alpha and beta equal; a technique known as *zero-window search*, *null-window search*, or *scout search*. This is particularly useful for win/loss searches near the end of a game where the extra depth gained from the narrow window and a simple win/loss evaluation function may lead to a conclusive result. If an aspiration search fails, it is straightforward to detect whether it failed *high* (high edge of window was too low) or *low* (lower edge of window was too high). This gives information about what window values might be useful in a re-search of the position.

Over time, other improvements have been suggested, and indeed the Falphabeta (fail-soft alpha-beta) idea of Fishburn is nearly universal and is already incorporated above in a slightly modified form. Fishburn also suggested a combination of the killer heuristic and zero-window search under the name Lalphabeta (“last move with minimal window alpha-beta search”.)

55.5 Other algorithms

More advanced algorithms that are even faster while still being able to compute the exact minimax value are known, such as SCOUT,^[11] Negascout and MTD-f.

Since the minimax algorithm and its variants are inherently depth-first, a strategy such as iterative deepening is usually used in conjunction with alpha–beta so that a reasonably good move can be returned even if the algorithm is interrupted before it has finished execution. Another advantage of using iterative deepening is that searches at shallower depths give move-ordering hints, as well as shallow alpha and beta estimates, that both can help produce cutoffs for higher depth searches much earlier than would otherwise be possible.

Algorithms like SSS*, on the other hand, use the best-first strategy. This can potentially make them more time-efficient, but typically at a heavy cost in space-efficiency.^[12]

55.6 See also

- Pruning (algorithm)
- Branch and bound
- Minimax
- Combinatorial optimization
- Negamax
- Transposition table

55.7 References

- George T. Heineman, Gary Pollice, and Stanley Selkow (2008). “Chapter 7: Path Finding in AI”. *Algorithms in a Nutshell*. O'Reilly Media. pp. 217–223. ISBN 978-0-596-51624-6.
- Judea Pearl, *Heuristics*, Addison-Wesley, 1984
- John P. Fishburn (1984). “Appendix A: Some Optimizations of α - β Search”. *Analysis of Speedup in Distributed Algorithms (revision of 1981 PhD thesis)*. UMI Research Press. pp. 107–111. ISBN 0-8357-1527-2.

- [1] Russell, Stuart J.; Norvig, Peter (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.). Upper Saddle River, New Jersey: Pearson Education, Inc. p. 167. ISBN 0-13-604259-7
- [2] McCarthy, John (LaTeX2HTML 27 November 2006). “Human Level AI Is Harder Than It Seemed in 1955”. Retrieved 2006-12-20. Check date values in: ldate= (help)

- [3] Newell, Allen and Herbert A. Simon (March 1976). “Computer Science as Empirical Inquiry: Symbols and Search” (PDF). *Communications of the ACM* **19** (3). Retrieved 2006-12-21.
- [4] Edwards, D.J. and Hart, T.P. (4 December 1961 to 28 October 1963). “The Alpha–beta Heuristic (AIM-030)”. Massachusetts Institute of Technology. Retrieved 2006-12-21. Check date values in: ldate= (help)
- [5] Kotok, Alan (XHTML 3 December 2004). “MIT Artificial Intelligence Memo 41”. Retrieved 2006-07-01. Check date values in: ldate= (help)
- [6] Marsland, T.A. (May 1987). “Computer Chess Methods (PDF) from Encyclopedia of Artificial Intelligence. S. Shapiro (editor)” (PDF). J. Wiley & Sons. pp. 159–171. Retrieved 2006-12-21.
- [7] • Knuth, D. E., and Moore, R. W. (1975). “An Analysis of Alpha–Beta Pruning”. *Artificial Intelligence* **6** (4): 293–326. doi:10.1016/0004-3702(75)90019-3.
- Reprinted as Chapter 9 in Knuth, Donald E. (2000). *Selected Papers on Analysis of Algorithms*. Stanford, California: Center for the Study of Language and Information - CSLI Lecture Notes, no. 102. ISBN 1-57586-212-3. OCLC 222512366.
- [8] Abramson, Bruce (June 1989). “Control Strategies for Two-Player Games”. *ACM Computing Surveys* **21** (2): 137. doi:10.1145/66443.66444. Retrieved 2008-08-20.
- [9] Pearl, Judea (August 1982). “The Solution for the Branching Factor of the Alpha–beta Pruning Algorithm and its Optimality”. *Communications of the ACM* **25** (8): 559–564. doi:10.1145/358589.358616.
- [10] Russell, Stuart J.; Norvig, Peter (2003), *Artificial Intelligence: A Modern Approach* (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2
- [11] Pearl, J., “SCOUT: A Simple Game-Searching Algorithm With Proven Optimal Properties,” *Proceedings of the First Annual National Conference on Artificial Intelligence*, Stanford University, August 18–21, 1980, pp. 143–145.
- [12] Pearl, Judea; Korf, Richard (1987). “Search techniques”, *Annual Review of Computer Science* **2**: 451–467, doi:10.1146/annurev.cs.02.060187.002315, Like its A* counterpart for single-player games, SSS* is optimal in terms of the average number of nodes examined; but its superior pruning power is more than offset by the substantial storage space and bookkeeping required.

55.8 External links

- http://www.emunix.emich.edu/~{ }evett/AI/AlphaBeta_movie/sld001.htm

- http://sern.ucalgary.ca/courses/CPSC/533/W99/presentations/L1_5B_McCullough_Melnyk/
- http://sern.ucalgary.ca/courses/CPSC/533/W99/presentations/L2_5B_Lima_Neitz/search.html
- <http://www.maths.nott.ac.uk/personal/anw/G13GAM/alphabet.html>
- <http://chess.verhelst.org/search.html>
- <http://www.frayn.net/beowulf/index.html>
- <http://hal.inria.fr/docs/00/12/15/16/PDF/RR-6062.pdf>
- Minimax (with or without alpha–beta pruning) algorithm visualization - game tree solving (Java Applet), for balance or off-balance trees
- Demonstration/animation of minimax game search algorithm with alpha–beta pruning (using html5, canvas, javascript, css)
- Java implementation used in a Checkers Game

Chapter 56

Artificial bee colony algorithm

In computer science and operations research, the **artificial bee colony algorithm (ABC)** is an optimization algorithm based on the intelligent foraging behaviour of honey bee swarm, proposed by Karaboga in 2005.^[1]

56.1 Algorithm

In the ABC model, the colony consists of three groups of bees: employed bees, onlookers and scouts. It is assumed that there is only one artificial employed bee for each food source. In other words, the number of employed bees in the colony is equal to the number of food sources around the hive. Employed bees go to their food source and come back to hive and dance on this area. The employed bee whose food source has been abandoned becomes a scout and starts to search for finding a new food source. Onlookers watch the dances of employed bees and choose food sources depending on dances. The main steps of the algorithm are given below:

- Initial food sources are produced for all employed bees
- REPEAT
 - Each employed bee goes to a food source in her memory and determines a neighbour source, then evaluates its nectar amount and dances in the hive
 - Each onlooker watches the dance of employed bees and chooses one of their sources depending on the dances, and then goes to that source. After choosing a neighbour around that, she evaluates its nectar amount.
 - Abandoned food sources are determined and are replaced with the new food sources discovered by scouts.
 - The best food source found so far is registered.
- UNTIL (requirements are met)

In ABC, a population based algorithm, the position of a food source represents a possible solution to the optimization problem and the nectar amount of a food source

corresponds to the quality (fitness) of the associated solution. The number of the employed bees is equal to the number of solutions in the population. At the first step, a randomly distributed initial population (food source positions) is generated. After initialization, the population is subjected to repeat the cycles of the search processes of the employed, onlooker, and scout bees, respectively. An employed bee produces a modification on the source position in her memory and discovers a new food source position. Provided that the nectar amount of the new one is higher than that of the previous source, the bee memorizes the new source position and forgets the old one. Otherwise she keeps the position of the one in her memory. After all employed bees complete the search process, they share the position information of the sources with the onlookers on the dance area. Each onlooker evaluates the nectar information taken from all employed bees and then chooses a food source depending on the nectar amounts of sources. As in the case of the employed bee, she produces a modification on the source position in her memory and checks its nectar amount. Providing that its nectar is higher than that of the previous one, the bee memorizes the new position and forgets the old one. The sources abandoned are determined and new sources are randomly produced to be replaced with the abandoned ones by artificial scouts.

56.2 Application to real-world problems

Since 2005, D. Karaboga and his research group have been studying the ABC algorithm and its applications to real world problems. In 2010, Hadidi et al. employed an Artificial Bee Colony (ABC) Algorithm based approach for structural optimization.^[2] In 2011, Y. Zhang et al. employed the ABC for various tasks, including multi-level thresholding,^[3] MR brain image classification,^[4] and face pose estimation.^[5] Artificial Bee Colony (ABC) algorithm has been used for nanoelectronic based phase-locked loop (PLL) optimization by O. Garitselov, S. P. Mohanty, and E. Kougnos to speedup physical design optimization.^[6]

56.3 See also

- Bees algorithm
- Evolutionary computation
- Intelligent Water Drops
- Swarm intelligence
- Particle Swarm Optimization
- Evolutionary multi-modal optimization

56.4 References

- [1] D. Dervis Karaboga, An Idea Based On Honey Bee Swarm for Numerical Optimization, Technical Report-TR06,Erciyes University, Engineering Faculty, Computer Engineering Department 2005.
- [2] Ali Hadidi, Sina Kazemzadeh Azad, Saeid Kazemzadeh Azad, Structural optimization using artificial bee colony algorithm, 2nd International Conference on Engineering Optimization, 2010, September 6 – 9, Lisbon, Portugal.
- [3] Yudong, Zhang; Lenan, Wu (2011). “Optimal multi-level Thresholding based on Maximum Tsallis Entropy via an Artificial Bee Colony Approach”. *Entropy* **13** (4): 841–859.
- [4] Yudong, Zhang; Lenan, Wu; Shuihua, Wang (2011). “Magnetic Resonance Brain Image Classification by an Improved Artificial Bee Colony Algorithm”. *Progress in Electromagnetics Research - Pier* **116**: 65–79.
- [5] Y. Zhang, L. Wu, Face Pose Estimation by Chaotic Artificial Bee Colony, International Journal of Digital Content Technology and its Applications, vol. 5, no. 2, (2011), pp. 55-63
- [6] O. Garitselov, S. P. Mohanty, and E. Kougnanos, “Accurate Polynomial Metamodeling-Based Ultra-Fast Bee Colony Optimization of a Nano-CMOS PLL”, Special Issue on Power, Parasitics, and Process-Variation (P3) Awareness in Mixed-Signal Design, Journal of Low Power Electronics (JOLPE), Volume 8, Issue 3, June, 2012, pp. 317–328.

56.5 External links

- Artificial Bee Colony Algorithm

Chapter 57

Auction algorithm

The term "**auction algorithm**"^[1] applies to several variations of a combinatorial optimization algorithm which solves **assignment problems**, and network optimization problems with linear and convex/nonlinear cost. An *auction algorithm* has been used in a business setting to determine the best prices on a set of products offered to multiple buyers. It is an iterative procedure, so the name "auction algorithm" is related to a sales **auction**, where multiple bids are compared to determine the best offer, with the final sales going to the highest bidders.

The original form of the auction algorithm is an iterative method to find the optimal prices and an assignment that maximizes the net benefit in a **bipartite graph**, the **maximum weight matching problem** (MWM).^{[2] [3]} This algorithm was first proposed by Dimitri Bertsekas in 1979. Detailed analysis and extensions to more general network optimization problems (ϵ -relaxation in 1986, and network auction in 1992) are provided in his network optimization books **Linear Network Optimization** 1991, and **Network Optimization: Continuous and Discrete Models** 1998. The auction algorithm has excellent computational complexity, as given in these books, and is reputed to be among the fastest for solving single commodity network optimization problems. In addition, the original version of this algorithm is known to possess a distributed nature particularly suitable for distributed systems, since its basic computational primitives (bidding and auctioning) are localized rather than relying on queries of global information.^[1] However, the original version that is intrinsically distributable has a pseudo-polynomial time complexity, which means that the running time depends on the input data pattern. Later versions have improved the time complexity to the state-of-the-art level by using techniques such as ϵ -*scaling* (also discussed in the original 1979 paper)^[4] but at the sacrifice of undermining its distributed characteristics. In order to retain the distributed nature and also attain a polynomial time complexity, recently some researchers from the multi-agent community have been trying to improve the earlier version of the auction algorithm by switching to a different economic model, namely, from the selfish bidders' perspective to a merchant's point of view, where the merchant of a market adjusts the article prices in order to quickly clear the inventory.^[5]

The ideas of the auction algorithm and ϵ -scaling^[1] are also central in preflow-push algorithms for single commodity linear network flow problems. In fact the preflow-push algorithm for max-flow can be derived by applying the original 1979 auction algorithm to the max flow problem after reformulation as an assignment problem; see the **1998 Network Optimization book**, by Bertsekas, Section 7.3.3. Moreover the preflow-push algorithm for the linear minimum cost flow problem is mathematically equivalent to the ϵ -relaxation method, which is obtained by applying the original auction algorithm after the problem is reformulated as an equivalent assignment problem.^[6]

A later variation of the auction algorithm that solves shortest path problems was introduced by Bertsekas in 1991.^[7] It is a simple algorithm for finding shortest paths in a **directed graph**. In the single origin/single destination case, the auction algorithm maintains a single path starting at the origin, which is then extended or contracted by a single node at each iteration. Simultaneously, at most one dual variable will be adjusted at each iteration, in order to either improve or maintain the value of a dual function. In the case of multiple origins, the auction algorithm is well-suited for parallel computation.^[7] The algorithm is closely related to auction algorithms for other network flow problems.^[7] According to computational experiments, the auction algorithm is generally inferior to other state-of-the-art algorithms for the all destinations shortest path problem, but is very fast for problems with few destinations (substantially more than one and substantially less than the total number of nodes); see the article by Bertsekas, Pallottino, and Scutella, **Polynomial Auction Algorithms for Shortest Paths**.

Auction algorithms for shortest hyperpath problems have been defined by De Leone and Pretolani in 1998. This is also a parallel auction algorithm for weighted bipartite matching, described by E. Jason Riedy in 2004.^[8]

57.1 Comparisons

The (sequential) auction algorithms for the shortest path problem have been the subject of experiments which have been reported in technical papers.^[9] Experiments clearly

show that the auction algorithm is inferior to the state-of-the-art shortest-path algorithms for finding the optimal solution.^[9]

Although in the auction algorithm, each iteration never decreases the total benefit (increases or remains the same), with the alternative *Hungarian algorithm* (from Kuhn, 1955; Munkres, 1957), each iteration always increases the total.

The auction algorithm of Bertsekas for finding shortest paths within a directed graph is reputed to perform very well on random graphs and on problems with few destinations.^[7]

57.2 See also

- Hungarian algorithm

57.3 References

- [1] Dimitri P. Bertsekas. “A distributed algorithm for the assignment problem”, original paper, 1979.
- [2] M.G. Resende, P.M. Pardalos. “Handbook of optimization in telecommunications”, 2006
- [3] M. Bayati, D. Shah, M. Sharma. “A Simpler Max-Product Maximum Weight Matching Algorithm and the Auction Algorithm”, 2006, webpage PDF: MIT-bpmwm-PDF.
- [4] Dimitri P. Bertsekas. “The auction algorithm for assignment and other network flow problems: A tutorial”. *Interfaces*, 1990
- [5] L. Liu, D. Shell. “Optimal Market-based Multi-Robot Task Allocation via Strategic Pricing”, 2013. online PDF
- [6] Dimitri P. Bertsekas. “Distributed Relaxation Algorithms for Linear Network Flow Problems,” Proc. of 25th IEEE CDC, Athens, Greece, 1986, pp. 2101-2106, online from IEEEXplore
- [7] Dimitri P. Bertsekas. “An auction algorithm for shortest paths”, *SIAM Journal on Optimization*, 1:425—447, 1991,PSU-bertsekas91auction
- [8] “The Parallel Auction Algorithm for Weighted Bipartite Matching”, E. Jason Riedy, UC Berkeley, February 2004, Berkeley-para4-PDF.
- [9] Larsen, Jesper; Pedersen, Ib (1999). “Experiments with the auction algorithm for the shortest path problem”. *Nordic J. of Computing* **6** (4): 403–42. ISSN 1236-6064., see also A note on the practical performance of the auction algorithm for the shortest path (1997) by the first author.

57.4 External links

- Dimitri P. Bertsekas. “Linear Network Optimization”, MIT Press, 1991, on-line.

- Dimitri P. Bertsekas. “Network Optimization: Continuous and Discrete Models”, Athena Scientific, 1998.
- Dimitri P. Bertsekas. “An auction algorithm for shortest paths”, *SIAM Journal on Optimization*, 1:425—447, 1991, webpage: PSU-bertsekas91auction.
- D.P. Bertsekas, S. Pallottino, M. G. Scutella. “Polynomial Auction Algorithms for Shortest Paths.”, Computational Optimization and Applications, Vol. 4, 1995, pp. 99-125.

Chapter 58

Automatic label placement

Automatic label placement, sometimes called **text placement** or **name placement**, comprises the computer methods of placing labels automatically on a map or chart. This is related to the **typographic design** of such labels.

The typical features depicted on a geographic map are line features (e.g. roads), area features (countries, parcels, forests, lakes, etc.), and point features (villages, cities, etc.). In addition to depicting the map's features in a geographically accurate manner, it is of critical importance to place the names that identify these features, in a way that the reader knows instantly which name describes which feature.

Automatic text placement is one of the most difficult, complex, and time-consuming problems in mapmaking and **GIS** (Geographic Information System). Other kinds of computer-generated graphics – like charts, graphs etc. – require good placement of labels as well, not to mention engineering drawings, and professional programs which produce these drawings and charts, like **spreadsheets** (e.g. Microsoft Excel) or computational software programs (e.g. Mathematica).

Naively placed labels overlap excessively, resulting in a map that is difficult or even impossible to read. Therefore, a GIS must allow a few possible placements of each label, and often also an option of resizing, rotating, or even removing (suppressing) the label. Then, it selects a set of placements that results in the least overlap, and has other desirable properties. For all but the most trivial setups, the problem is **NP-hard**.

58.1 Rule-based algorithms

Rule-based algorithms try to emulate an experienced human cartographer. Over centuries, cartographers have developed the art of mapmaking and label placement. For example, an experienced cartographer repeats road names several times for long roads, instead of placing them once, or in the case of Ocean City depicted by a point very close to the shore, the cartographer would place the label “Ocean City” over the water to emphasize that it is a coastal town.^[1]

Cartographers work based on accepted conventions and rules and they place labels in order of importance. For example, New York City, Vienna, Berlin, Paris, or Tokyo must show up on country maps because they are high-priority labels. Once those are placed, the cartographer places the next most important class of labels, for example major roads, rivers, and other large cities. In every step they ensure that (1) the text is placed in a way that the reader easily associates it with the feature, and (2) the label does not overlap with those already placed on the map.

58.2 Local optimization algorithms

The simplest **greedy algorithm** places consecutive labels on the map in positions that result in minimal overlap of labels. Its results are not perfect even for very simple problems, but it is extremely fast.

Slightly more complex algorithms rely on local optimization to reach a local optimum of a placement evaluation function – in each iteration placement of a single label is moved to another position, and if it improves the result, the move is preserved. It performs reasonably well for maps that are not too densely labelled. Slightly more complex variations try moving 2 or more labels at the same time. The algorithm ends after reaching some local optimum.

A simple algorithm – **simulated annealing** – yields good results with relatively good performance. It works like local optimization, but it may keep a change even if it worsens the result. The chance of keeping such a change is $\exp \frac{-\Delta E}{T}$, where ΔE is the change in the evaluation function, and T is the *temperature*. The temperature is gradually lowered according to the *annealing schedule*. When the temperature is high, simulated annealing performs almost random changes to the label placement, being able to escape a local optimum. Later, when hopefully a very good local optimum has been found, it behaves in a manner similar to local optimization. The main challenges in developing a simulated annealing solution are choosing a good evaluation function and a good anneal-

ing schedule. Generally too fast cooling will degrade the solution, and too slow cooling will degrade the performance, but the schedule is usually quite a complex algorithm, with more than just one parameter.

Another class of direct search algorithms are the various evolutionary algorithms, e.g. genetic algorithms.

58.3 Divide-and-conquer algorithms

One simple optimization that is important on real maps is dividing a set of labels into smaller sets that can be solved independently. Two labels are *rivals* if they can overlap in one of the possible placements. Transitive closure of this relation divides the set of labels into possibly much smaller sets. On uniformly and densely labelled maps, usually the single set will contain the majority of labels, and on maps for which the labelling is not uniform it may bring very big performance benefits. For example when labelling a map of the world, America is labelled independently from Eurasia etc.

58.4 2-satisfiability algorithms

If a map labeling problem can be reduced to a situation in which each remaining label has only two potential positions in which it can be placed, then it may be solved efficiently by using an instance of 2-satisfiability to find a placement avoiding any conflicting pairs of placements; several exact and approximate label placement algorithms for more complex types of problems are based on this principle.^[2]

58.5 Other algorithms

Automatic label placement algorithms can use any of the algorithms for finding the Maximum disjoint set from the set of potential labels.

Other algorithms are also used, like various graph solutions, integer programming etc.

58.6 Notes

[1] Slocum, Terry (2010). *Thematic Cartography and Geovisualization*. Upper Saddle River, NJ: Pearson. p. 576. ISBN 0-13-801006-4.

[2] Doddi, Srinivas; Marathe, Madhav V.; Mirzaian, Andy; Moret, Bernard M. E.; Zhu, Binhai (1997), "Map labeling and its generalizations", *Proc. 8th ACM-SIAM Symp. Discrete Algorithms (SODA)*, pp. 148–157; Formann, M.;

Wagner, F. (1991), "A packing problem with applications to lettering of maps", *Proc. 7th ACM Symp. Computational Geometry*, pp. 281–288; Poon, Chung Keung; Zhu, Binhai; Chin, Francis (1998), "A polynomial time solution for labeling a rectilinear map", *Information Processing Letters* **65** (4): 201–207, doi:10.1016/S0020-0190(98)00002-7; Wagner, Frank; Wolff, Alexander (1997), "A practical map labeling algorithm", *Computational Geometry: Theory and Applications* **7** (5–6): 387–404, doi:10.1016/S0925-7721(96)00007-7.

58.7 References

- Imhof, E., "Die Anordnung der Namen in der Karte," *Annuaire International de Cartographie II*, Orell-Füssli Verlag, Zürich, 93–129, 1962.
- Freeman, H., Map data processing and the annotation problem, Proc. 3rd Scandinavian Conf. on Image Analysis, Chartwell-Bratt Ltd. Copenhagen, 1983.
- Ahn, J. and Freeman, H., "A program for automatic name placement," Proc. AUTO-CARTO 6, Ottawa, 1983. 444–455.
- Freeman, H., "Computer Name Placement," ch. 29, in *Geographical Information Systems*, 1, D.J. Maguire, M.F. Goodchild, and D.W. Rhind, John Wiley, New York, 1991, 449–460.
- Podolskaya N. N. Automatic Label De-Confliction Algorithms for Interactive Graphics Applications. *Information technologies* (ISSN 1684-6400), 9, 2007, p. 45–50. In Russian: Подольская Н.Н. Алгоритмы автоматического отброса формулаторов для интерактивных графических приложений. *Информационные технологии*, 9, 2007, с. 45–50.

58.8 External links

- Alexander Wolff's Map Labeling Site
- The Map-Labeling Bibliography
- Label placement
- An Empirical Study of Algorithms for Point-Feature Label Placement

Chapter 59

Bees algorithm

In computer science and operations research, the Bees Algorithm is a population-based search algorithm which was developed in 2005.^[1] It mimics the food foraging behaviour of honey bee colonies. In its basic version the algorithm performs a kind of neighbourhood search combined with global search, and can be used for both combinatorial optimization and continuous optimization. The only condition for the application of the Bees Algorithm is that some measure of topological distance between the solutions is defined. The effectiveness and specific abilities of the Bees Algorithm have been proven in a number of studies. ^{[2][3]}



The Bees Algorithm is inspired by the foraging behaviour of honey bees.

59.1 Honey bees foraging strategy in nature

A colony of honey bees can extend itself over long distances (over 14 km)^[4] and in multiple directions simultaneously to harvest nectar or pollen from multiple food sources (flower patches). A small fraction of the colony constantly searches the environment looking for new flower patches. These scout bees move randomly in the area surrounding the hive, evaluating the profitability (net energy yield) of the food sources encountered.^[4]

When they return to the hive, the scouts deposit the food harvested. Those individuals that found a highly profitable food source go to an area in the hive called the “dance floor”, and perform a ritual known as the waggle dance.^[5] Through the waggle dance a scout bee communicates the location of its discovery to idle onlookers, which join in in the exploitation of the flower patch. Since the length of the dance is proportional to the scout’s rating of the food source, more foragers get recruited to harvest the best rated flower patches. After dancing, the scout return to the food source it discovered to collect more food. As long as they are evaluated as profitable, rich food sources will be advertised by the scouts when they return to the hive. Recruited foragers may waggle dance as well, increasing the recruitment for highly rewarding flower patches. Thanks to this autocatalytic process, the bee colony is able to quickly switch the focus of the foraging effort on the most profitable flower patches.^[4]

59.2 The Bees Algorithm

The Bees Algorithm^{[2][6]} mimics the foraging strategy of honey bees to look for the best solution to an optimisation problem. Each candidate solution is thought of as a food source (flower), and a population (colony) of n agents (bees) is used to search the solution space. Each time an artificial bee visits a flower (lands on a solution), it evaluates its profitability (fitness).

The Bees Algorithm consists of an initialisation procedure and a main search cycle which is iterated for a given number T of times, or until a solution of acceptable fitness is found. Each search cycle is composed of five procedures: recruitment, local search, neighbourhood shrinking, site abandonment, and global search.

The pseudocode for the standard Bees Algorithm^[2]

```
1 for i=1,...,ns i scout[i]=Initialise_scout()
ii flower_patch[i]=Initialise_flower_patch(scout[i])
2 do until stopping_condition=TRUE
i Recruitment() ii for i =1,...,nb 1
flower_patch[i]=Local_search(flower_patch[i]) 2
flower_patch[i]=Site_abandonment(flower_patch[i]) 3
flower_patch[i]=Neighbourhood_shrinking(flower_patch[i])
```

```

iii      for      i      =      nb,...,ns      1
flower_patch[i]=Global_search(flower_patch[i])

```

In the initialisation routine ns scout bees are randomly placed in the search space, and evaluate the fitness of the solutions where they land. For each solution, a neighbourhood (called flower patch) is delimited.

In the recruitment procedure, the scouts that visited the $nb \leq ns$ fittest solutions (best sites) perform the waggle dance. That is, they recruit foragers to search further the neighbourhoods of the most promising solutions. The scouts that located the very best $ne \leq nb$ solutions (elite sites) recruit nre foragers each, whilst the remaining $nb-ne$ scouts recruit $nrb \leq nre$ foragers each. Thus, the number of foragers recruited depends on the profitability of the food source.

In the local search procedure, the recruited foragers are randomly scattered within the flower patches enclosing the solutions visited by the scouts (local exploitation). If any of the foragers in a flower patch lands on a solution of higher fitness than the solution visited by the scout, that forager becomes the new scout. If no forager finds a solution of higher fitness, the size of the flower patch is shrunk (neighbourhood shrinking procedure). Usually, flower patches are initially defined over a large area, and their size is gradually shrunk by the neighbourhood shrinking procedure. As a result, the scope of the local exploration is progressively focused on the area immediately close to the local fitness best. If no improvement in fitness is recorded in a given flower patch for a pre-set number of search cycles, the local maximum of fitness is considered found, the patch is abandoned (site abandonment), and a new scout is randomly generated.

As in biological bee colonies,^[4] a small number of scouts keeps exploring the solution space looking for new regions of high fitness (global search). The global search procedure re-initialises the last $ns-nb$ flower patches with randomly generated solutions.

At the end of one search cycle, the scout population is again composed of ns scouts: nr scouts produced by the local search procedure (some of which may have been re-initialised by the site abandonment procedure), and $ns-nb$ scouts generated by the global search procedure. The total artificial bee colony size is $n=ne+nre+(nb-ne)\bullet nrb+ns$ (elite sites foragers + remaining best sites foragers + scouts) bees.

59.3 Applications

The Bees Algorithm has found many applications in engineering, such as:

Optimisation of classifiers / clustering systems [7][8][9]

Manufacturing^{[10][11][12][13][14][15]}

Control^{[16][17][18][19]}

Bioengineering^{[20][21]}

1 Other optimisation problems^{[22][23][24][25][26]}

Multi-objective optimisation^{[27][28][29]}

59.4 See also

- Mathematical optimization
- Metaheuristic
- Evolutionary computation
- Swarm intelligence
- Particle swarm optimization
- Ant colony optimization algorithms
- Artificial bee colony algorithm
- Intelligent Water Drops
- Lévy flight foraging hypothesis
- Manufacturing Engineering Centre

59.5 References

- [1] Pham DT, Ghanbarzadeh A, Koc E, Otri S, Rahim S and Zaidi M. The Bees Algorithm. Technical Note, Manufacturing Engineering Centre, Cardiff University, UK, 2005.
- [2] Pham, D.T., Castellani, M. (2009), The Bees Algorithm – Modelling Foraging Behaviour to Solve Continuous Optimisation Problems. Proc. ImechE, Part C, 223(12), 2919-2938.
- [3] Pham, D.T. and Castellani, M. (2013), Benchmarking and Comparison of Nature-Inspired Population-Based Continuous Optimisation Algorithms, Soft Computing, 1-33.
- [4] Tereshko V., Loengarov A., (2005) Collective Decision-Making in Honey Bee Foraging Dynamics. Journal of Computing and Information Systems, 9(3), 1-7.
- [5] Von Frisch, K. (1967) The Dance Language and Orientation of Bees. Harvard University Press, Cambridge, MA.
- [6] Pham D.T., Ghanbarzadeh A., Koc E., Otri S., Rahim S., Zaidi M., The Bees Algorithm, A Novel Tool for Complex Optimisation Problems, Proc 2nd Int Virtual Conf on Intelligent Production Machines and Systems (IPROMS 2006), Oxford: Elsevier, pp. 454-459, 2006.
- [7] Pham D.T., Zaidi M., Mahmuddin M., Ghanbarzadeh A., Koc E., Otri S. (2007), Using the bees algorithm to optimise a support vector machine for wood defect classification, IPROMS 2007 Innovative Production Machines and Systems Virtual Conference.
- [8] Pham D.T., Darwish A.H. (2010), Using the bees algorithm with Kalman filtering to train an artificial neural network for pattern classification, Journal of Systems and Control Engineering 224(7), 885-892.

- [9] Pham D.T., Suarez-Alvarez M.M., Prostov Y.I. (2011), Random search with k-prototypes algorithm for clustering mixed datasets, *Proceedings Royal Society*, 467, 2387-2403.
- [10] Pham D.T., Castellani M., Ghanbarzadeh A. (2007), Preliminary design using the Bees Algorithm, *Proceedings Eighth LAMDAMAP International Conference on Laser Metrology, CMM and Machine Tool Performance*. Cardiff - UK, 420-429.
- [11] Pham, D.T., Otri S., Darwish A.H. (2007), Application of the Bees Algorithm to PCB assembly optimisation, *Proceedings 3rd International Virtual Conference on Intelligent Production Machines and Systems (IPROMS 2007)*, Whittles, Dunbeath, Scotland, 511-516.
- [12] Pham D.T., Koç E., Lee J.Y., Phruksanant J. (2007), Using the Bees Algorithm to Schedule Jobs for a Machine, *Proceedings 8th international Conference on Laser Metrology, CMM and Machine Tool Performance (LAMDAMAP)*. Cardiff, UK, Euspen, 430-439.
- [13] Baykasoglu A., Özbakir L., Tapkan P. (2009), The bees algorithm for workload balancing in examination job assignment, *European Journal Industrial Engineering* 3(4) 424-435.
- [14] Özbakir L., Tapkan P. (2011), Bee colony intelligence in zone constrained two-sided assembly line balancing problem, *Expert Systems with Applications* 38, 11947-11957.
- [15] Xu W., Zhou Z., Pham D.T., Liu Q., Ji C., Meng W. (2012), Quality of service in manufacturing networks: a service framework and its implementation, *International Journal Advanced Manufacturing Technology*, 63(9-12), 1227-1237.
- [16] Pham D.T., Darwish A.H., Eldukhri E.E. (2009), Optimisation of a fuzzy logic controller using the Bees Algorithm, *International Journal of Computer Aided Engineering and Technology*, 1, 250-264.
- [17] Alfi A., Khosravi A., Razavi S.E. (2011), Bee Algorithm-Based Nolinear Optimal Control Applied To A Continuous Stirred-Tank Chemical Reactor, *Global Journal of Pure & Applied Science and Technology - GJPAST* 1(2), 73-79.
- [18] Fahmy A.A., Kalyoncu M., Castellani M. (2012), Automatic Design of Control Systems for Robot Manipulators Using the Bees Algorithm, *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 226(4), 497-508.
- [19] Castellani M., Pham Q.T., Pham D.T. (2012), Dynamic Optimisation by a Modified Bees Algorithm, *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 226(7), 956-971.
- [20] Bahamish H.A.A., Abdullah R., Salam R.A. (2008), Protein Conformational Search Using Bees Algorithm, *Second Asia International Conference on Modeling & Simulation (AICMS 08)*, Kuala Lumpur, Malaysia, IEEE Press, 911-916.
- [21] Ruz G.A., Goles E. (2013), Learning gene regulatory networks using the bees algorithm, *Neural Computing and Applications*, 22(1), 63-70.
- [22] Guney K., Onay M. (2010), Bees algorithm for interference suppression of linear antenna arrays by controlling the phase-only and both the amplitude and phase, *Expert Systems with Applications* 37, 3129-3135.
- [23] Xu S., Yu F., Luo Z., Ji Z., Pham D.T., Qiu R. (2011), Adaptive Bees Algorithm - Bioinspiration from Honeybee Foraging to Optimize Fuel Economy of a Semi-Track Air-Cushion Vehicle, *The Computer Journal* 54(9), 1416-1426.
- [24] Pham D.T., Koç E. (2011), Design of a two-dimensional recursive filter using the bees algorithm, *International Journal Automation and Computing* 7(3) 399-402.
- [25] Kavousi A., Vahidi B., Salehi R., Bakhshizadeh M.K., Farokhnia N., Fathi S.H. (2012), Application of the Bee Algorithm for Selective Harmonic Elimination Strategy in Multilevel Inverters, *IEEE Transactions on Power Electronics* 27(4) 1689-1696.
- [26] Jevtic A., Gutierrez-Martin A., Andina D.A., Jamshidi M. (2012), Distributed Bees Algorithm for Task Allocation in Swarm of Robots, *IEEE Systems Journal* 6(2) 296-304.
- [27] Lee J.Y., Darwish A.H. (2008), Multi-objective Environmental/Economic Dispatch Using the Bees Algorithm with Weighted Sum, *Proceedings of the EU-Korea Conference on Science and Technology (EKC2008)*, Ed. S.D. Yoo, Heidelberg, D, Springer Berlin Heidelberg, 267-274.
- [28] Sayadi F., Ismail M., Misran N., Jumari K. (2009), Multi-Objective Optimization Using the Bees Algorithm in Time-Varying Channel for MIMO MC-CDMA Systems, *European Journal of Scientific Research* 33(3), 411-428.
- [29] Mansouri Poor M., Shisheh Saz M. (2012), Multi-Objective Optimization of Laminates with Straight Free Edges and Curved Free Edges by Using Bees Algorithm, *American Journal of Advanced Scientific Research* 1(4), 130-136.

59.6 External links

- The Bees Algorithm website
- Boffins put dancing bees to work – BBC News
- An optimisation algorithm based on honey bees' food foraging behaviour

Chapter 60

Benson's algorithm

Not to be confused with Benson's algorithm (Go), a method to find the unconditionally alive stones in the game Go.

Benson's algorithm, named after Harold Benson, is a method for solving linear multi-objective optimization problems. This works by finding the “efficient extreme points in the outcome set”.^[1] The primary concept in Benson's algorithm is to evaluate the upper image of the vector optimization problem by cutting planes.^[2]

60.1 Idea of algorithm

Consider a vector linear program

$$\min_C P x \text{ to subject } Ax \geq b$$

for $P \in \mathbb{R}^{q \times n}$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and a polyhedral convex ordering cone C having nonempty interior and containing no lines. The feasible set is $S = \{x \in \mathbb{R}^n : Ax \geq b\}$. In particular, Benson's algorithm finds the **extreme points** of the set $P[S] + C$, which is called upper image.^[2]

In case of $C = \mathbb{R}_+^q := \{y \in \mathbb{R}^q : y_1 \geq 0, \dots, y_q \geq 0\}$, one obtains the special case of a multi-objective linear program (**multiobjective optimization**).

60.2 Implementations

60.2.1 Bensolve - a free VLP solver (C programming language)

- www.bensolve.org

60.3 References

[1] Harold P. Benson (1998). “An Outer Approximation Algorithm for Generating All Efficient Extreme Points in

the Outcome Set of a Multiple Objective Linear Programming Problem”. *Journal of Global Optimization* **13** (1): 1–24. doi:10.1023/A:1008215702611. Retrieved September 21, 2013.

[2] Andreas Löhne (2011). *Vector Optimization with Infimum and Supremum*. Springer. pp. 162–169. ISBN 9783642183508.

Chapter 61

Berndt–Hall–Hall–Hausman algorithm

The **Berndt–Hall–Hall–Hausman (BHHH) algorithm** is a numerical optimization algorithm similar to the Gauss–Newton algorithm. It is named after the four originators: Ernst R. Berndt, B. Hall, Robert Hall, and Jerry Hausman.

61.1 Usage

If a nonlinear model is fitted to the data one often needs to estimate coefficients through optimization. A number of optimisation algorithms have the following general structure. Suppose that the function to be optimized is $Q(\beta)$. Then the algorithms are iterative, defining a sequence of approximations, β_k given by

$$\beta_{k+1} = \beta_k - \lambda_k A_k \frac{\partial Q}{\partial \beta}(\beta_k),$$

where β_k is the parameter estimate at step k , and λ_k is a parameter (called step size) which partly determines the particular algorithm. For the BHHH algorithm λ_k is determined by calculations within a given iterative step, involving a line-search until a point β_{k+1} is found satisfying certain criteria. In addition, for the BHHH algorithm, Q has the form

$$Q = \sum_{i=1}^N Q_i$$

and A is calculated using

$$A_k = \left[\sum_{i=1}^N \frac{\partial \ln Q_i}{\partial \beta}(\beta_k) \frac{\partial \ln Q_i}{\partial \beta}(\beta_k)' \right]^{-1}.$$

In other cases, e.g. **Newton–Raphson**, A_k can have other forms. The BHHH algorithm has the advantage that, if certain conditions apply, convergence of the iterative procedure is guaranteed.

61.2 See also

- Davidon–Fletcher–Powell (DFP) algorithm
- Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm

61.3 Further reading

- Berndt, E.; Hall, B.; Hall, R.; Hausman, J. (1974). “Estimation and Inference in Nonlinear Structural Models”. *Annals of Economic and Social Measurement* 3: 653–665.
- Gill, P.; Murray, W.; Wright, M. (1981). *Practical Optimization*. London: Harcourt Brace.
- Harvey, A. C. (1990). *The Econometric Analysis of Time Series* (Second ed.). Cambridge: MIT Press. pp. 137–138. ISBN 0-262-08189-X.
- Luenberger, D. (1972). *Introduction to Linear and Nonlinear Programming*. Reading, Massachusetts: Addison Wesley.
- Sokolov, S. N.; Silin, I. N. (1962). “Determination of the coordinates of the minima of functionals by the linearization method”. *Joint Institute for Nuclear Research preprint D-810, Dubna*.

Chapter 62

Big M method

In operations research, the **Big M method** is a method of solving linear programming problems using the simplex algorithm. The Big M method extends the power of the simplex algorithm to problems that contain “greater-than” constraints. It does so by associating the constraints with large negative constants which would not be part of any optimal solution, if it exists.

62.1 Algorithm

The simplex algorithm is the original and still one of the most widely used methods for solving linear maximization problems. However, to apply it, the origin (all variables equal to 0) must be a feasible point. This condition is satisfied only when all the constraints (except non-negativity) are less-than constraints with a positive constant on the right-hand side. The Big M method introduces surplus and artificial variables to convert all inequalities into that form. The “Big M” refers to a large number associated with the artificial variables, represented by the letter M.

The steps in the algorithm are as follows:

1. Multiply the inequality constraints to ensure that the right hand side is positive.
2. If the problem is of minimization, transform to maximization by multiplying the objective by -1
3. For any greater-than constraints, introduce surplus and artificial variables (as shown below)
4. Choose a large positive M and introduce a term in the objective of the form $-M$ multiplying the artificial variables
5. For less-than or equal constraints, introduce slack variables so that all constraints are equalities
6. Solve the problem using the usual simplex method.

For example $x + y \leq 100$ becomes $x + y + s_1 = 100$, whilst $x + y \geq 100$ becomes $x + y - a_1 = 100$. The artificial variables must be shown to be 0. The function to be maximised is rewritten to include the sum of all the artificial

variables. Then row reductions are applied to gain a final solution.

The value of M must be chosen sufficiently large so that the artificial variable would not be part of any feasible solution.

For a sufficiently large M, the optimal solution contains any artificial variables in the basis (i.e. positive values) if and only if the problem is not feasible.

62.2 Other usage

The Big M method sometimes refers to any formulation of a linear optimization problem in which violations of a constraint are associated with a large positive penalty constant, M.

Another usage refers to using a large positive constant, M, to ensure that the constraint is not tight. For example, suppose x and y are ≥ 0 . Then for a sufficiently large M and z binary variable (0 or 1), the following constraint ensures that when $z=1$, $x=y$: $x - y \leq M(1 - z)$

62.3 See also

- Two phase method (linear programming) another approach for solving problems with \geq constraints
- Karush–Kuhn–Tucker conditions, which apply to Non-Linear Optimization problems with inequality constraints.

62.4 References and external links

Bibliography

- Griva, Igor; Nash, Stephan G.; Sofer, Ariela. *Linear and Nonlinear Optimization* (2nd ed.). Society for Industrial Mathematics. ISBN 978-0-89871-661-0.

Discussion

- Simplex – Big M Method, Lynn Killen, Dublin City University.
- The Big M Method, businessmanagement-courses.org
- The Big M Method, Mark Hutchinson

Chapter 63

Bin packing problem

In the **bin packing problem**, objects of different volumes must be packed into a finite number of bins or containers each of volume V in a way that minimizes the number of bins used. In computational complexity theory, it is a combinatorial NP-hard problem.

There are many variations of this problem, such as 2D packing, linear packing, packing by weight, packing by cost, and so on. They have many applications, such as filling up containers, loading trucks with weight capacity constraints, creating file backups in media and technology mapping in Field-programmable gate array semiconductor chip design.

The bin packing problem can also be seen as a special case of the **cutting stock problem**. When the number of bins is restricted to 1 and each item is characterised by both a volume and a value, the problem of maximising the value of items that can fit in the bin is known as the knapsack problem.

Despite the fact that the bin packing problem has an NP-hard computational complexity, optimal solutions to very large instances of the problem can be produced with sophisticated algorithms. In addition, many heuristics have been developed: for example, the **first fit algorithm** provides a fast but often non-optimal solution, involving placing each item into the first bin in which it will fit. It requires $\Theta(n \log n)$ time, where n is the number of elements to be packed. The algorithm can be made much more effective by first sorting the list of elements into decreasing order (sometimes known as the first-fit decreasing algorithm), although this still does not guarantee an optimal solution, and for longer lists may increase the running time of the algorithm. It is known, however, that there always exists at least one ordering of items that allows first-fit to produce an optimal solution.^[1]

A variant of bin packing that occurs in practice is when items can share space when packed into a bin. Specifically, a set of items could occupy less space when packed together than the sum of their individual sizes. This variant is known as VM packing^[2] since when virtual machines (VMs) are packed in a server, their total memory requirement could decrease due to pages shared by the VMs that need only be stored once. If items can share space in arbitrary ways, the bin packing problem is hard

to even approximate. However, if the space sharing fits into a hierarchy, as is the case with memory sharing in virtual machines, the bin packing problem can be efficiently approximated.

63.1 Formal statement

Given a bin S of size V and a list of n items with sizes a_1, \dots, a_n to pack, find an integer number of bins B and a B -partition $S_1 \cup \dots \cup S_B$ of the set $\{1, \dots, n\}$ such that $\sum_{i \in S_k} a_i \leq V$ for all $k = 1, \dots, B$. A solution is *optimal* if it has minimal B . The B -value for an optimal solution is denoted **OPT** below. A possible Integer Linear Programming formulation of the problem is:

where $y_i = 1$ if bin i is used and $x_{ij} = 1$ if item j is put into bin i .^[3]

63.2 First-fit algorithm

This is a very straightforward greedy approximation algorithm. The algorithm processes the items in arbitrary order. For each item, it attempts to place the item in the first bin that can accommodate the item. If no bin is found, it opens a new bin and puts the item within the new bin.

It is rather simple to show this algorithm achieves an approximation factor of 2, that is, the number of bins used by this algorithm is no more than twice the optimal number of bins. In other words, it is impossible for 2 bins to be at most half full because such a possibility implies that at some point, exactly one bin was at most half full and a new one was opened to accommodate an item of size at most $V/2$. But since the first one has at least a space of $V/2$, the algorithm will not open a new bin for any item whose size is at most $V/2$. Only after the bin fills with more than $V/2$ or if an item with a size larger than $V/2$ arrives, the algorithm may open a new bin.

Thus if we have B bins, at least $B - 1$ bins are more than half full. Therefore $\sum_{i=1}^n a_i > \frac{B-1}{2}V$. Because $\frac{\sum_{i=1}^n a_i}{V}$ is a lower bound of the optimum value OPT , we get that $B - 1 < 2OPT$ and therefore $B \leq 2OPT$.^[4] See the

analysis below for better approximation results.

63.3 Analysis of approximate algorithms

The *best fit decreasing* and *first fit decreasing* strategies are among the simplest heuristic algorithms for solving the bin packing problem. They have been shown to use no more than $11/9 \text{ OPT} + 1$ bins (where **OPT** is the number of bins given by the optimal solution).^[5] The simpler of these, the *First Fit Decreasing* (FFD) strategy, operates by first sorting the items to be inserted in decreasing order by their sizes, and then inserting each item into the first bin in the list with sufficient remaining space. Sometimes, however, one does not have the option to sort the input, for example, when faced with an [online](#) bin packing problem. In 2007, it was proven that the bound $11/9 \text{ OPT} + 6/9$ for FFD is [tight](#).^[6] MFFD^[7] (a variant of FFD) uses no more than $71/60 \text{ OPT} + 1$ bins^[8] (i.e. bounded by about 1.18 OPT , compared to about 1.22 OPT for FFD). In 2013, Sgall and Dósa gave a tight upper bound for the first-fit (FF) strategy, showing that it never needs more than $17/10 \text{ OPT}$ bins for any input.

It is NP-hard to distinguish whether **OPT** is 2 or 3, thus for all $\epsilon > 0$, bin packing is hard to approximate within $3/2 - \epsilon$. (If such an approximation exists, one could determine whether n non-negative integers can be partitioned into two sets with the same sum in polynomial time. However, this problem is known to be NP-hard.) Consequently, the bin packing problem does not have a polynomial-time approximation scheme (PTAS) unless P = NP. On the other hand, for any $0 < \epsilon \leq 1$, it is possible to find a solution using at most $(1 + \epsilon)\text{OPT} + 1$ bins in polynomial time. This approximation type is known as asymptotic PTAS.^{[9][10]}

63.4 Exact algorithm

Martello and Toth^[11] developed an exact algorithm for the 1-D bin-packing problem, called MTP. A faster alternative is the Bin Completion algorithm proposed by Korf in 2002.^[12]

63.5 Software

63.6 See also

- If the number of bins is to be fixed or constrained, and the size of the bins is to be minimised, that is a different problem which is equivalent to the Multiprocessor scheduling problem
- Guillotine problem

- Packing problem
- Partition problem
- Subset sum problem

63.7 Notes

- [1] Lewis, Rhyd (2009). “A General-Purpose Hill-Climbing Method for Order Independent Minimum Grouping Problems: A Case Study in Graph Colouring and Bin Packing”. *Computers and Operations Research* 36 (7): 2295–2310. doi:10.1016/j.cor.2008.09.004.
- [2] Sindelar, Sitaraman & Shenoy 2011, pp. 367–378
- [3] Silvano Martello and Paolo Toth (1990). *Knapsack problems*. Chichester, UK: John Wiley and Sons. p. 221. ISBN 0471924202.
- [4] Vazirani 2003, p. 74.
- [5] Yue 1991, pp. 321–331.
- [6] Dósa 2007, pp. 1–11.
- [7] Garey & Johnson 1985, pp. 65–106.
- [8] Yue & Zhang 1995, pp. 318–330.
- [9] Vazirani 2003, pp. 74–76.
- [10] de la Vega & Lueker 1981, pp. 349–355
- [11] Martello & Toth 1990, pp. 237–240.
- [12] Korf 2002

63.8 References

1. Korf, Richard E. (2002), *A new algorithm for optimal bin packing*.
1. Vazirani, Vijay V. (2003), *Approximation Algorithms*, Berlin: Springer, ISBN 3-540-65367-8
2. Yue, Minyi (October 1991), “A simple proof of the inequality FFD (L) $\leq 11/9 \text{ OPT} (L) + 1$, $\forall L$ for the FFD bin-packing algorithm”, *Acta Mathematicae Applicatae Sinica* 7 (4): 321–331, doi:10.1007/BF02009683, ISSN 0168-9673 |chapter= ignored (help)
3. Dósa, György (2007), “The Tight Bound of First Fit Decreasing Bin-Packings Algorithm Is $\text{FFD}(I) \leq (11/9)\text{OPT}(I) + 6/9$ ”, in Chen, Bo; Patterson, Mike; Zhang, Guochuan, *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, 4614/2007, Springer Berlin / Heidelberg, pp. 1–11, doi:10.1007/978-3-540-74450-4, ISBN 978-3-540-74449-8, ISSN 0302-9743

4. Xia, Binzhou; Tan, Zhiyi (2010), “Tighter bounds of the First Fit algorithm for the bin-packing problem”, *Discrete Applied Mathematics* **158** (15): 1668–1675, doi:10.1016/j.dam.2010.05.026, ISSN 0166-218X lchapter= ignored (help)
 5. Garey, Michael R.; Johnson, David S. (1985), “A 71/60 theorem for bin packing*1”, *Journal of Complexity* **1**: 65–106, doi:10.1016/0885-064X(85)90022-6 lchapter= ignored (help)
 6. Yue, Minyi; Zhang, Lei (July 1995), “A simple proof of the inequality $MFFD(L) \leq 71/60 OPT(L) + 1, L$ for the MFFD bin-packing algorithm”, *Acta Mathematicae Applicatae Sinica* **11** (3): 318–330, doi:10.1007/BF02011198, ISSN 0168-9673 lchapter= ignored (help)
 7. Fernandez de la Vega, W.; Lueker, G. S. (December 1981), “Bin packing can be solved within $1 + \epsilon$ in linear time”, *Combinatorica* (Springer Berlin / Heidelberg) **1** (4): 349–355, doi:10.1007/BF02579456, ISSN 0209-9683 lchapter= ignored (help)
 8. Lewis, R. (2009), “A General-Purpose Hill-Climbing Method for Order Independent Minimum Grouping Problems: A Case Study in Graph Colouring and Bin Packing”, *Computers and Operations Research* **36** (7): 2295–2310, doi:10.1016/j.cor.2008.09.004
 9. Silvano Martello and Paolo Toth (1990), Knapsack Problems Algorithms and Computer Implementations.
 10. Michael R. Garey and David S. Johnson (1979), Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman. ISBN 0-7167-1045-5. A4.1: SR1, p. 226.
 11. David S. Johnson, Alan J. Demers, Jeffrey D. Ullman, M. R. Garey, Ronald L. Graham. Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. SICOMP, Volume 3, Issue 4. 1974.
 12. Lodi A., Martello S., Monaci, M., Vigo, D. (2010) Two-Dimensional Bin Packing Problems. In V.Th. Paschos (Ed.), “Paradigms of Combinatorial Optimization”, Wiley/ISTE, p. 107-129
 13. Dósa G., Sgall J. (2013) First Fit bin packing: A tight analysis. To appear in STACS 2013.
1. Sindelar, Michael; Sitaraman, Ramesh; Shenoy, Prashant (2011), “Sharing-Aware Algorithms for Virtual Machine Colocation”, *Proceedings of 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), San Jose, CA, June 2011:* 367–378

63.9 External links

- API for 3D bin packing
- PHP Class to pack files without exceeding a given size limit
- An implementation of several bin packing heuristics in Haskell, including FFD and MFID.
- Cutting And Packing Algorithms Research Framework, including a number of bin packing algorithms and test data.
- A simple on-line bin-packing algorithm
- Optimizing Three-Dimensional Bin Packing
- Fpart : open-source command-line tool to pack files (C, BSD-licensed)
- Bin Packing and Cutting Stock Solver Algorithm

Chapter 64

Bland's rule

In mathematical optimization, **Bland's rule** (also known as **Bland's algorithm** or **Bland's anti-cycling rule**) is an algorithmic refinement of the simplex method for linear optimization.

With Bland's rule, the simplex algorithm solves feasible linear optimization problems without cycling.^{[1][2][3]} There are examples of degenerate linear optimization problems on which the original simplex algorithm would cycle forever. Such cycles are avoided by Bland's rule for choosing a column to enter the basis.

Bland's rule was developed by Robert G. Bland, now a professor of operations research at Cornell University.

64.1 Algorithm

One uses Bland's rule during an iteration of the simplex method to decide first what column (known as the *entering variable*) and then row (known as the *leaving variable*) in the tableau to pivot on. Assuming that the problem is to minimize the objective function, the algorithm is loosely defined as follows:

1. Choose the lowest-numbered (i.e., leftmost) nonbasic column with a negative (reduced) cost.
2. Now among the rows choose the one with the lowest ratio between the (transformed) right hand side and the coefficient in the pivot tableau where the coefficient is greater than zero. If the minimum ratio is shared by several rows, choose the row with the lowest-numbered column (variable) basic in it.

64.2 Extensions to oriented matroids

In the abstract setting of oriented matroids, Bland's rule cycles on some examples. A restricted class of oriented matroids on which Bland's rule avoids cycling has been termed “Bland oriented matroids” by Jack Edmonds. Another pivoting rule, the criss-cross algorithm, avoids cycles on all oriented-matroid linear-programs.^[4]

64.3 Notes

- [1] Bland (1977).
- [2] Christos H. Papadimitriou, Kenneth Steiglitz (1998-01-29). *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications. pp. 53–55.
- [3] Brown University - Department of Computer Science (2007-10-18). “Notes on the Simplex Algorithm”. Retrieved 2007-12-17.
- [4] Fukuda, Komei; Terlaky, Tamás (1997). Thomas M. Liebling and Dominique de Werra, ed. “Criss-cross methods: A fresh view on pivot algorithms”. *Mathematical Programming: Series B* **79** (1-3) (Amsterdam: North-Holland Publishing Co.). pp. 369–395. doi:10.1007/BF02614325. MR 1464775.

64.4 Further reading

- Bland, Robert G. (May 1977). “New finite pivoting rules for the simplex method”. *Mathematics of Operations Research* **2** (2): 103–107. doi:10.1287/moor.2.2.103. JSTOR 3689647. MR 459599.
- George B. Dantzig and Mukund N. Thapa. 2003. *Linear Programming 2: Theory and Extensions*. Springer-Verlag.
- Katta G. Murty, *Linear Programming*, Wiley, 1983.
- Evar D. Nering and Albert W. Tucker, 1993, *Linear Programs and Related Problems*, Academic Press.
- M. Padberg, *Linear Optimization and Extensions*, Second Edition, Springer-Verlag, 1999.
- Christos H. Papadimitriou and Kenneth Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Corrected republication with a new preface, Dover. (computer science)
- Alexander Schrijver, *Theory of Linear and Integer Programming*. John Wiley & sons, 1998, ISBN 0-471-98232-6 (mathematical)

- Michael J. Todd (February 2002). “The many facets of linear programming”. *Mathematical Programming* **91** (3). (Invited survey, from the International Symposium on Mathematical Programming.)

Chapter 65

BOBYQA

BOBYQA (Bound Optimization BY Quadratic Approximation)^[1] is a numerical optimization algorithm by Michael J. D. Powell. It is also the name of Powell's Fortran 77 implementation of the algorithm.

BOBYQA solves bound constrained optimization problems without using derivatives of the objective function, which makes it a derivative-free algorithm. The algorithm solves the problem using a trust region method that forms quadratic models by interpolation. One new point is computed on each iteration, usually by solving a trust region subproblem subject to the bound constraints, or alternatively, by choosing a point to replace an interpolation point so as to promote good linear independence in the interpolation conditions.

The same as NEWUOA, BOBYQA constructs the quadratic models by the least Frobenius norm updating [2] technique.

BOBYQA software was released on January 5th, 2009.^[3]

In the comment of the software's source code,^[4] it is said that the name BOBYQA denotes "Bound Approximation BY Quadratic Approximation", which should be a typo of "Bound Optimization BY Quadratic Approximation".

65.1 See also

- TOLMIN
- COBYLA
- UOBYQA
- NEWUOA
- LINCOA

65.2 References

- [1] Powell, M. J. D. (June 2009). The BOBYQA algorithm for bound constrained optimization without derivatives (Report). Department of Applied Mathematics and Theoretical Physics, Cambridge University. DAMTP 2009/NA06. Retrieved 2014-02-14.

- [2] Powell, M. J. D. (2004). "Least Frobenius norm updating of quadratic models that satisfy interpolation conditions". *Mathematical Programming* (Springer) **100**: 183–215. doi:10.1007/s10107-003-0490-7.
- [3] "A repository of Powell's software". Retrieved 2014-01-18.
- [4] "Source code of BOBYQA software". Retrieved 2014-02-14.

65.3 External links

- Source code of BOBYQA software

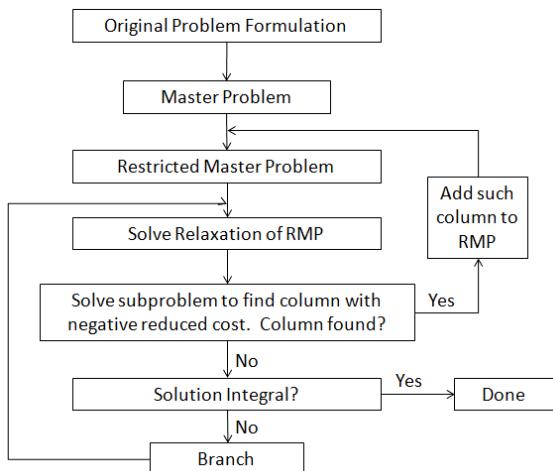
Chapter 66

Branch and price

In applied mathematics, **branch and price** is a method of combinatorial optimization for solving integer linear programming (ILP) and mixed integer linear programming (MILP) problems with many variables. The method is a hybrid of branch and bound and column generation methods.

66.1 Description of Algorithm

Branch and price is a branch and bound method in which at each node of the search tree, columns may be added to the LP relaxation. At the start of the algorithm, sets of columns are excluded from the LP relaxation in order to reduce the computational and memory requirements and then columns are added back to the LP relaxation as needed. The approach is based on the observation that for large problems most columns will be nonbasic and have their corresponding variable equal to zero in any optimal solution. Thus, the large majority of the columns are irrelevant for solving the problem.



Outline of branch and price algorithm. Adapted from [1]

The algorithm typically begins by using a reformulation, such as **Dantzig-Wolfe decomposition**, to form what is known as the **Master Problem**. The decomposition is performed to obtain a problem formulation that gives better bounds when the relaxation is solved than when

the relaxation of the original formulation is solved. But, the decomposition usually contains many variables and so a modified version, called the **Restricted Master Problem**, that only considers a subset of the columns is solved.^[2] Then, to check for optimality, a subproblem called the **pricing problem** is solved to find columns that can enter the basis and reduce the objective function (for a minimization problem). This involves finding a column that has a negative **reduced cost**. Note that the pricing problem itself may be difficult to solve but since it is not necessary to find the column with the most negative reduced cost, heuristic and local search methods can be used.^[3] The subproblem must only be solved to completion in order to prove that an optimal solution to the Restricted Master Problem is also an optimal solution to the Master Problem. Each time a column is found with negative reduced cost, it is added to the Restricted Master Problem and the relaxation is reoptimized. If no columns can enter the basis and the solution to the relaxation is not integer, then branching occurs.^[1]

Most branch and price algorithms are problem specific since the problem must be formulated in such a way so that effective branching rules can be formulated and so that the pricing problem is relatively easy to solve.^[4]

If cutting planes are used to tighten LP relaxations within a branch and cut algorithm, the method is known as **branch price and cut**.^[5]

66.2 Applications of Branch and Price

The branch and price method can be used to solve problems in a variety of application areas, including

- Graph multi-coloring.^[3] This is a generalization of the **graph coloring** problem in which each node in a graph must be assigned a preset number of colors and any nodes that share an edge cannot have a color in common. The objective is then to find the minimum number of colors needed to have a valid coloring. The multi-coloring problem can be used to model a variety of applications including job

scheduling and telecommunication channel assignment.

- Vehicle routing problems.^[2]
- Generalized assignment problem.^[6]

- Barnhart, Cynthia; Johnson, Ellis L.; Nemhauser, George L.; Savelsbergh, Martin W. P.; Vance, Pamela H. (1998), “Branch-and-price: column generation for solving huge integer programs”, *Operations Research* **46** (3): 316–329, doi:10.1287/opre.46.3.316, JSTOR 222825

66.3 See also

- Branch and cut
- Branch and bound
- Delayed column generation

66.4 External References

- Lecture slides on branch and price
- More lecture slides
- Even more lecture slides
- Prototype code for a generic branch and price algorithm
- BranchAndCut.org FAQ
- SCIP an open source framework for branch-cut-and-price and a mixed integer programming solver
- ABACUS - A Branch-And-CUT System - open source software

66.5 References

- [1] Akella, M.; S. Gupta; A. Sarkar. “Branch and Price: Column Generation for Solving Huge Integer Programs”.
- [2] Feillet, Dominique (2010). “A tutorial on column generation and branch-and-price for vehicle routing problems”. *4OR* **8** (4): 407–424. doi:10.1007/s10288-010-0130-z.
- [3] Mehrota, Anuj; M.A. Trick (2007). “A Branch-and-price Approach for Graph Multi-Coloring”. *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*: 15–29.
- [4] Lubbecke, M. “Generic Branch-Cut-and-Price”.
- [5] Desrosiers, J.; M.E. Lubbecke (2010). “Branch-Price-and-Cut Algorithms”. *Wiley Encyclopedia of Operations Research and Management Science*.
- [6] Savelsbergh, M. (1997). “A branch-and-price algorithm for the generalized assignment problem”. *Operations Research*. 831-841. doi:10.1287/opre.45.6.831.

Chapter 67

CMA-ES

CMA-ES stands for Covariance Matrix Adaptation Evolution Strategy. Evolution strategies (ES) are stochastic, derivative-free methods for numerical optimization of non-linear or non-convex continuous optimization problems. They belong to the class of evolutionary algorithms and evolutionary computation. An evolutionary algorithm is broadly based on the principle of biological evolution, namely the repeated interplay of variation (via recombination and mutation) and selection: in each generation (iteration) new individuals (candidate solutions, denoted as x) are generated by variation, usually in a stochastic way, of the current parental individuals. Then, some individuals are selected to become the parents in the next generation based on their fitness or objective function value $f(x)$. Like this, over the generation sequence, individuals with better and better f -values are generated.

In an evolution strategy, new candidate solutions are sampled according to a multivariate normal distribution in the \mathbb{R}^n . Recombination amounts to selecting a new mean value for the distribution. Mutation amounts to adding a random vector, a perturbation with zero mean. Pairwise dependencies between the variables in the distribution are represented by a covariance matrix. The covariance matrix adaptation (CMA) is a method to update the covariance matrix of this distribution. This is particularly useful, if the function f is ill-conditioned.

Adaptation of the covariance matrix amounts to learning a second order model of the underlying objective function similar to the approximation of the inverse Hessian matrix in the Quasi-Newton method in classical optimization. In contrast to most classical methods, fewer assumptions on the nature of the underlying objective function are made. Only the ranking between candidate solutions is exploited for learning the sample distribution and neither derivatives nor even the function values themselves are required by the method.

67.1 Principles

Two main principles for the adaptation of parameters of the search distribution are exploited in the CMA-ES algorithm.

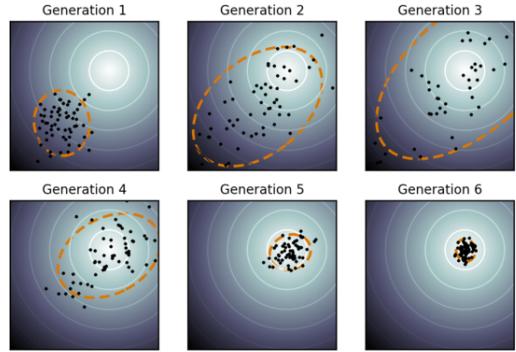


Illustration of an actual optimization run with covariance matrix adaptation on a simple two-dimensional problem. The spherical optimization landscape is depicted with solid lines of equal f -values. The population (dots) is much larger than necessary, but clearly shows how the distribution of the population (dotted line) changes during the optimization. On this simple problem, the population concentrates over the global optimum within a few generations.

First, a maximum-likelihood principle, based on the idea to increase the probability of successful candidate solutions and search steps. The mean of the distribution is updated such that the likelihood of previously successful candidate solutions is maximized. The covariance matrix of the distribution is updated (incrementally) such that the likelihood of previously successful search steps is increased. Both updates can be interpreted as a natural gradient descent. Also, in consequence, the CMA conducts an iterated principal components analysis of successful search steps while retaining *all* principal axes. Estimation of distribution algorithms and the Cross-Entropy Method are based on very similar ideas, but estimate (non-incrementally) the covariance matrix by maximizing the likelihood of successful solution *points* instead of successful search *steps*.

Second, two paths of the time evolution of the distribution mean of the strategy are recorded, called search or evolution paths. These paths contain significant information about the correlation between consecutive steps. Specifically, if consecutive steps are taken in a similar direction, the evolution paths become long. The evolution paths are exploited in two ways. One path is used for the

covariance matrix adaptation procedure in place of single successful search steps and facilitates a possibly much faster variance increase of favorable directions. The other path is used to conduct an additional step-size control. This step-size control aims to make consecutive movements of the distribution mean orthogonal in expectation. The step-size control effectively prevents premature convergence yet allowing fast convergence to an optimum.

67.2 Algorithm

In the following the most commonly used $(\mu/\mu_w, \lambda)$ -CMA-ES is outlined, where in each iteration step a weighted combination of the μ best out of λ new candidate solutions is used to update the distribution parameters. The main loop consists of three main parts: 1) sampling of new solutions, 2) re-ordering of the sampled solutions based on their fitness, 3) update of the internal state variables based on the re-ordered samples. A pseudocode of the algorithm looks as follows.

```

set λ // number of samples per iteration, at least two,
generally > 4 initialize m , σ , C = I , pσ = 0 ,
pc = 0 // initialize state variables while not terminate
// iterate for i in {1...λ} // sample λ new solutions and
evaluate them xi = sample_multivariate_normal(mean=
m , covariance_matrix= σ²C ) fi = fitness( xi ) x1...λ
← xs(1)...s(λ) with s(i) = argsort( f1...λ , i ) // sort solutions
m' = m // we need later m - m' and xi - m'
m ← update_m (x1, ..., xλ) // move mean to better
solutions pσ ← update_ps (pσ, σ⁻¹C⁻¹/²(m - m'))
// update isotropic evolution path pc ← update_pc (p,
σ⁻¹(m - m'), ||pσ||) // update anisotropic evolution path
C ← update_C (C, pc, (x1 - m')/σ, ..., (xλ - m')/σ)
// update covariance matrix σ ← update_sigma (σ, ||pσ||)
// update step-size using isotropic path length return m
or x1
```

The order of the five update assignments is relevant. In the following, the update equations for the five state variables are specified.

Given are the search space dimension n and the iteration step k . The five state variables are

$$m_k \in \mathbb{R}^n$$

$$\sigma_k > 0$$

C_k , a symmetric and positive definite $n \times n$ covariance matrix with $C_0 = I$ and

$$pσ ∈ ℝⁿ, p_c ∈ ℝⁿ$$

The iteration starts with sampling $\lambda > 1$ candidate solutions $x_i \in \mathbb{R}^n$ from a multivariate normal distribution $\mathcal{N}(m_k, \sigma_k^2 C_k)$, i.e. for $i = 1, \dots, \lambda$

$$\begin{aligned} x_i &\sim \mathcal{N}(m_k, \sigma_k^2 C_k) \\ &\sim m_k + \sigma_k \times \mathcal{N}(0, C_k) \end{aligned}$$

The second line suggests the interpretation as perturbation (mutation) of the current favorite solution vector m_k (the distribution mean vector). The candidate solutions x_i are evaluated on the objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ to be minimized. Denoting the f -sorted candidate solutions as

$$\{x_{i:\lambda} \mid i = 1 \dots \lambda\} = \{x_i \mid i = 1 \dots \lambda\} \text{ and } f(x_{1:\lambda}) \leq \dots \leq f(x_{\mu:\lambda}) \leq \dots$$

the new mean value is computed as

$$\begin{aligned} m_{k+1} &= \sum_{i=1}^{\mu} w_i x_{i:\lambda} \\ &= m_k + \sum_{i=1}^{\mu} w_i (x_{i:\lambda} - m_k) \end{aligned}$$

where the positive (recombination) weights $w_1 \geq w_2 \geq \dots \geq w_\mu > 0$ sum to one. Typically, $\mu \leq \lambda/2$ and the weights are chosen such that $\mu_w := 1 / \sum_{i=1}^{\mu} w_i^2 \approx \lambda/4$. The only feedback used from the objective function here and in the following is an ordering of the sampled candidate solutions due to the indices $i : \lambda$.

The step-size σ_k is updated using *cumulative step-size adaptation* (CSA), sometimes also denoted as *path length control*. The evolution path (or search path) $pσ$ is updated first.

$$\begin{aligned} pσ &\leftarrow \underbrace{(1 - cσ)}_{\text{factor discount}} pσ + \sqrt{1 - (1 - cσ)^2} \underbrace{\sqrt{\mu_w} C_k^{-1/2} \frac{m_{k+1} - m_k}{σ_k}}_{\substack{\text{variance discounted for complements} \\ \text{of displacement } m \\ \text{as distributed } \mathcal{N}(0, I) \text{ selection neutral under}}} \\ σ_{k+1} &= σ_k \times \exp \left(\underbrace{\frac{cσ}{dσ} \left(\frac{\|pσ\|}{E\|\mathcal{N}(0, I)\|} - 1 \right)}_{\substack{\text{selection neutral under 0 about unbiased}}} \right) \end{aligned}$$

where

$cσ^{-1} \approx n/3$ is the backward time horizon for the evolution path $pσ$ and larger than one,

$μ_w = (\sum_{i=1}^{\mu} w_i^2)^{-1}$ is the variance effective selection mass and $1 \leq μ_w \leq \mu$ by definition of w_i ,

$C_k^{-1/2} = \sqrt{C_k}^{-1} = \sqrt{C_k^{-1}}$ is the unique symmetric square root of the inverse of C_k , and

d_σ is the damping parameter usually close to one. For $d_\sigma = \infty$ or $c_\sigma = 0$ the step-size remains unchanged.

The step-size σ_k is increased if and only if $\|p_\sigma\|$ is larger than the expected value

$$\begin{aligned} E\|\mathcal{N}(0, I)\| &= \sqrt{2} \Gamma((n+1)/2) / \Gamma(n/2) \\ &\approx \sqrt{n} (1 - 1/(4n) + 1/(21n^2)) \end{aligned}$$

and decreased if it is smaller. For this reason, the step-size update tends to make consecutive steps C_k^{-1} -conjugate, in that after the adaptation has been successful $\left(\frac{m_{k+2}-m_{k+1}}{\sigma_{k+1}}\right)^T C_k^{-1} \frac{m_{k+1}-m_k}{\sigma_k} \approx 0$.^[1]

Finally, the covariance matrix is updated, where again the respective evolution path is updated first.

$$\begin{aligned} p_c &\leftarrow \underbrace{(1 - c_c)}_{\text{factor discount}} \underbrace{p_c + \mathbf{1}_{[0, \alpha\sqrt{n}]}(\|p_\sigma\|) \sqrt{1 - (1 - c_c)^2} \sqrt{\mu_w}}_{\text{function indicator}} \underbrace{\text{as distributed } \mathcal{N}(0, C_k^{-1})}_{\text{selection neutral under}} \\ C_{k+1} &= \underbrace{(1 - c_1 - c_\mu + c_s)}_{\text{factor discount}} \underbrace{C_k + c_1 p_c p_c^T}_{\text{matrix one rank}} + c_\mu \sum_{i=1}^{\mu} w_i \underbrace{\frac{x_{i:\lambda}}{\|x_{i:\lambda}\|}}_{\text{population size, offspring number}}^T \underbrace{\frac{w_i}{\mu}}_{\text{number of parents/points for recombination weights}} \underbrace{\frac{x_{i:\lambda}}{\|x_{i:\lambda}\|}}_{\text{rank-one matrix}} \end{aligned}$$

where T denotes the transpose and

$c_c^{-1} \approx n/4$ is the backward time horizon for the evolution path p_c and larger than one,

$\alpha \approx 1.5$ and the indicator function $\mathbf{1}_{[0, \alpha\sqrt{n}]}(\|p_\sigma\|)$ evaluates to one iff $\|p_\sigma\| \in [0, \alpha\sqrt{n}]$ or, in other words, $\|p_\sigma\| \leq \alpha\sqrt{n}$, which is usually the case,

$$c_s = (1 - \mathbf{1}_{[0, \alpha\sqrt{n}]}(\|p_\sigma\|))^2 c_1 c_c (2 - c_c)$$

$c_1 \approx 2/n^2$ is the learning rate for the rank-one update of the covariance matrix and

$c_\mu \approx \mu_w/n^2$ is the learning rate for the rank- μ update of the covariance matrix and must not exceed $1 - c_c$.

The covariance matrix update tends to increase the likelihood for p_c and for $(x_{i:\lambda} - m_k)/\sigma_k$ to be sampled from $\mathcal{N}(0, C_{k+1})$. This completes the iteration step.

The number of candidate samples per iteration, λ , is not determined a priori and can vary in a wide range. Smaller values, for example $\lambda = 10$, lead to more local search behavior. Larger values, for example $\lambda = 10n$ with default value $\mu_w \approx \lambda/4$, render the search more global. Sometimes the algorithm is repeatedly restarted with increasing λ by a factor of two for each restart.^[2] Besides of setting λ (or possibly μ instead, if for example λ is pre-determined by the number of available processors), the above introduced parameters are not specific to the given objective function and therefore not meant to be modified by the user.

67.3 Example code in MATLAB/Octave

```
function xmin=purecmaes % (mu/mu_w, lambda)
CMA-ES % -----
----- % User defined input parameters
----- need to be edited strfitnessfct = 'frosenbrock'; % name of objective/fitness function N = 20; % number of objective variables/problem dimension xmean =
rand(N,1)m% objective variables initial point sigma =
0.3; % coordinate wise standard deviation (step size)
stopfitness = 1e-10; % stop if fitness < stopfitness (minimization)
stopeval = 1e3*N^2; % stop after stopeval
number of function evaluations % Strategy parameter
setting: Selection lambda = 4+floor(3*log(N)); %
population size, offspring number mu = lambda/2; %
number of parents/points for recombination weights =
log(mu+1/2)*log(1/mu); % muXone array for
rank-one matrix weighted recombination mu = floor(mu);
weights = weights/sum(weights); % normalize recombination
weights array mueff=sum(weights)^2/sum(weights.^2); %
variance-effectiveness of sum w_i x_i % Strategy
parameter setting: Adaptation cc = (4+mueff/N) / (N+4 +
2*mueff/N); % time constant for cumulation for C cs =
(mueff+2) / (N+mueff+5); % t-const for cumulation for
sigma control c1 = 2 / ((N+1.3)^2+mueff); % learning
rate for rank-one update of C cmu = min(1-c1, 2 *
(mueff-2+1/mueff) / ((N+2)^2+mueff)); % and for
rank-mu update damps = 1 + 2*max(0, sqrt((mueff-
1)/(N+1))-1) + cs; % damping for sigma % usually
close to 1 % Initialize dynamic (internal) strategy
parameters and constants pc = zeros(N,1); ps = zeros(N,1);
% evolution paths for C and sigma B = eye(N,N); %
B defines the coordinate system D = ones(N,1); %
diagonal D defines the scaling C = B * diag(D.^2) * B';
% covariance matrix C invsqrtC = B * diag(D.^-1) *
B'; % C^-1/2 eigeneval = 0; % track update of B and D
chiN=N^0.5*(1-1/(4*N)+1/(21*N^2)); % expectation
of % ||N(0,I)|| == norm(randn(N,1)) %
----- Generation Loop ----- counteval
```

```

= 0; % the next 40 lines contain the 20 lines of interesting
code while counteval < stopeval % Generate and evaluate
lambda offspring for k=1:lambda, arx(:,k) = xmean +
sigma * B .* (D .* randn(N,1)); % m + sig * Normal(0,C)
arfitness(k) = feval(strfitnessfct, arx(:,k)); % objective
function call counteval = counteval+1; end % Sort
by fitness and compute weighted mean into xmean
[arfitness, arindex] = sort(arfitness); % minimization
xold = xmean; xmean = arx(:,arindex(1:mu))*weights;
% recombination, new mean value % Cumulation:
Update evolution paths ps = (1-cs)*ps ... + sqrt(cs*(2-
cs)*mueff) * invsqrtC * (xmean-xold) / sigma; hsig =
norm(ps)/sqrt(1-(1-cs)^(2*cumulative/lambda))/chiN <
1.4 + 2/(N+1); pc = (1-cc)*pc ... + hsig * sqrt(cc*(2-
cc)*mueff) * (xmean-xold) / sigma; % Adapt covariance
matrix C artmp = (1/sigma) * (arx(:,arindex(1:mu))-_
repmat(xold,1,mu)); C = (1-c1-cmu) * C ... % regard
old matrix + c1 * (pc*pc' ... % plus rank one update
+ (1-hsig) * cc*(2-cc) * C) ... % minor correction if
hsig==0 + cmu * artmp * diag(weights) * artmp'; % plus
rank mu update % Adapt step size sigma sigma = sigma *
exp((cs/damps)*(norm(ps)/chiN - 1)); % Decom-
position of C into B*diag(D.^2)*B' (diagonalization)
if counteval - eigeneval > lambda/(c1+cmu)/N/10 % to achieve O(N^2) eigeneval = counteval; C = triu(C) +
triu(C,1)'; % enforce symmetry [B,D] = eig(C); % eigen decomposition, B==normalized eigenvectors D =
sqrt(diag(D)); % D is a vector of standard deviations now invsqrtC = B * diag(D.^-1) * B'; end % Break, if
fitness is good enough or condition exceeds 1e14, better
termination methods are advisable if arfitness(1) <=
stopfitness || max(D) > 1e7 * min(D) break; end end %
while, end generation loop xmin = arx(:, arindex(1));
% Return best point of last iteration. % Notice that
xmean is expected to be even % better. % -----
----- function
f=frosenbrock(x) if size(x,1) < 2 error('dimension must
be greater one'); end f = 100*sum((x(1:end-1).^2 -
x(2:end)).^2) + sum((x(1:end-1)-1).^2);

```

67.4 Theoretical Foundations

Given the distribution parameters—mean, variances and covariances—the **normal probability distribution** for sampling new candidate solutions is the **maximum entropy probability distribution** over \mathbb{R}^n , that is, the sample distribution with the minimal amount of prior information built into the distribution. More considerations on the update equations of CMA-ES are made in the following.

67.4.1 Variable Metric

The CMA-ES implements a stochastic variable-metric method. In the very particular case of a convex-quadratic

objective function

$$f(x) = \frac{1}{2}(x - x^*)^T H(x - x^*)$$

the covariance matrix C_k adapts to the inverse of the Hessian matrix H , up to a scalar factor and small random fluctuations. More general, also on the function $g \circ f$, where g is strictly increasing and therefore order preserving and f is convex-quadratic, the covariance matrix C_k adapts to H^{-1} , up to a scalar factor and small random fluctuations.

67.4.2 Maximum-Likelihood Updates

The update equations for mean and covariance matrix maximize a **likelihood** while resembling an expectation-maximization algorithm. The update of the mean vector m maximizes a log-likelihood, such that

$$m_{k+1} = \arg \max_m \sum_{i=1}^{\mu} w_i \log p_N(x_{i:\lambda} | m)$$

where

$$\log p_N(x) = -\frac{1}{2} \log \det(2\pi C) - \frac{1}{2}(x - m)^T C^{-1}(x - m)$$

denotes the log-likelihood of x from a multivariate normal distribution with mean m and any positive definite covariance matrix C . To see that m_{k+1} is independent of C remark first that this is the case for any diagonal matrix C , because the coordinate-wise maximizer is independent of a scaling factor. Then, rotation of the data points or choosing C non-diagonal are equivalent.

The rank- μ update of the covariance matrix, that is, the right most summand in the update equation of C_k , maximizes a log-likelihood in that

$$\sum_{i=1}^{\mu} w_i \frac{x_{i:\lambda} - m_k}{\sigma_k} \left(\frac{x_{i:\lambda} - m_k}{\sigma_k} \right)^T = \arg \max_C \sum_{i=1}^{\mu} w_i \log p_N \left(\frac{x_{i:\lambda} - m_k}{\sigma_k} \right)$$

for $\mu \geq n$ (otherwise C is singular, but substantially the same result holds for $\mu < n$). Here, $p_N(x|C)$ denotes the likelihood of x from a multivariate normal distribution with zero mean and covariance matrix C . Therefore, for $c_1 = 0$ and $c_\mu = 1$, C_{k+1} is the above maximum-likelihood estimator. See estimation of covariance matrices for details on the derivation.

67.4.3 Natural Gradient Descent in the Space of Sample Distributions

Akimoto *et al.*^[3] and Glasmachers *et al.*^[4] discovered independently that the update of the distribution parameters resembles the descend in direction of a sampled **natural gradient** of the expected objective function value $E f(x)$ (to be minimized), where the expectation is taken under the sample distribution. With the parameter setting of $c_\sigma = 0$ and $c_1 = 0$, i.e. without step-size control and rank-one update, CMA-ES can thus be viewed as an instantiation of **Natural Evolution Strategies** (NES).^{[3][4]} The natural gradient is independent of the parameterization of the distribution. Taken with respect to the parameters θ of the sample distribution p , the gradient of $E f(x)$ can be expressed as

$$\begin{aligned}\nabla_\theta E(f(x)|\theta) &= \nabla_\theta \int_{\mathbb{R}^n} f(x)p(x)dx \\ &= \int_{\mathbb{R}^n} f(x)\nabla_\theta p(x)dx \\ &= \int_{\mathbb{R}^n} f(x)p(x)\nabla_\theta \ln p(x)dx \\ &= E(f(x)\nabla_\theta \ln p(x|\theta))\end{aligned}$$

where $p(x) = p(x|\theta)$ depends on the parameter vector θ , the so-called **score function**, $\nabla_\theta \ln p(x|\theta) = \frac{\nabla_\theta p(x)}{p(x)}$, indicates the relative sensitivity of p w.r.t. θ , and the expectation is taken with respect to the distribution p . The **natural gradient** of $E f(x)$, complying with the **Fisher information metric** (an informational distance measure between probability distributions and the curvature of the relative entropy), now reads

$$\tilde{\nabla} E(f(x)|\theta) = F_\theta^{-1} \nabla_\theta E(f(x)|\theta)$$

where the **Fisher information matrix** F_θ is the expectation of the **Hessian** of -lnp and renders the expression independent of the chosen parameterization. Combining the previous equalities we get

$$\begin{aligned}\tilde{\nabla} E(f(x)|\theta) &= F_\theta^{-1} E(f(x)\nabla_\theta \ln p(x|\theta)) \\ &= E(f(x)F_\theta^{-1} \nabla_\theta \ln p(x|\theta))\end{aligned}$$

A Monte Carlo approximation of the latter expectation takes the average over λ samples from p

$$\tilde{\nabla} \hat{E}_\theta(f) := - \sum_{i=1}^{\lambda} \underbrace{w_i}_{\substack{\text{weight preference} \\ \text{from direction candidate } x_{i:\lambda}}} \underbrace{F_\theta^{-1} \nabla_\theta \ln p(x_{i:\lambda}|\theta)}_{\substack{\text{from direction candidate } x_{i:\lambda}}}$$

where the notation $i : \lambda$ from above is used and therefore w_i are monotonously decreasing in i .

Ollivier *et al.*^[3] finally found a rigorous formulation for the more robust weights, w_i , as they are defined in the CMA-ES (weights are zero for $i > \mu$), formulated as consistent estimator for the **CDF** of $f(X)$, $X \sim p(\cdot|\theta)$ at the point $f(x_{i:\lambda})$, composed with a fixed monotonous decreased transformation w , i.e.,

$$w_i = w \left(\frac{\text{rank}(f(x_{i:\lambda})) - 1/2}{\lambda} \right)$$

Let

$$\theta = [m_k^T \text{vec}(C_k)^T \sigma_k]^T \in \mathbb{R}^{n+n^2+1}$$

such that $p(\cdot|\theta)$ is the density of the **multivariate normal distribution** $\mathcal{N}(m_k, \sigma_k^2 C_k)$. Then, we have an explicit expression for the inverse of the Fisher information matrix where σ_k is fixed

$$F_{\theta|\sigma_k}^{-1} = \begin{bmatrix} \sigma_k^2 C_k & 0 \\ 0 & 2C_k \otimes C_k \end{bmatrix}$$

and for

$$\ln p(x|\theta) = \ln p(x|m_k, \sigma_k^2 C_k) = -\frac{1}{2}(x-m_k)^T \sigma_k^{-2} C_k^{-1} (x-m_k) - \frac{1}{2} \ln d$$

and, after some calculations, the updates in the CMA-ES turn out as^[3]

$$\begin{aligned}m_{k+1} &= m_k - \underbrace{[\tilde{\nabla} \hat{E}_\theta(f)]_{1,\dots,n}}_{\substack{\text{mean for gradient natural}}} \\ &= m_k + \sum_{i=1}^{\lambda} w_i (x_{i:\lambda} - m_k)\end{aligned}$$

and

$$\begin{aligned}C_{k+1} &= C_k + c_1 (p_c p_c^T - C_k) - c_\mu \overbrace{\text{mat}([\tilde{\nabla} \hat{E}_\theta(f)]_{n+1,\dots,n+n^2})}^{\substack{\text{matrix covariance for gradient natural}}} \\ &= C_k + c_1 (p_c p_c^T - C_k) + c_\mu \sum_{i=1}^{\lambda} w_i \left(\frac{x_{i:\lambda} - m_k}{\sigma_k} \left(\frac{x_{i:\lambda} - m_k}{\sigma_k} \right)^T \right)\end{aligned}$$

where mat forms the proper matrix from the respective natural gradient sub-vector. That means, setting $c_1 = c_\sigma = 0$, the CMA-ES updates descend in direction of the approximation $\tilde{\nabla} \hat{E}_\theta(f)$ of the natural gradient while using different step-sizes (learning rates) for the orthogonal parameters m and C respectively.

67.4.4 Stationarity or Unbiasedness

It is comparatively easy to see that the update equations of CMA-ES satisfy some stationarity conditions, in that they are essentially unbiased. Under neutral selection, where $x_{i:\lambda} \sim \mathcal{N}(m_k, \sigma_k^2 C_k)$, we find that

$$E(m_{k+1} | m_k) = m_k$$

and under some mild additional assumptions on the initial conditions

$$E(\log \sigma_{k+1} | \sigma_k) = \log \sigma_k$$

and with an additional minor correction in the covariance matrix update for the case where the indicator function evaluates to zero, we find

$$E(C_{k+1} | C_k) = C_k$$

67.4.5 Invariance

Invariance properties imply uniform performance on a class of objective functions. They have been argued to be an advantage, because they allow to generalize and predict the behavior of the algorithm and therefore strengthen the meaning of empirical results obtained on single functions. The following invariance properties have been established for CMA-ES.

- Invariance under order-preserving transformations of the objective function value f , in that for any $h : \mathbb{R}^n \rightarrow \mathbb{R}$ the behavior is identical on $f : x \mapsto g(h(x))$ for all strictly increasing $g : \mathbb{R} \rightarrow \mathbb{R}$. This invariance is easy to verify, because only the f -ranking is used in the algorithm, which is invariant under the choice of g .
- Scale-invariance, in that for any $h : \mathbb{R}^n \rightarrow \mathbb{R}$ the behavior is independent of $\alpha > 0$ for the objective function $f : x \mapsto h(\alpha x)$ given $\sigma_0 \propto 1/\alpha$ and $m_0 \propto 1/\alpha$.
- Invariance under rotation of the search space in that for any $h : \mathbb{R}^n \rightarrow \mathbb{R}$ and any $z \in \mathbb{R}^n$ the behavior on $f : x \mapsto h(Rx)$ is independent of the orthogonal matrix R , given $m_0 = R^{-1}z$. More general, the algorithm is also invariant under general linear transformations R when additionally the initial covariance matrix is chosen as $R^{-1}R^{-1T}$.

Any serious parameter optimization method should be translation invariant, but most methods do not exhibit all the above described invariance properties. A prominent example with the same invariance properties is the Nelder–Mead method, where the initial simplex must be chosen respectively.

67.4.6 Convergence

Conceptual considerations like the scale-invariance property of the algorithm, the analysis of simpler evolution strategies, and overwhelming empirical evidence suggest that the algorithm converges on a large class of functions fast to the global optimum, denoted as x^* . On some functions, convergence occurs independently of the initial conditions with probability one. On some functions the probability is smaller than one and typically depends on the initial m_0 and σ_0 . Empirically, the fastest possible convergence rate in k for rank-based direct search methods can often be observed (depending on the context denoted as *linear* or *log-linear* or *exponential* convergence). Informally, we can write

$$\|m_k - x^*\| \approx \|m_0 - x^*\| \times e^{-ck}$$

for some $c > 0$, and more rigorously

$$\frac{1}{k} \sum_{i=1}^k \log \frac{\|m_i - x^*\|}{\|m_{i-1} - x^*\|} = \frac{1}{k} \log \frac{\|m_k - x^*\|}{\|m_0 - x^*\|} \rightarrow -c < 0 \quad \text{for } k \rightarrow \infty$$

or similarly,

$$E \log \frac{\|m_k - x^*\|}{\|m_{k-1} - x^*\|} \rightarrow -c < 0 \quad \text{for } k \rightarrow \infty.$$

This means that on average the distance to the optimum decreases in each iteration by a “constant” factor, namely by $\exp(-c)$. The convergence rate c is roughly $0.1\lambda/n$, given λ is not much larger than the dimension n . Even with optimal σ and C , the convergence rate c cannot largely exceed $0.25\lambda/n$, given the above recombination weights w_i are all non-negative. The actual linear dependencies in λ and n are remarkable and they are in both cases the best one can hope for in this kind of algorithm. Yet, a rigorous proof of convergence is missing.

67.4.7 Interpretation as Coordinate System Transformation

Using a non-identity covariance matrix for the multivariate normal distribution in evolution strategies is equivalent to a coordinate system transformation

of the solution vectors,^[5] mainly because the sampling equation

$$\begin{aligned}x_i &\sim m_k + \sigma_k \times \mathcal{N}(0, C_k) \\&\sim m_k + \sigma_k \times C_k^{1/2} \mathcal{N}(0, I)\end{aligned}$$

can be equivalently expressed in an “encoded space” as

$$\underbrace{C_k^{-1/2} x_i}_{\text{space encode the in represented}} \sim \underbrace{C_k^{-1/2} m_k}_{\text{}} + \sigma_k \times \mathcal{N}(0, I)$$

The covariance matrix defines a bijective transformation (encoding) for all solution vectors into a space, where the sampling takes place with identity covariance matrix. Because the update equations in the CMA-ES are invariant under linear coordinate system transformations, the CMA-ES can be re-written as an adaptive encoding procedure applied to a simple evolution strategy with identity covariance matrix.^[5] This adaptive encoding procedure is not confined to algorithms that sample from a multivariate normal distribution (like evolution strategies), but can in principle be applied to any iterative search method.

67.5 Performance in Practice

In contrast to most other evolutionary algorithms, the CMA-ES is, from the users perspective, quasi parameter-free. The user has to choose an initial solution point, $m_0 \in \mathbb{R}^n$, and the initial step-size, $\sigma_0 > 0$. Optionally, the number of candidate samples λ (population size) can be modified by the user in order to change the characteristic search behavior (see above) and termination conditions can or should be adjusted to the problem at hand.

The CMA-ES has been empirically successful in hundreds of applications and is considered to be useful in particular on non-convex, non-separable, ill-conditioned, multi-modal or noisy objective functions. The search space dimension ranges typically between two and a few hundred. Assuming a black-box optimization scenario, where gradients are not available (or not useful) and function evaluations are the only considered cost of search, the CMA-ES method is likely to be outperformed by other methods in the following conditions:

- on low-dimensional functions, say $n < 5$, for example by the downhill simplex method or surrogate-based methods (like kriging with expected improvement);
- on separable functions without or with only negligible dependencies between the design variables in particular in the case of multi-modality or large dimension, for example by differential evolution;

- on (nearly) convex-quadratic functions with low or moderate condition number of the Hessian matrix, where BFGS or NEWUOA are typically ten times faster;
- on functions that can already be solved with a comparatively small number of function evaluations, say no more than $10n$, where CMA-ES is often slower than, for example, NEWUOA or Multilevel Coordinate Search (MCS).

On separable functions, the performance disadvantage is likely to be most significant in that CMA-ES might not be able to find at all comparable solutions. On the other hand, on non-separable functions that are ill-conditioned or rugged or can only be solved with more than $100n$ function evaluations, the CMA-ES shows most often superior performance.

67.6 Variations and Extensions

The (1+1)-CMA-ES^[6] generates only one candidate solution per iteration step which becomes the new distribution mean if it is better than the current mean. For $c_c = 1$ the (1+1)-CMA-ES is a close variant of Gaussian adaptation. Some Natural Evolution Strategies are close variants of the CMA-ES with specific parameter settings. Natural Evolution Strategies do not utilize evolution paths (that means in CMA-ES setting $c_c = c_\sigma = 1$) and they formalize the update of variances and covariances on a Cholesky factor instead of a covariance matrix. The CMA-ES has also been extended to multiobjective optimization as MO-CMA-ES.^[7] Another remarkable extension has been the addition of a negative update of the covariance matrix with the so-called active CMA.^[8]

With the advent of niching methods in evolutionary strategies, the question of an optimal niche radius arises. An “adaptive individual niche radius” is introduced in^[9]

67.7 See also

- Global optimization
- Stochastic optimization

67.8 References

- [1] Hansen, N. (2006), “The CMA evolution strategy: a comparing review”, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, Springer, pp. 1769–1776
- [2] Auger, A.; N. Hansen (2005). “A Restart CMA Evolution Strategy With Increasing Population Size”. 2005

IEEE Congress on Evolutionary Computation, Proceedings. IEEE. pp. 1769–1776.

- [3] Akimoto, Y.; Y. Nagata and I. Ono and S. Kobayashi (2010). “Bidirectional Relation between CMA Evolution Strategies and Natural Evolution Strategies”. *Parallel Problem Solving from Nature, PPSN XI*. Springer. pp. 154–163.
- [4] Glasmachers, T.; T. Schaul, Y. Sun, D. Wierstra and J. Schmidhuber (2010). “Exponential Natural Evolution Strategies”. *Genetic and Evolutionary Computation Conference GECCO*. Portland, OR.
- [5] Hansen, N. (2008). “Adaptive Encoding: How to Render Search Coordinate System Invariant”. *Parallel Problem Solving from Nature, PPSN X*. Springer. pp. 205–214.
- [6] Igel, C.; T. Suttorp and N. Hansen (2006). “A Computational Efficient Covariance Matrix Update and a (1+1)-CMA for Evolution Strategies”. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. ACM Press. pp. 453–460.
- [7] Igel, C.; N. Hansen and S. Roth (2007). “Covariance Matrix Adaptation for Multi-objective Optimization”. *Evolutionary Computation* (MIT press) **15** (1): 1–28. doi:10.1162/evco.2007.15.1.1. PMID 17388777.
- [8] Jastrebski, G.A.; D.V. Arnold (2006). “Improving Evolution Strategies through Active Covariance Matrix Adaptation”. *2006 IEEE World Congress on Computational Intelligence, Proceedings*. IEEE. pp. 9719–9726. doi:10.1109/CEC.2006.1688662.
- [9] Shir, Ofer M.; Bäck, Thomas (2006). Springer. pp. 142–151. Missing or empty title= (help)

67.9 Bibliography

- Hansen N, Ostermeier A (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, **9**(2) pp. 159–195.
- Hansen N, Müller SD, Koumoutsakos P (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, **11**(1) pp. 1–18.
- Hansen N, Kern S (2004). Evaluating the CMA evolution strategy on multimodal test functions. In Xin Yao et al., editors, *Parallel Problem Solving from Nature - PPSN VIII*, pp. 282–291, Springer.
- Igel C, Hansen N, Roth S (2007). Covariance Matrix Adaptation for Multi-objective Optimization. *Evolutionary Computation*, **15**(1) pp. 1–28.

67.10 External links

- A short introduction to CMA-ES by N. Hansen
- The CMA Evolution Strategy: A Tutorial
- CMA-ES source code page

Chapter 68

COBYLA

Constrained optimization by linear approximation (**COBYLA**) is a numerical optimization method for constrained problems where the derivative of the objective function is not known, invented by Michael J. D. Powell. That is, COBYLA can find the vector $\vec{x} \in \mathcal{S}$ with $\mathcal{S} \subseteq \mathbb{R}^n$ that has the minimal (or maximal) $f(\vec{x})$ without knowing the gradient of f . COBYLA is also the name of Powell's software implementation of the algorithm in Fortran.^[1]

Powell invented COBYLA while working for Westland Helicopters.^[2] It proceeds by iteratively approximating the actual objective function f with linear programs.

68.1 References

- [1] Andrew R. Conn; Katya Scheinberg; Ph. L. Toint (1997). “On the convergence of derivative-free methods for unconstrained optimization”. *Approximation theory and optimization: tributes to MJD Powell*. pp. 83–108.
- [2] M. J. D. Powell (2007). A view of algorithms for optimization without derivatives. Cambridge University Technical Report DAMTP 2007.

68.2 See also

- TOLMIN
- UOBYQA
- NEWUOA
- BOBYQA
- LINCOA
- Nelder–Mead method

68.3 External links

- Source code of COBYLA software
- A repository of Powell's software

Chapter 69

Coffman–Graham algorithm

In job shop scheduling and graph drawing, the **Coffman–Graham algorithm** is an algorithm, named after Edward G. Coffman, Jr. and Ronald Graham, for arranging the elements of a partially ordered set into a sequence of levels. The algorithm chooses an arrangement such that an element that comes after another in the order is assigned to a lower level, and such that each level has a number of elements that does not exceed a fixed width bound W . When $W = 2$, it uses the minimum possible number of distinct levels,^[1] and in general it uses at most $2 - 2/W$ times as many levels as necessary.^{[2][3]}

69.1 Problem statement and applications

In the version of the job shop scheduling problem solved by the Coffman–Graham algorithm, one is given a set of n jobs J_1, J_2, \dots, J_n , together with a system of precedence constraints $J_i < J_j$ requiring that job J_i be completed before job J_j begins. Each job is assumed to take unit time to complete. The scheduling task is to assign each of these jobs to time slots on a system of W identical processors, minimizing the **makespan** of the assignment (the time from the beginning of the first job until the completion of the final job). Abstractly, the precedence constraints define a partial order on the jobs, so the problem can be rephrased as one of assigning the elements of this partial order to levels (time slots) in such a way that each time slot has at most as many jobs as processors (at most W elements per level), respecting the precedence constraints. This application was the original motivation for Coffman and Graham to develop their algorithm.^{[1][4]}

In the layered graph drawing framework outlined by Sugiyama, Tagawa & Toda (1981)^[5] and implemented in systems such as Microsoft Automatic Graph Layout, the input is a directed graph, and a drawing of a graph is constructed in several stages:^{[6][7]}

1. A feedback arc set is chosen, and the edges of this set reversed, in order to convert the input into a directed acyclic graph.
2. The vertices of the graph are given integer y-

coordinates in such a way that, for each edge, the starting vertex of the edge has a higher coordinate than the ending vertex, with at most W vertices sharing the same y-coordinate.

3. Dummy vertices are introduced within each edge so that the subdivided edges all connect pairs of vertices that are in adjacent levels of the drawing.
4. Within each group of vertices with the same y-coordinate, the vertices are permuted in order to minimize the **number of crossings** in the resulting drawing, and the vertices are assigned x-coordinates consistently with this permutation.
5. The vertices and edges of the graph are drawn with the coordinates assigned to them.

In this framework, the y-coordinate assignment again involves grouping elements of a partially ordered set (the vertices of the graph, with the **reachability** ordering on the vertex set) into layers (sets of vertices with the same y-coordinate), which is the problem solved by the Coffman–Graham algorithm.^[6] Although there exist alternative approaches than the Coffman–Graham algorithm to the layering step, these alternatives in general are either not able to incorporate a bound on the maximum width of a level or rely on complex integer programming procedures.^[8]

More abstractly, both of these problems can be formalized as a problem in which the input consists of a partially ordered set and an integer W . The desired output is an assignment of integer level numbers to the elements of the partially ordered set such that, if $x < y$ is an ordered pair of related elements of the partial order, the number assigned to x is smaller than the number assigned to y , such that at most W elements are assigned the same number as each other, and minimizing the difference between the smallest and the largest assigned numbers.

69.2 The algorithm

The Coffman–Graham algorithm performs the following steps.^[6]

1. Represent the partial order by its **transitive reduction** or **covering relation**, a directed acyclic graph G that has an edge from x to y whenever $x < y$ and there does not exist any third element z of the partial order for which $x < z < y$. In the graph drawing applications of the Coffman–Graham algorithm, the resulting directed acyclic graph may not be the same as the graph being drawn, and in the scheduling applications it may not have an edge for every precedence constraint of the input: in both cases, the transitive reduction removes redundant edges that are not necessary for defining the partial order.
2. Construct a **topological ordering** of G in which the vertices are ordered **lexicographically** by the set of positions of their incoming neighbors. To do so, add the vertices one at a time to the ordering, at each step choosing a vertex v to add such that the incoming neighbors of v are all already part of the partial ordering, and such that the most recently added incoming neighbor of v is earlier than the most recently added incoming neighbor of any other vertex that could be added in place of v . If two vertices have the same most recently added incoming neighbor, the algorithm breaks the tie in favor of the one whose second most recently added incoming neighbor is earlier, etc.
3. Assign the vertices of G to levels in the reverse of the topological ordering constructed in the previous step. For each vertex v , add v to a level that is at least one step higher than the highest level of any outgoing neighbor of v , that does not already have W elements assigned to it, and that is as low as possible subject to these two constraints.

69.3 Analysis

As Coffman & Graham (1972) originally proved, their algorithm computes an optimal assignment for $W = 2$; that is, for scheduling problems with unit length jobs on two processors, or for layered graph drawing problems with at most two vertices per layer.^[1] A closely related algorithm also finds the optimal solution for scheduling of jobs with varying lengths, allowing pre-emption of scheduled jobs, on two processors.^[9] For $W > 2$, the Coffman–Graham algorithm uses a number of levels (or computes a schedule with a makespan) that is within a factor of $2 - 2/W$ of optimal.^{[2][3]} For instance, for $W = 3$, this means that it uses at most $4/3$ times as many levels as is optimal. When the partial order of precedence constraints is an interval order, or belongs to several related classes of partial orders, the Coffman–Graham algorithm finds a solution with the minimum number of levels regardless of its width bound.^[10]

As well as finding schedules with small makespan, the Coffman–Graham algorithm (modified from the presen-

tation here so that it topologically orders the **reverse graph** of G and places the vertices as early as possible rather than as late as possible) minimizes the **total flow time** of two-processor schedules, the sum of the completion times of the individual jobs. A related algorithm can be used to minimize the total flow time for a version of the problem in which preemption of jobs is allowed.^[11]

Coffman & Graham (1972) and Lenstra & Rinnooy Kan (1978)^[12] state the time complexity of the Coffman–Graham algorithm, on an n -element partial order, to be $O(n^2)$. However, this analysis omits the time for constructing the transitive reduction, which is not known to be possible within this bound. Sethi (1976) shows how to implement the topological ordering stage of the algorithm in linear time, based on the idea of **partition refinement**.^[13] Sethi also shows how to implement the level assignment stage of the algorithm efficiently by using a disjoint-set data structure. In particular, with a version of this structure published later by Gabow & Tarjan (1985), this stage also takes linear time.^[14]

69.4 References

- [1] Coffman, E. G., Jr.; Graham, R. L. (1972), “Optimal scheduling for two-processor systems”, *Acta Informatica* **1**: 200–213, doi:10.1007/bf00288685, MR 0334913.
- [2] Lam, Shui; Sethi, Ravi (1977), “Worst case analysis of two scheduling algorithms”, *SIAM Journal on Computing* **6** (3): 518–536, doi:10.1137/0206037, MR 0496614.
- [3] Braschi, Bertrand; Trystram, Denis (1994), “A new insight into the Coffman–Graham algorithm”, *SIAM Journal on Computing* **23** (3): 662–669, doi:10.1137/S0097539790181889, MR 1274650.
- [4] Leung, Joseph Y.-T. (2004), “Some basic scheduling algorithms”, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press, ISBN 978-1-58488-397-5.
- [5] Sugiyama, Kozo; Tagawa, Shôjirô; Toda, Mitsuhiro (1981), “Methods for visual understanding of hierarchical system structures”, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11 (2): 109–125, doi:10.1109/TSMC.1981.4308636, MR 0611436.
- [6] di Battista, Giuseppe; Eades, Peter; Tamassia, Roberto; Tollis, Ioannis G. (1999), “Chapter 9: Layered drawings of digraphs”, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, pp. 265–302.
- [7] Bastert, Oliver; Matuszewski, Christian (2001), “Layered drawings of digraphs”, in Kaufmann, Michael; Wagner, Dorothea, *Drawing Graphs: Methods and Models*, Lecture Notes in Computer Science **2025**, Springer-Verlag, pp. 87–120, doi:10.1007/3-540-44969-8_5. Bastert and Matuszewski also include a description of the Coffman–Graham algorithm; however, they omit the transitive reduction stage of the algorithm.

- [8] Healy, Patrick; Nikolov, Nikola S. (2002), “How to layer a directed acyclic graph”, *Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001, Revised Papers*, Lecture Notes in Computer Science **2265**, Springer-Verlag, pp. 16–30, doi:10.1007/3-540-45848-4_2, MR 1962416.
- [9] Muntz, R. R.; Coffman, E. G. (1969), “Optimal preemptive scheduling on two-processor systems”, *IEEE Transactions on Computers* **18**: 1014–1020, doi:10.1109/T-C.1969.222573.
- [10] Chardon, Marc; Moukrim, Aziz (2005), “The Coffman-Graham algorithm optimally solves UET task systems with overinterval orders”, *SIAM Journal on Discrete Mathematics* **19** (1): 109–121, doi:10.1137/S0895480101394999, MR 2178187.
- [11] Coffman, E. G., Jr.; Sethuraman, J.; Timkovsky, V. G. (2003), “Ideal preemptive schedules on two processors”, *Acta Informatica* **39** (8): 597–612, doi:10.1007/s00236-003-0119-6, MR 1996238.
- [12] Lenstra, J. K.; Rinnooy Kan, A. H. G. (1978), “Complexity of scheduling under precedence constraints”, *Operations Research* **26** (1): 22–35, doi:10.1287/opre.26.1.22, JSTOR 169889, MR 0462553.
- [13] Sethi, Ravi (1976), “Scheduling graphs on two processors”, *SIAM Journal on Computing* **5** (1): 73–82, doi:10.1137/0205005, MR 0398156.
- [14] Gabow, Harold N.; Tarjan, Robert Endre (1985), “A linear-time algorithm for a special case of disjoint set union”, *Journal of Computer and System Sciences* **30** (2): 209–221, doi:10.1016/0022-0000(85)90014-5, MR 801823.

Chapter 70

Column generation

Column generation or **delayed column generation** is an efficient algorithm for solving larger linear programs.

The overarching idea is that many linear programs are too large to consider all the variables explicitly. Since most of the variables will be non-basic and assume a value of zero in the optimal solution, only a subset of variables need to be considered in theory when solving the problem. Column generation leverages this idea to generate only the variables which have the potential to improve the objective function—that is, to find variables with negative reduced cost (assuming without loss of generality that the problem is a minimization problem).

The problem being solved is split into two problems: the master problem and the subproblem. The master problem is the original problem with only a subset of variables being considered. The subproblem is a new problem created to identify a new variable. The objective function of the subproblem is the reduced cost of the new variable with respect to the current dual variables, and the constraints require that the variable obey the naturally occurring constraints.

The process works as follows. The master problem is solved—from this solution, we are able to obtain dual prices for each of the constraints in the master problem. This information is then utilized in the objective function of the subproblem. The subproblem is solved. If the objective value of the subproblem is negative, a variable with negative reduced cost has been identified. This variable is then added to the master problem, and the master problem is re-solved. Re-solving the master problem will generate a new set of dual values, and the process is repeated until no negative reduced cost variables are identified. The subproblem returns a solution with non-negative reduced cost, we can conclude that the solution to the master problem is optimal.

In many cases, this allows large linear programs that had been previously considered intractable to be solved. The classical example of a problem where this is successfully used is the **cutting stock problem**. One particular technique in linear programming which uses this kind of approach is the **Dantzig–Wolfe decomposition** algorithm. Additionally, column generation has been applied to many problems such as crew scheduling, vehicle rout-

ing, and the capacitated p-median problem.

Chapter 71

Constructive cooperative coevolution

The **constructive cooperative coevolutionary algorithm** (also called C³) is an global optimisation algorithm in artificial intelligence based on the multi-start architecture of the greedy randomized adaptive search procedure (GRASP).^{[1][2]} It incorporates the existing cooperative coevolutionary algorithm (CC).^[3] The considered problem is decomposed into subproblems. These subproblems are optimised separately while exchanging information in order to solve the complete problem. An optimisation algorithm, usually but not necessarily an evolutionary algorithm, is embedded in C³ for optimising those subproblems. The nature of the embedded optimisation algorithm determines whether C³'s behaviour is deterministic or stochastic.

The C³ optimisation algorithm was originally designed for simulation-based optimisation^{[4][5]} but it can be used for global optimisation problems in general.^[5] Its strength over other optimisation algorithms, specifically cooperative coevolution, is that it is better able to handle non-separable optimisation problems.^[4]

71.1 Algorithm

As shown in the psuedo code below, an iteration of C³ exists of two phases. In Phase I, the constructive phase, a feasible solution for the entire problem is constructed in a stepwise manner. Considering a different subproblem in each step. After the final step, all subproblems are considered and a solution for the complete problem has been constructed. This constructed solution is then used as initial solution in Phase II, the local improvement phase. The CC algorithm is employed to further optimise the constructed solution. A cycle of Phase II includes optimising the subproblems separately while keeping the parameters of the other subproblems fixed to a central blackboard solution. When this is done for each subproblem, the found solution are combined during a “collaboration” step, and the best one among the produced combinations becomes the blackboard solution for the next cycle. In the next cycle, the same is repeated. Phase II, and thereby the current iteration, are terminated when the search of the CC algorithm stagnates and no significantly better solutions are being found. Then, the next iteration

is started. At the start of the next iteration, a new feasible solution is constructed, utilising solutions that were found during the Phase I of the previous iteraton(s). This constructed solution is then used as initial solution in Phase II in the same way as in the first iteration. This is repeated until one of the termination criteria for the optimisation is reached, e.g. maximum number of evaluations.

```
{Sphase1} ← ∅ WHILE termination criteria not satisfied DO IF {Sphase1} == ∅ THEN {Sphase1} ← SubOpt(∅, 1) END IF WHILE pphase1 not completely constructed DO pphase1 ← GetBest({Sphase1}) {Sphase1} ← SubOpt(pphase1, inext subproblem) END WHILE pphase2 ← GetBest({Sphase1}) WHILE not stagnate DO {Sphase2} ← ∅ FOR each subproblem i DO {Sphase2} ← SubOpt(pphase2, i) END FOR {Sphase2} ← Collab({Sphase2}) pphase2 ← GetBest({Sphase2}) END WHILE END WHILE
```

71.2 Applications

The constructive cooperative coevolution algorithm has been applied to different types of problems, e.g. a set of standard benchmark functions,^[4] optimisation of sheet metal press lines,^{[4][5][6]} and interacting production stations.^[5] The C³ algorithm has been embedded with, amongst others, the differential evolution algorithm^[7] and the particle swarm optimiser^[8] for the subproblem optimisations.

71.3 See also

- Cooperative coevolution
- Metaheuristic
- Stochastic search
- Differential evolution
- Swarm intelligence
- Genetic algorithms
- Hyper-heuristics

71.4 References

- [1] T.A. Feo and M.G.C. Resende (1989) A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.
- [2] T.A. Feo and M.G.C. Resende (1995) Greedy randomized adaptive search procedures. *J. of Global Optimization*, 6:109–133, 1995.
- [3] M. A. Potter and K. A. D. Jong, “A cooperative coevolutionary approach to function optimization,” in PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature. London, UK: Springer-Verlag, 1994, pp. 249–257.
- [4] Glorieux E., Danielsson F., Svensson B., Lennartson B., Optimisation of interacting production stations using a Constructive Cooperative Coevolutionary approach, 2014 IEEE International Conference on Automation Science and Engineering (CASE), pp.322-327, August 2014, Taipei, Taiwan
- [5] Glorieux E., Svensson B., Danielsson F., Lennartson B., A Constructive Cooperative Coevolutionary Algorithm Applied to Press Line Optimisation. Proceedings of the 24th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM), pp.909-917, May 2014, San Antonio, Texas, USA
- [6] Glorieux E., Svensson B., Danielsson F., Lennartson B., Optimised Control of Sheet Metal Press Lines, Proceedings of the 6th International Swedish Production Symposium 2014, September 2014, Gothenburg, Sweden
- [7] Storn, Rainer, and Kenneth Price. “Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces.” *Journal of global optimization* 11.4 (1997): 341-359.
- [8] Eberhart, Russ C., and James Kennedy. “A new optimizer using particle swarm theory.” Proceedings of the sixth international symposium on micro machine and human science. Vol. 1. 1995.
-

Chapter 72

Crew scheduling

Crew scheduling is the process of assigning crews to operate transportation systems, such as rail lines or aircraft.

72.1 Complex

Most transportation systems use software to manage the crew scheduling process. Crew scheduling becomes more and more complex as you add variables to the problem. These variables can be as simple as 1 location, 1 skill requirement, 1 shift of work and 1 set roster of people. In the Transportation industries, such as Rail or mainly Air Travel, these variables become very complex. In Air Travel for instance, there are numerous rules or “constraints” that are introduced. These mainly deal with legalities relating to work shifts and time, and a crew members qualifications for working on a particular aircraft. Add numerous locations to the equation and Collective Bargaining and Federal labor laws and these become new considerations for the problem solving method. Fuel is also a major consideration as aircraft and other vehicles require a lot of costly fuel to operate. Finding the most efficient route and staffing it with properly qualified personnel is a critical financial consideration. The same applies to rail travel.

The problem is computationally difficult and there are competing mathematical methods of solving the problem. Although not easy to describe in one sentence, the goal is the essentially same for any method of attacking the problem:

Within a set of constraints and rules, move a set roster of people with certain qualifications, from place to place with the least amount of personnel and aircraft or vehicles in the least amount of time.

Lowest cost has traditionally been the major driver for any crew scheduling solution.

72.2 4 Parts

Although not a “rule”, We can describe at least 4 parts of the equation that are ingested by the computational process:

- People and their qualifications and abilities.
- Aircraft or vehicles and their “People” qualification requirements and their cost to operate over distance.
- Locations and the time and distance between each location.
- Work rules for the personnel, including Shift hours and seniority.

In crew scheduling the rules and constraints are typically a combination of:

- government regulations concerning flight time, duty time and required rest, designed to promote **aviation safety** and limit crew fatigue,
- crew bid requests, vacations,
- labor agreements
- aircraft maintenance schedules
- crew member qualification and licensing
- other constraints related to training
- pairing experienced crew members with more junior crew members
- returning crew to their base at the end of their trip (called **deadheading**)

All of these issues need to be addressed in order to create a satisfactory solution for personnel and management of the organization. For the Crew member in a seniority based system schedules are decided largely on workplace seniority. Those at the top of a seniority list are allowed some choices. As assignments are made and the remaining roster of personnel becomes fewer, managements’ systems start to assign the remaining trips based on a weighting of the 4 previously mentioned variables, without any input from personnel.

This does not allow the personnel to have any choice or voice in the schedules they receive. This lack of scheduling awareness until the end of each scheduling period is a major workforce issue and an employee morale problem,

often creating a tenuous situation especially where a collective bargaining agreement is in place, and particularly at negotiation time.

72.3 Disruptions

Additional unplanned disruptions in schedules due to weather and air traffic control delays can disrupt schedules, so crew scheduling software remains an area for ongoing research.^[1]

72.4 Related topics (systems)

- Preferential bidding system
- Fatigue (safety)
- Automated planning and scheduling

72.5 Related topics (algorithms and software)

- Linear programming
- Column generation
- Tabu search
- Fatigue Avoidance Scheduling Tool

72.6 References

[1] <http://www.engr.pitt.edu/~{ }schaefer/Papers/UncertainCrewSched.pdf> "Airline crew Scheduling under Uncertainty"

Chapter 73

Cross-entropy method

The **cross-entropy (CE) method** attributed to Reuven Rubinstein is a general Monte Carlo approach to combinatorial and continuous multi-extremal optimization and importance sampling. The method originated from the field of *rare event simulation*, where very small probabilities need to be accurately estimated, for example in network reliability analysis, queueing models, or performance analysis of telecommunication systems. The CE method can be applied to static and noisy combinatorial optimization problems such as the traveling salesman problem, the quadratic assignment problem, DNA sequence alignment, the max-cut problem and the buffer allocation problem, as well as continuous global optimization problems with many local extrema.

In a nutshell the CE method consists of two phases:

1. Generate a random data sample (trajectories, vectors, etc.) according to a specified mechanism.
2. Update the parameters of the random mechanism based on the data to produce a “better” sample in the next iteration. This step involves minimizing the *cross-entropy* or Kullback–Leibler divergence.

73.1 Estimation via importance sampling

Consider the general problem of estimating the quantity $\ell = \mathbb{E}_{\mathbf{u}}[H(\mathbf{X})] = \int H(\mathbf{x}) f(\mathbf{x}; \mathbf{u}) d\mathbf{x}$, where H is some *performance function* and $f(\mathbf{x}; \mathbf{u})$ is a member of some parametric family of distributions. Using importance sampling this quantity can be estimated as $\hat{\ell} = \frac{1}{N} \sum_{i=1}^N H(\mathbf{X}_i) \frac{f(\mathbf{X}_i; \mathbf{u})}{g(\mathbf{X}_i)}$, where $\mathbf{X}_1, \dots, \mathbf{X}_N$ is a random sample from g . For positive H , the theoretically optimal importance sampling density (pdf) is given by $g^*(\mathbf{x}) = H(\mathbf{x}) f(\mathbf{x}; \mathbf{u}) / \ell$. This, however, depends on the unknown ℓ . The CE method aims to approximate the optimal PDF by adaptively selecting members of the parametric family that are closest (in the Kullback–Leibler sense) to the optimal PDF g^* .

73.2 Generic CE algorithm

1. Choose initial parameter vector $\mathbf{v}^{(0)}$; set $t = 1$.
2. Generate a random sample $\mathbf{X}_1, \dots, \mathbf{X}_N$ from $f(\cdot; \mathbf{v}^{(t-1)})$
1. Solve for $\mathbf{v}^{(t)}$, where $\mathbf{v}^{(t)} = \operatorname{argmax}_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^N H(\mathbf{X}_i) \frac{f(\mathbf{X}_i; \mathbf{u})}{f(\mathbf{X}_i; \mathbf{v}^{(t-1)})} \log f(\mathbf{X}_i; \mathbf{v})$
2. If convergence is reached then **stop**; otherwise, increase t by 1 and reiterate from step 2.

In several cases, the solution to step 3 can be found *analytically*. Situations in which this occurs are

- When f belongs to the natural exponential family
- When f is discrete with finite support
- When $H(\mathbf{X}) = I_{\{\mathbf{x} \in A\}}$ and $f(\mathbf{X}_i; \mathbf{u}) = f(\mathbf{X}_i; \mathbf{v}^{(t-1)})$, then $\mathbf{v}^{(t)}$ corresponds to the maximum likelihood estimator based on those $\mathbf{X}_k \in A$.

73.3 Continuous optimization—example

The same CE algorithm can be used for optimization, rather than estimation. Suppose the problem is to maximize some function $S(x)$, for example, $S(x) = e^{-(x-2)^2} + 0.8e^{-(x+2)^2}$. To apply CE, one considers first the *associated stochastic problem* of estimating $\mathbb{P}_{\theta}(S(X) \geq \gamma)$ for a given level γ , and parametric family $\{f(\cdot; \theta)\}$, for example the 1-dimensional Gaussian distribution, parameterized by its mean μ_t and variance σ_t^2 (so $\theta = (\mu, \sigma^2)$ here). Hence, for a given γ , the goal is to find θ so that $D_{\text{KL}}(I_{\{S(x) \geq \gamma\}} \| f_{\theta})$ is minimized. This is done by solving the sample version (stochastic counterpart) of the KL divergence minimization problem, as in step 3 above. It turns out that parameters that minimize the stochastic counterpart for this choice of target distribution and parametric family are the sample

mean and sample variance corresponding to the *elite samples*, which are those samples that have objective function value $\geq \gamma$. The worst of the elite samples is then used as the level parameter for the next iteration. This yields the following randomized algorithm that happens to coincide with the so-called Estimation of Multivariate Normal Algorithm (EMNA), an estimation of distribution algorithm.

73.3.1 Pseudo-code

```

1. mu:=-6; sigma2:=100; t:=0; maxits=100; // Initialize
parameters
2. N:=100; Ne:=10; // 3. while t < maxits
and sigma2 > epsilon // While maxits not exceeded and
not converged
4. X = SampleGaussian(mu,sigma2,N); // Obtain N samples from current sampling distribution
5. S = exp(-(X-2)^2) + 0.8 exp(-(X+2)^2); // Evaluate ob-
jective function at sampled points
6. X = sort(X,S); // Sort X by objective function values (in descending order)
7. mu = mean(X(1:Ne)); sigma2=var(X(1:Ne)); // Up-
date parameters of sampling distribution
8. t = t+1; // Increment iteration counter
9. return mu // Return mean
of final sampling distribution as solution

```

- Rubinstein, R.Y., Kroese, D.P. (2004). *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*, Springer-Verlag, New York.

73.7 External links

- Homepage for the CE method

73.4 Related methods

- Simulated annealing
- Genetic algorithms
- Harmony search
- Estimation of distribution algorithm
- Tabu search

73.5 See also

- Cross entropy
- Kullback–Leibler divergence
- Randomized algorithm
- Importance sampling

73.6 References

- De Boer, P-T., Kroese, D.P, Mannor, S. and Rubinstei, R.Y. (2005). A Tutorial on the Cross-Entropy Method. *Annals of Operations Research*, **134** (1), 19–67.
- Rubinstein, R.Y. (1997). Optimization of Computer simulation Models with Rare Events, *European Journal of Operations Research*, **99**, 89–112.

Chapter 74

Cuckoo search

This article is about the search algorithm. For the hashing algorithm, see [cuckoo hashing](#).

Cuckoo search (CS) is an optimization algorithm developed by Xin-she Yang and Suash Deb in 2009.^{[1][2]} It was inspired by the [obligate brood parasitism](#) of some [cuckoo species](#) by laying their eggs in the nests of other host birds (of other species). Some host birds can engage direct conflict with the intruding cuckoos. For example, if a host bird discovers the eggs are not their own, it will either throw these alien eggs away or simply abandon its nest and build a new nest elsewhere. Some cuckoo species such as the [New World](#) brood-parasitic *Tapera* have evolved in such a way that female parasitic cuckoos are often very specialized in the mimicry in colors and pattern of the eggs of a few chosen host species^[3]

Cuckoo search idealized such breeding behavior, and thus can be applied for various optimization problems. It seems that it can outperform other metaheuristic algorithms in applications.^[4]

Cuckoo search (CS) uses the following representations:

Each egg in a nest represents a solution, and a cuckoo egg represents a new solution. The aim is to use the new and potentially better solutions (cuckoos) to replace a not-so-good solution in the nests. In the simplest form, each nest has one egg. The algorithm can be extended to more complicated cases in which each nest has multiple eggs representing a set of solutions.

CS is based on three idealized rules:

1. Each cuckoo lays one egg at a time, and dumps its egg in a randomly chosen nest;
2. The best nests with high quality of eggs will carry over to the next generation;
3. The number of available hosts nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with a probability $p_a \in (0, 1)$. Discovering operate on some set of worst nests, and discovered solutions dumped from farther calculations.

In addition, Yang and Deb discovered that the random-walk style search is better performed by Lévy flights

rather than simple random walk.

The pseudo-code can be summarized as:

Objective function: $f(\mathbf{x})$, $\mathbf{x} = (x_1, x_2, \dots, x_d)$; Generate an initial population of n host nests; While ($t < \text{MaxGeneration}$) or (stop criterion) Get a cuckoo randomly (say, i) and replace its solution by performing Lévy flights; Evaluate its quality/fitness F_i [For maximization, $F_i \propto f(\mathbf{x}_i)$]; Choose a nest among n (say, j) randomly; if ($F_i > F_j$), Replace j by the new solution; end if A fraction (p_a) of the worse nests are abandoned and new ones are built; Keep the best solutions/nests; Rank the solutions/nests and find the current best; Pass the current best solutions to the next generation; end while

An important advantage of this algorithm is its simplicity. In fact, comparing with other population- or agent-based metaheuristic algorithms such as [particle swarm optimization](#) and [harmony search](#), there is essentially only a single parameter p_a in CS (apart from the population size n). Therefore, it is very easy to implement.

74.1 Random walks and the step size

An important issue is the applications of Lévy flights and random walks in the generic equation for generating new solutions

$$\mathbf{x}_{t+1} = \mathbf{x}_t + sE_t,$$

where E_t is drawn from a standard normal distribution with zero mean and unity standard deviation for random walks, or drawn from Lévy distribution for Lévy flights. Obviously, the random walks can also be linked with the similarity between a cuckoo's egg and the host's egg which can be tricky in implementation. Here the step size s determines how far a random walker can go for a fixed number of iterations. The generation of Lévy step size is often tricky, and a comparison of three algorithms (including Mantegna's^[5]) was performed by Leccardi^[6] who found an implementation of Chambers et al's approach^[7] to be the most computationally efficient due to the low number of random numbers required.

If s is too large, then the new solution generated will be too far away from the old solution (or even jump out side of the bounds). Then, such a move is unlikely to be accepted. If s is too small, the change is too small to be significant, and consequently such search is not efficient. So a proper step size is important to maintain the search as efficient as possible.

As an example, for simple isotropic random walks, we know that the average distance r traveled in the d -dimension space is

$$r^2 = 2dDt,$$

where $D = s^2/2\tau$ is the effective diffusion coefficient. Here s is the step size or distance traveled at each jump, and τ is the time taken for each jump. The above equation implies that^[8]

$$s^2 = \frac{\tau r^2}{t d}.$$

For a typical length scale L of a dimension of interest, the local search is typically limited in a region of $r = L/10$. For $\tau = 1$ and $t=100$ to 1000 , we have $s \approx 0.01L$ for $d=1$, and $s \approx 0.001L$ for $d=10$. Therefore, we can use $s/L=0.001$ to 0.01 for most problems. Though the exact derivation may require detailed analysis of the behaviour of Lévy flights.^[9]

Algorithm and convergence analysis will be fruitful, because there are many open problems related to metaheuristics^[10]

74.2 Implementation

The pseudo code was given in a sequential form, but Yang and Deb provides a vectorized implementation which is very efficient.^[11] In the real world, if a cuckoo's egg is very similar to a host's eggs, then this cuckoo's egg is less likely to be discovered, thus the fitness should be related to the difference in solutions. Therefore, it is a good idea to do a random walk in a biased way with some random step sizes. For Matlab implementation, this biased random walk can partly be achieved by

```
stepsize=rand*(nest(randperm(n),:)-  
nest(randperm(n),:));  
new_nest=nest+stepsize.*K;
```

where $K=\text{rand}(\text{size}(nest))>pa$ and pa is the discovery rate.

A standard CS matlab can be found here <http://www.mathworks.com/matlabcentral/fileexchange/29809-cuckoo-search-CS-algorithm>.^[12] However, there is some controversy on whether or not this implementation truly reflects the originally published cuckoo search

algorithm. The biased walk described above closely resembles another optimization algorithm, differential evolution. It was shown that this biased walk has a larger impact on the performance of the algorithm than the cuckoo search algorithm itself.^[13]

An object-oriented software implementation of cuckoo search was provided by Bacanin^[14] On the other hand, a modified cuckoo search algorithm is also implemented for unconstrained optimization problems.^[15]

An open source implementation of Modified Cuckoo Search can be found here <https://code.google.com/p/modifed-cs/>

74.3 Modified cuckoo search

A modification of the standard Cuckoo Search was made by Walton et al.^[16] with the aim to speed up convergence. The modification involves the additional step of information exchange between the top eggs. It was shown that Modified Cuckoo Search (MCS) outperforms the standard cuckoo search and other algorithms, in terms of convergence rates, when applied to a series of standard optimization benchmark objective functions.

Subsequently, the modified cuckoo search has been applied to optimize unstructured mesh which reduces computational cost significantly.^{[17][18]} In addition, another interesting improvement to cuckoo search is the so-called quantum-inspired cuckoo search with convincing results^[19]

74.4 Multiobjective cuckoo search (MOCS)

A multiobjective cuckoo search (MOCS) method has been formulated to deal with multi-criteria optimization problems.^[20] This approach uses random weights to combine multiple objectives to a single objective. As the weights vary randomly, Pareto fronts can be found so the points can be distributed diversely over the fronts.

74.5 Hybridization

Though cuckoo search is a swarm-intelligence-based algorithm, it can be still hybridized with other swarm-based algorithms such as PSO. For example, a hybrid CS-PSO algorithm seems to remedy the defect of PSO.^[21]

74.6 Applications

The applications of Cuckoo Search into engineering optimization problems^[22] have shown its promising effi-

ciency. For example, for both spring design and welded beam design problems, CS obtained better solutions than existing solutions in literature. A promising discrete cuckoo search algorithm is recently proposed to solve nurse scheduling problem.^[23] An efficient computation approach based on cuckoo search has been proposed for data fusion in wireless sensor networks.^{[24][25]} Cuckoo search is adapted to solve NP-hard combinatorial optimization problems like *travelling salesman problem*.^[26] A new quantum-inspired cuckoo search was developed to solve Knapsack problems, which shows its effectiveness.^[27] Cuckoo search can also be used to efficiently generate independent test paths for structural software testing^[28] and test data generation.^{[29][30]} Cuckoo search algorithm has been used for a nanoelectronic technology based operation-amplifier (OP-AMP) integrated circuit optimization which converged to optimal solutions very fast accurately.^[31]

A conceptual comparison of the cuckoo search with Particle swarm optimization, Differential evolution and Artificial bee colony algorithm suggest that Cuckoo search and Differential evolution algorithms provide more robust results than Particle swarm optimization and Artificial bee colony algorithm.^[32] An extensive detailed study of various structural optimization problems suggests that cuckoo search obtains better results than other algorithms.^[33] In addition, a new software testing approach has been developed based on cuckoo search.^[34] In addition, cuckoo search is particularly suitable for large scale problems, as shown in a recent study.^[35] Cuckoo search has been applied to train neural networks with improved performance.^[36] Furthermore, CS is successfully applied to train spiking neural models.^[37] Cuckoo search has also been used to optimize web service composition process and planning graphs.^[38]

Cuckoo search is a reliable approach for embedded system design^[39] and design optimization^[40] including optimum design of steel frames.^[41]

More recent studies suggest that cuckoo search can outperform other algorithms in milling applications,^[42] manufacturing scheduling,^[43] and others.^[44] An interesting application of cuckoo search is to solve boundary value problems.^[45] More recently, cuckoo search algorithm is used for optimal parameter estimation of nonlinear Muskingum flood routing model.^[46]

74.7 References

- [1] X.-S. Yang; S. Deb (December 2009). *Cuckoo search via Lévy flights*. World Congress on Nature & Biologically Inspired Computing (NaBIC 2009). IEEE Publications. pp. 210–214. arXiv:1003.1594v1.
- [2] Inderscience (27 May 2010). “Cuckoo designs spring”. Alphagalileo.org. Retrieved 2010-05-27.
- [3] R. B. Payne, M. D. Sorenson, and K. Klitz, The Cuckoos, Oxford University Press, (2005).
- [4] [http://www.scientificcomputing.com/
news-DA-Novel-Cuckoo-Search-Algorithm-Beats-Particle-Swarm-Optimization.aspx](http://www.scientificcomputing.com/news-DA-Novel-Cuckoo-Search-Algorithm-Beats-Particle-Swarm-Optimization.aspx)
- [5] R. N. Mantegna, Fast, accurate algorithm for numerical simulation of Lévy stable stochastic processes, Physical Review E, Vol.49, 4677-4683 (1994).
- [6] M. Leccardi, Comparison of three algorithms for Levy noise generation, Proceedings of fifth EUROMECH nonlinear dynamics conference (2005).
- [7] J. M. Chambers, C. L. Mallows, and B. W. Stuck, A method for simulating stable random variables, Journal of the American Statistical Association, Vol.71, 340-344 (1976)
- [8] X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms*, 2nd Edition, Luniver Press, (2010).
- [9] M. Gutowski, Lévy flights as an underlying mechanism for global optimization algorithms, ArXiv Mathematical Physics e-Prints, June, (2001).
- [10] X. S. Yang, Metaheuristic optimization: algorithm analysis and open problems, in: Experimental Algorithms (SEA2011), Eds (P. M. Pardalos and S. Rebennack), LNCS 6630, pp.21-32 (2011).
- [11] X.-S. Yang and S. Deb, “Engineering optimisation by cuckoo search”, Int. J. Mathematical Modelling and Numerical Optimisation”, Vol. 1, No. 4, 330-343 (2010). <http://arxiv.org/abs/1005.2908>
- [12] Matlab demo code
- [13] S. Walton; O. Hassan, M.R. Brown and K. Morgan (24 May 2013). “Comment on Paper by Bhargava, Faatén and Bonilla-Petriciolet”. *Fluid Phase Equilibria*. doi:10.1016/j.fluid.2013.05.011.
- [14] N. Bacanin, An object-oriented software implementation of a novel cuckoo search algorithm, Proc. of the 5th European Conference on European Computing Conference (ECC'11), pp. 245-250 (2011).
- [15] M. Tuba, M. Subotic, and N. Stanarevic, Modified cuckoo search algorithm for unconstrained optimization problems, Proc. of the 5th European Conference on European Computing Conference (ECC'11), pp. 263-268 (2011).
- [16] S. Walton; O. Hassan, K. Morgan and M.R. Brown (30 June 2011). “Modified cuckoo search: A new gradient free optimisation algorithm”. *Chaos, Solitons & Fractals*. doi:10.1016/j.chaos.2011.06.004.
- [17] S. Walton, O. Hassan, K. Morgan, Using proper orthogonal decomposition to reduce the order of optimization problems, in: Proc. 16th Int. Conf. on Finite Elements in Flow Problems (Eds. Wall W.A. and Givemeier V.), Munich, p.90 (2011).
- [18] Walton, S., Hassan, O. and Morgan, K. (2012), Reduced order mesh optimisation using proper orthogonal decomposition and a modified cuckoo search. *Int. J. Numer. Meth. Engng.*. doi: 10.1002/nme.4400

- [19] A. Layeb, A novel quantum inspired cuckoo search for knapsack problems, *Int. J. Bio-Inspired Computation*, Vol. 3, 297-305 (2011).
- [20] X. S. Yang and S. Deb, Multiobjective cuckoo search for design optimization, *Computers and Operations Research*, October (2011). doi:10.1016/j.cor.2011.09.026
- [21] F. Wang, L. Lou, X. He, Y. Wang, Hybrid optimization algorithm of PSO and Cuckoo Search, in: Proc. of 2nd Int. Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC'11), pp. 1172-1175 (2011).
- [22] Sean Walton, Oubay Hassan, Kenneth Morgan, Selected Engineering Applications of Gradient Free Optimisation Using Cuckoo Search and Proper Orthogonal Decomposition, *Archives of Computational Methods in Engineering* June 2013, Volume 20, Issue 2, pp 123-154, <http://link.springer.com/article/10.1007/s11831-013-9083-7>
- [23] Tein L. H. and Ramli R., Recent advancements of nurse scheduling models and a potential path, in: Proc. 6th IMT-GT Conference on Mathematics, Statistics and its Applications (ICMSA 2010), pp. 395-409 (2010). http://research.utar.edu.my/CMS/ICMSA2010/ICMSA2010_Proceedings/files/statistics/ST-Lim.pdf
- [24] M. Dhivya, M. Sundarambal, L. N. Anand, Energy Efficient Computation of Data Fusion in Wireless Sensor Networks Using Cuckoo Based Particle Approach (CBPA), *Int. J. of Communications, Network and System Sciences*, Vol. 4, No. 4, 249-255 (2011).
- [25] M. Dhivya and M. Sundarambal, Cuckoo search for data gathering in wireless sensor networks, *Int. J. Mobile Communications*, Vol. 9, pp. 642-656 (2011).
- [26] A. Ouazarab, B. Ahiod, and X.-S. Yang, Discrete cuckoo search algorithm for the travelling salesman problem, *Neural Computing and Applications*, (2013). doi:10.1007/s00521-013-1402-2.
- [27] A. Layeb, A novel quantum-inspired cuckoo search for Knapsack problems, *Int. J. Bio-inspired Computation*, Vol. 4, (2011).
- [28] P. R. Srivastava, M. Chis, S. Deb and X. S. Yang, An efficient optimization algorithm for structural software testing, *Int. J. Artificial Intelligence*, Vol. 9 (S12), 68-77(2012)
- [29] K. Perumal, J. M. Ungati, G. Kumar, N. Jain, R. Gaurav and P. R. Srivastava, Test data generation: a hybrid approach using cuckoo and tabu search, *Swarm, Evolutionary, and Memetic Computing (SEMCCO2011)*, Lecture Notes in Computer Sciences, Vol. 7077, 46-54 (2011)
- [30] Jeya Mala Dharmalingam, K. Sabari Nathan, S. Balamurugan: A hybrid test optimization framework using memetic algorithm with cuckoo flocking based search approach. *SBST 2014 Proceedings of the 7th International Workshop on Search-Based Software Testing*, 37-38, ACM, doi:10.1145/2593833.2593843
- [31] G. Zheng, S. P. Mohanty, and E. Kougnos, "Metamodel-Assisted Fast and Accurate Optimization of an OP-AMP for Biomedical Applications", in *Proceedings of the 11th IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 273--278, 2012.
- [32] P. Civicioglu and E. Besdok, A conception comparison of the cuckoo search, particle swarm optimization, differential evolution and artificial bee colony algorithms, *Artificial Intelligence Review*, doi:10.1007/s10462-011-92760, 6 July (2011).
- [33] A. H. Gandomi, X. S. Yang, A. H. Alavi, Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems, *Engineering with Computers*, Vol. 27, July (2011).
- [34] K. Choudhary and G. N. Purohit, A new testing approach using cuckoo search to achieve multi-objective genetic algorithm, *J. of Computing*, Vol. 3, No. 4, 117-119 (2011).
- [35] E. R. Speed, Evolving a Mario agent using cuckoo search and softmax heuristics, *Games Innovations Conference (ICE-GIC)*, pp. 1-7 (2010). doi:10.1109/ICEGIC.2010.5716893
- [36] E. Valian, S. Mohanna and S. Tavakoli, Improved cuckoo search algorithm for feedforward neural network training, *Int. J. Artificial Intelligence and Applications*, Vol. 2, No. 3, 36-43(2011).
- [37] R. A. Vazquez, Training spiking neural models using cuckoo search algorithm, *2011 IEEE Congress on Evolutionary Computation (CEC'11)*, pp.679-686 (2011).
- [38] V. R. Chifu, C. B. Pop, I. Salomie, D> S. Suia and A. N. Niculici, Optimizing the semantic web service composition process using cuckoo search, in: *Intelligent Distributed Computing V, Studies in Computational Intelligence*, Vol. 382, pp. 93-102 (2012).
- [39] A. Kumar and S. Chakarverty, Design optimization for reliable embedded system using Cuckoo Search,in: Proc. of 3rd Int. Conference on Electronics Computer Technology (ICECT2011), pp. 564-268 (2011).
- [40] A. Kumar and S. Chakarverty, Design optimization using genetic algorithm and Cuckoo Search, in: Proc. of IEEE Int. Conference on Electro/Information Technology (EIT), pp. 1-5 (2011).
- [41] A. Kaveh, T. Bakhshpoori, Optimum design of steel frames using cuckoo search algorithm with Lévy flights, *Structural Design of Tall and Special Buildings*, Vol. 21, online first 28 Nov 2011.<http://onlinelibrary.wiley.com/doi/10.1002/tal.754/abstract>
- [42] A. R. Yildiz, Cuckoo search algorithm for the selection of optimal machine parameters in milling operations, *Int. J. Adv. Manuf. Technol.*, (2012). doi:10.1007/s00170-012-4013-7
- [43] S. Burnwal, S. Deb, Scheduling optimization of flexible manufacturing system using cuckoo search-based approach, *Int. J. Adv Manuf Technol*, (2012).

- [44] H. Q. Zheng and Y. Zhou, A novel cuckoo search optimization algorithm based on Gauss distribution, *J. Computational Information Systems*, Vol. 8, 4193-4200 (2012).
 - [45] A. Noghrehabadi, M. Ghalambaz and A. Vosough, A hybrid power series -- Cuckoo search optimization algorithm to electrostatic deflection of micro fixed-fixed actuators, *Int. J. Multidisciplinary Science and Engineering*, Vol. 2, No. 4,22-26 (2011).
 - [46] H. Karahan, G. Gurarslan, Z.W. Geem (2014): A new nonlinear Muskingum flood routing model incorporating lateral flow, *Engineering Optimization*, DOI: 10.1080/0305215X.2014.918115
47. Sivakumar, L & Kotteeswaran, R 2014, 'Soft computing based partial-retuning of decentralised PI Controller of nonlinear multivariable process', *Advances in Intelligent Systems and Computing (AISC)*, Springer International Publishing, Switzerland, Vol.248, pp. 117–124.

Chapter 75

Derivation of the conjugate gradient method

In numerical linear algebra, the conjugate gradient method is an iterative method for numerically solving the linear system

$$Ax = b$$

where A is symmetric positive-definite. The conjugate gradient method can be derived from several different perspectives, including specialization of the conjugate direction method for optimization, and variation of the Arnoldi/Lanczos iteration for eigenvalue problems.

The intent of this article is to document the important steps in these derivations.

75.1 Derivation from the conjugate direction method

The conjugate gradient method can be seen as a special case of the conjugate direction method applied to minimization of the quadratic function

$$f(x) = x^T Ax - 2b^T x.$$

75.1.1 The conjugate direction method

In the conjugate direction method for minimizing

$$f(x) = x^T Ax - 2b^T x.$$

one starts with an initial guess x_0 and the corresponding residual $r_0 = b - Ax_0$, and computes the iterate and residual by the formulae

$$\alpha_i = \frac{p_i^T r_i}{p_i^T A p_i},$$

$$x_{i+1} = x_i + \alpha_i p_i,$$

$$r_{i+1} = r_i - \alpha_i A p_i$$

where p_0, p_1, p_2, \dots are a series of mutually conjugate directions, i.e.,

$$p_i^T A p_j = 0$$

for any $i \neq j$.

The conjugate direction method is imprecise in the sense that no formulae are given for selection of the directions p_0, p_1, p_2, \dots . Specific choices lead to various methods including the conjugate gradient method and Gaussian elimination.

75.2 Derivation from the Arnoldi/Lanczos iteration

Further information: Arnoldi iteration and Lanczos iteration

The conjugate gradient method can also be seen as a variant of the Arnoldi/Lanczos iteration applied to solving linear systems.

75.2.1 The general Arnoldi method

In the Arnoldi iteration, one starts with a vector r_0 and gradually builds an orthonormal basis $\{v_1, v_2, v_3, \dots\}$ of the Krylov subspace

$$\mathcal{K}(A, r_0) = \{r_0, Ar_0, A^2r_0, \dots\}$$

by defining $v_i = w_i / \|w_i\|_2$ where

$$w_i = \begin{cases} r_0 & \text{if } i = 1, \\ A v_{i-1} - \sum_{j=1}^{i-1} (v_j^T A v_{i-1}) v_j & \text{if } i > 1. \end{cases}$$

In other words, for $i > 1$, v_i is found by Gram-Schmidt orthogonalizing $A v_{i-1}$ against $\{v_1, v_2, \dots, v_{i-1}\}$ followed by normalization.

Put in matrix form, the iteration is captured by the equation

$$\mathbf{A}\mathbf{V}_i = \mathbf{V}_{i+1}\tilde{\mathbf{H}}_i$$

where

$$\mathbf{V}_i = [\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_i],$$

$$\tilde{\mathbf{H}}_i = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \cdots & h_{1,i} \\ h_{21} & h_{22} & h_{23} & \cdots & h_{2,i} \\ h_{31} & h_{32} & h_{33} & \cdots & h_{3,i} \\ \ddots & \ddots & \ddots & \ddots & \vdots \\ h_{i,i-1} & h_{i,i} & h_{i,i} & h_{i,i} & h_{i+1,i} \end{bmatrix}$$

with

$$h_{ji} = \begin{cases} \mathbf{v}_j^T \mathbf{A} \mathbf{v}_i & \text{if } j \leq i, \\ \|\mathbf{w}_{i+1}\|_2 & \text{if } j = i+1, \\ 0 & \text{if } j > i+1. \end{cases}$$

When applying the Arnoldi iteration to solving linear systems, one starts with $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$, the residual corresponding to an initial guess \mathbf{x}_0 . After each step of iteration, one computes $\mathbf{y}_i = \mathbf{H}_i^{-1}(\|\mathbf{r}_0\|_2 \mathbf{e}_1)$ and the new iterate $\mathbf{x}_i = \mathbf{x}_0 + \mathbf{V}_i \mathbf{y}_i$.

75.2.2 The direct Lanczos method

For the rest of discussion, we assume that \mathbf{A} is symmetric positive-definite. With symmetry of \mathbf{A} , the upper Hessenberg matrix $\mathbf{H}_i = \mathbf{V}_i^T \mathbf{A} \mathbf{V}_i$ becomes symmetric and thus tridiagonal. It then can be more clearly denoted by

$$\mathbf{H}_i = \begin{bmatrix} a_1 & b_2 & & & \\ b_2 & a_2 & b_3 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{i-1} & a_{i-1} & b_i \\ & & & b_i & a_i \end{bmatrix}.$$

This enables a short three-term recurrence for \mathbf{v}_i in the iteration, and the Arnoldi iteration is reduced to the Lanczos iteration.

Since \mathbf{A} is symmetric positive-definite, so is \mathbf{H}_i . Hence, \mathbf{H}_i can be LU factorized without partial pivoting into

$$\mathbf{H}_i = \mathbf{L}_i \mathbf{U}_i = \begin{bmatrix} 1 & & & & \\ c_2 & 1 & & & \\ & \ddots & \ddots & & \\ & & c_{i-1} & 1 & \\ & & & c_i & 1 \end{bmatrix} \begin{bmatrix} d_1 & b_2 & & & \\ d_2 & d_3 & b_3 & & \\ & \ddots & \ddots & \ddots & \\ & & & & \end{bmatrix}$$

with convenient recurrences for c_i and d_i :

$$c_i = b_i/d_{i-1},$$

$$d_i = \begin{cases} a_1 & \text{if } i = 1, \\ a_i - c_i b_i & \text{if } i > 1. \end{cases}$$

Rewrite $\mathbf{x}_i = \mathbf{x}_0 + \mathbf{V}_i \mathbf{y}_i$ as

$$\mathbf{x}_i = \mathbf{x}_0 + \mathbf{V}_i \mathbf{H}_i^{-1}(\|\mathbf{r}_0\|_2 \mathbf{e}_1)$$

$$= \mathbf{x}_0 + \mathbf{V}_i \mathbf{U}_i^{-1} \mathbf{L}_i^{-1}(\|\mathbf{r}_0\|_2 \mathbf{e}_1)$$

with

$$\mathbf{P}_i = \mathbf{V}_i \mathbf{U}_i^{-1},$$

$$\mathbf{z}_i = \mathbf{L}_i^{-1}(\|\mathbf{r}_0\|_2 \mathbf{e}_1).$$

It is now important to observe that

$$\mathbf{P}_i = [\mathbf{P}_{i-1} \quad \mathbf{p}_i],$$

$$\mathbf{z}_i = \begin{bmatrix} \mathbf{z}_{i-1} \\ \zeta_i \end{bmatrix}.$$

In fact, there are short recurrences for \mathbf{p}_i and ζ_i as well:

$$\mathbf{p}_i = \frac{1}{d_i} (\mathbf{v}_i - b_i \mathbf{p}_{i-1}),$$

$$\zeta_i = -c_i \zeta_{i-1}.$$

With this formulation, we arrive at a simple recurrence for \mathbf{x}_i :

$$\begin{aligned} \mathbf{x}_i &= \mathbf{x}_0 + \mathbf{P}_i \mathbf{z}_i \\ &= \mathbf{x}_0 + \mathbf{P}_{i-1} \mathbf{z}_{i-1} + \zeta_i \mathbf{p}_i \\ &= \mathbf{x}_{i-1} + \zeta_i \mathbf{p}_i. \end{aligned}$$

The relations above straightforwardly lead to the direct Lanczos method, which turns out to be slightly more complex.

75.2.3 The conjugate gradient method from imposing orthogonality and conjugacy

If we allow \mathbf{p}_i to scale and compensate for the scaling in the constant factor, we potentially can have simpler recurrences of the form:

$$\mathbf{x}_{i-1} = \mathbf{x}_{i-1} + \alpha_{i-1} \mathbf{p}_{i-1},$$

$$\mathbf{b}_{i-1} = \mathbf{r}_{i-1} - \alpha_{i-1} \mathbf{A} \mathbf{p}_{i-1},$$

$$\mathbf{p}_i = \mathbf{r}_{i-1} - \beta_{i-1} \mathbf{p}_{i-1}.$$

As premises for the simplification, we now derive the orthogonality of \mathbf{r}_i and conjugacy of \mathbf{p}_i , i.e., for $i \neq j$,

,

$$\mathbf{r}_i^T \mathbf{r}_j = 0,$$

$$\mathbf{p}_i^T \mathbf{A} \mathbf{p}_j = 0.$$

The residuals are mutually orthogonal because \mathbf{r}_i is essentially a multiple of \mathbf{v}_{i+1} since for $i = 0$, $\mathbf{r}_0 = \|\mathbf{r}_0\|_2 \mathbf{v}_1$, for $i > 0$,

$$\begin{aligned} \mathbf{r}_i &= \mathbf{b} - \mathbf{A} \mathbf{x}_i \\ &= \mathbf{b} - \mathbf{A}(\mathbf{x}_0 + \mathbf{V}_i \mathbf{y}_i) \\ &= \mathbf{r}_0 - \mathbf{A} \mathbf{V}_i \mathbf{y}_i \\ &= \mathbf{r}_0 - \mathbf{V}_{i+1} \tilde{\mathbf{H}}_i \mathbf{y}_i \\ &= \mathbf{r}_0 - \mathbf{V}_i \mathbf{H}_i \mathbf{y}_i - h_{i+1,i} (\mathbf{e}_i^T \mathbf{y}_i) \mathbf{v}_{i+1} \\ &= \|\mathbf{r}_0\|_2 \mathbf{v}_1 - \mathbf{V}_i (\|\mathbf{r}_0\|_2 \mathbf{e}_1) - h_{i+1,i} (\mathbf{e}_i^T \mathbf{y}_i) \mathbf{v}_{i+1} \\ &= -h_{i+1,i} (\mathbf{e}_i^T \mathbf{y}_i) \mathbf{v}_{i+1}. \end{aligned}$$

To see the conjugacy of \mathbf{p}_i , it suffices to show that $\mathbf{P}_i^T \mathbf{A} \mathbf{P}_i$ is diagonal:

$$\begin{aligned} \mathbf{P}_i^T \mathbf{A} \mathbf{P}_i &= \mathbf{U}_i^{-T} \mathbf{V}_i^T \mathbf{A} \mathbf{V}_i \mathbf{U}_i^{-1} \\ &= \mathbf{U}_i^{-T} \mathbf{H}_i \mathbf{U}_i^{-1} \\ &= \mathbf{U}_i^{-T} \mathbf{L}_i \mathbf{U}_i \mathbf{U}_i^{-1} \\ &= \mathbf{U}_i^{-T} \mathbf{L}_i \end{aligned}$$

is symmetric and lower triangular simultaneously and thus must be diagonal.

Now we can derive the constant factors α_i and β_i with respect to the scaled \mathbf{p}_i by solely imposing the orthogonality of \mathbf{r}_i and conjugacy of \mathbf{p}_i .

Due to the orthogonality of \mathbf{r}_i , it is necessary that $\mathbf{r}_{i+1}^T \mathbf{r}_i = (\mathbf{r}_i - \alpha_i \mathbf{A} \mathbf{p}_i)^T \mathbf{r}_i = 0$. As a result,

$$\begin{aligned} \alpha_i &= \frac{\mathbf{r}_i^T \mathbf{r}_i}{\mathbf{r}_i^T \mathbf{A} \mathbf{p}_i} \\ &= \frac{\mathbf{r}_i^T \mathbf{r}_i}{(\mathbf{p}_i - \beta_{i-1} \mathbf{p}_{i-1})^T \mathbf{A} \mathbf{p}_i} \\ &= \frac{\mathbf{r}_i^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{A} \mathbf{p}_i}. \end{aligned}$$

Similarly, due to the conjugacy of \mathbf{p}_i , it is necessary that $\mathbf{p}_{i+1}^T \mathbf{A} \mathbf{p}_i = (\mathbf{r}_{i+1} + \beta_i \mathbf{p}_i)^T \mathbf{A} \mathbf{p}_i = 0$. As a result,

$$\begin{aligned} \beta_i &= -\frac{\mathbf{r}_{i+1}^T \mathbf{A} \mathbf{p}_i}{\mathbf{p}_i^T \mathbf{A} \mathbf{p}_i} \\ &= -\frac{\mathbf{r}_{i+1}^T (\mathbf{r}_i - \mathbf{r}_{i+1})}{\alpha_i \mathbf{p}_i^T \mathbf{A} \mathbf{p}_i} \\ &= \frac{\mathbf{r}_{i+1}^T \mathbf{r}_{i+1}}{\mathbf{r}_i^T \mathbf{r}_i}. \end{aligned}$$

This completes the derivation.

75.3 References

1. Hestenes, M. R.; Stiefel, E. (December 1952). "Methods of conjugate gradients for solving linear systems" (PDF). *Journal of Research of the National Bureau of Standards* **49** (6).
2. Saad, Y. (2003). "Chapter 6: Krylov Subspace Methods, Part I". *Iterative methods for sparse linear systems* (2nd ed.). SIAM. ISBN 978-0-89871-534-7.

Chapter 76

Derivative-free optimization

Derivative free optimization (or **derivative-free optimization**) is a subject of mathematical optimization. It may refer to problems for which derivative information is unavailable (**derivative-free optimization problems**), or methods that do not use derivatives (**derivative-free optimization methods**).^[1]

76.1 Introduction

76.2 Examples of derivative-free optimization problems

76.3 Derivative-free optimization algorithms

Coordinate descent

Cuckoo search

Genetic algorithms

Nelder-Mead method

Particle swarm optimization

Pattern search

76.4 Software

76.5 See also

- Mathematical optimization

76.6 References

[1] Conn, A. R.; Scheinberg, K.; Vicente, L. N. (2009). *Introduction to Derivative-Free Optimization*. MPS-SIAM Book Series on Optimization. Philadelphia: SIAM. Retrieved 2014-01-18.

Chapter 77

Destination dispatch

Destination dispatch is an optimization technique used for multi-elevator installations, which groups passengers for the same destinations into the same elevators, thereby reducing waiting and travel times when compared to a traditional approach where all passengers wishing to ascend or descend enter the same lift and then request their destination.

Using destination dispatch, passengers request travel to a particular floor using a keypad, touch screen or proximity card room-key prior in the lobby and are immediately directed to an appropriate elevator shaft.

77.1 Algorithms

Based on information about the trips that passengers wish to make, the controller will dynamically allocate individuals to elevators to avoid excessive intermediate stops. Overall trip-times can be reduced by 25% with capacity up by 30%.^[1]

Controllers can also offer different levels of service to passengers based on information contained in their key-cards. A high-privilege user may be allocated the nearest available elevator and always be guaranteed a direct service to their floor, and may be allocated an elevator with exclusive use; other users may be provided with extended door-opening times.^[2]

77.2 Limitations

The smooth operation of a destination dispatch system depends upon each passenger indicating their destination intention separately. In most cases, the elevator system has no way of differentiating a group of passengers from a single passenger if the group's destination is only keyed in a single time. This could potentially lead to an elevator stopping to pick up more passengers than the elevator actually has capacity for, creating delays for other users. This situation is handled by two solutions, a load vane sensor on the elevator or a group function button on keypad. The load vane tells the elevator controller that there is a high load in car and doesn't stop at other floors until

the load is low enough to pick up more passengers. The group function button asks for how many passengers are going to a floor, and then the system sends the correct number of elevators to that floor if available.

77.3 Manufacturers

Elevator manufacturers that offer destination dispatch include Otis Elevator, Kone Corporation, ThyssenKrupp, Schindler Group, Fujitec, Mitsubishi and Motion Control Engineering.

77.4 References

- [1] "Destination Dispatch".
- [2] "Personalization".

77.5 External links

- Understanding the Benefits and Limitations of Destination Dispatch

Chapter 78

Differential evolution

In evolutionary computation, **differential evolution** (DE) is a method that optimizes a problem by iteratively trying to improve a **candidate solution** with regard to a given measure of quality. Such methods are commonly known as **metaheuristics** as they make few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, metaheuristics such as DE do not guarantee an optimal solution is ever found.

DE is used for multidimensional real-valued **functions** but does not use the **gradient** of the problem being optimized, which means DE does not require for the optimization problem to be **differentiable** as is required by classic optimization methods such as **gradient descent** and **quasi-newton methods**. DE can therefore also be used on optimization problems that are not even continuous, are noisy, change over time, etc.^[1]

DE optimizes a problem by maintaining a population of candidate solutions and creating new candidate solutions by combining existing ones according to its simple formulae, and then keeping whichever candidate solution has the best score or fitness on the optimization problem at hand. In this way the optimization problem is treated as a black box that merely provides a measure of quality given a candidate solution and the gradient is therefore not needed.

DE is originally due to Storn and Price.^{[2][3]} Books have been published on theoretical and practical aspects of using DE in **parallel computing**, **multiobjective optimization**, **constrained optimization**, and the books also contain surveys of application areas.^{[4][5][6]}

78.1 Algorithm

A basic variant of the DE algorithm works by having a population of **candidate solutions** (called agents). These agents are moved around in the search-space by using simple mathematical **formulae** to combine the positions of existing agents from the population. If the new position of an agent is an improvement it is accepted and forms part of the population, otherwise the new position is simply discarded. The process is repeated and by do-

ing so it is hoped, but not guaranteed, that a satisfactory solution will eventually be discovered.

Formally, let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be the cost function which must be minimized or fitness function which must be maximized. The function takes a candidate solution as argument in the form of a **vector of real numbers** and produces a real number as output which indicates the fitness of the given candidate solution. The **gradient** of f is not known. The goal is to find a solution m for which $f(m) \leq f(p)$ for all p in the search-space, which would mean m is the global minimum. Maximization can be performed by considering the function $h := -f$ instead.

Let $\mathbf{x} \in \mathbb{R}^n$ designate a candidate solution (agent) in the population. The basic DE algorithm can then be described as follows:

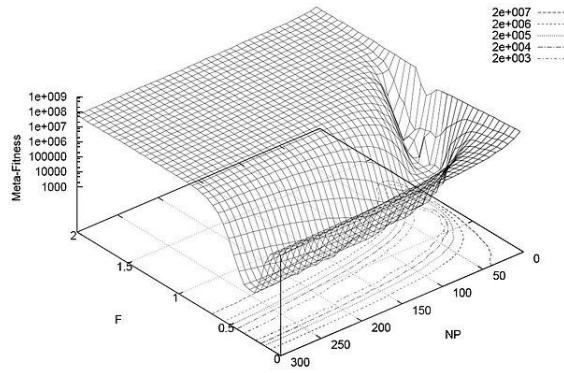
- Initialize all agents \mathbf{x} with random positions in the search-space.
- Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat the following:
 - For each agent \mathbf{x} in the population do:
 - Pick three agents \mathbf{a}, \mathbf{b} , and \mathbf{c} from the population at random, they must be distinct from each other as well as from agent \mathbf{x}
 - Pick a random index $R \in \{1, \dots, n\}$ (n being the dimensionality of the problem to be optimized).
 - Compute the agent's potentially new position $\mathbf{y} = [y_1, \dots, y_n]$ as follows:
 - For each i , pick a uniformly distributed number $r_i \equiv U(0, 1)$
 - If $r_i < CR$ or $i = R$ then set $y_i = a_i + F \times (b_i - c_i)$ otherwise set $y_i = x_i$
 - (In essence, the new position is outcome of binary crossover of agent \mathbf{x} with intermediate agent $\mathbf{z} = \mathbf{a} + F \times (\mathbf{b} - \mathbf{c})$.)
 - If $f(\mathbf{y}) < f(\mathbf{x})$ then replace the agent in the population with the improved candi-

date solution, that is, replace \mathbf{x} with \mathbf{y} in the population.

- Pick the agent from the population that has the highest fitness or lowest cost and return it as the best found candidate solution.

Note that $F \in [0, 2]$ is called the *differential weight* and $CR \in [0, 1]$ is called the *crossover probability*, both these parameters are selectable by the practitioner along with the population size $NP \geq 4$ see below.

78.2 Parameter selection



Performance landscape showing how the basic DE performs in aggregate on the Sphere and Rosenbrock benchmark problems when varying the two DE parameters NP and F , and keeping fixed $CR = 0.9$.

The choice of DE parameters F , CR and NP can have a large impact on optimization performance. Selecting the DE parameters that yield good performance has therefore been the subject of much research. Rules of thumb for parameter selection were devised by Storn et al.^{[3][4]} and Liu and Lampinen.^[7] Mathematical convergence analysis regarding parameter selection was done by Zaharie.^[8] Meta-optimization of the DE parameters was done by Pedersen^{[9][10]} and Zhang et al.^[11]

78.3 Variants

Variants of the DE algorithm are continually being developed in an effort to improve optimization performance. Many different schemes for performing crossover and mutation of agents are possible in the basic algorithm given above, see e.g.^[3] More advanced DE variants are also being developed with a popular research trend being to perturb or adapt the DE parameters during optimization, see e.g. Price et al.,^[4] Liu and Lampinen,^[12] Qin and Suganthan,^[13] Civicioglu^[14] and Brest et al.^[15] There are also some work in making a hybrid optimization method using DE combined with other optimizers.^[16]

78.4 Sample code

The following is a specific pseudocode implementation of differential evolution, written similar to the Java language. For more generalized pseudocode, please see the listing in the [Algorithm section](#) above.

```
//definition of one individual in population class
Individual { //normally DifferentialEvolution uses
floating point variables var float data1, data2 //but
using integers is possible too var integer data3 } class
DifferentialEvolution { //Variables //linked list that
has our population inside var LinkedList<Individual>
population=new LinkedList<Individual>() //New instance
of Random number generator var Random
random=new Random() var integer PopulationSize=20
//differential weight [0,2] var float F=1 //crossover
probability [0,1] var float CR=0.5 //dimensionality
of problem, means how many variables problem has.
this case 3 (data1,data2,data3) var integer N=3; //This
function tells how well given individual performs at
given problem. function float fitnessFunction(Individual
in) { ... return fitness } //this is main function of
program function void Main() { //Initialize population
whit individuals that have been initialized whit uniform
random noise //uniform noise means random value
inside your search space var i=0 while(i<populationSize)
{ var Individual individual= new Individual()
individual.data1=random.UniformNoise() individual.
data2=random.UniformNoise() //integers cant take
floating point values and they need to be either rounded
individual.data3=Math.Floor( random.UniformNoise())
population.add(individual) i++ } i=0 var j //main loop
of evolution. while (!StoppingCriteria) { i++
j=0 while (j<populationSize) { //calculate new candidate
solution //pick random point from population var integer
x=Math.floor(random.UniformNoise()%population.size()-1))
var integer a,b,c //pick three different
random points from population do{
a=Math.floor(random.UniformNoise()%population.size()-1))
}while(a==x); do{ b=Math.floor(random.UniformNoise()%population.size())
}while(b==x) b=a); do{
c=Math.floor(random.UniformNoise()%population.size()-1))
}while(c==x | c==a | c==b); // Pick a random index [0-
Dimensionality] var integer R=rand.nextInt()%N;
//Compute the agent's new position var Individual
original=population.get(x) var Individual
candidate=original.clone() var Individual
individual1=population.get(a) var Individual
individual2=population.get(b) var Individual
individual3=population.get(c) //if(i==R | i<CR)
//candidate=a+f*(b-c) //else //candidate=x if(
Math.floor((random.UniformNoise()%N)==R
| random.UniformNoise()%1<CR){ candidate.
data1=individual1.data1+F*(individual12.data1-
individual3.data1) } // else isn't needed because we
cloned original to candidate if(
Math.floor((random.UniformNoise()%N)==R
```

```

|           random.UniformNoise()%1<CR}{      [7] Liu, J.; Lampinen, J. (2002). "On setting the control pa-
candidate.data2=individual1.data2+F*(individual12.data2-    rameter of the differential evolution method". Proceed-
individual3.data2) } //integer work same as          ings of the 8th International Conference on Soft Computing
floating points but they need to be rounded if(        (MENDEL). Brno, Czech Republic. pp. 11–18.
Math.floor((random.UniformNoise()%N)==R
|           random.UniformNoise()%1<CR}{      [8] Zaharie, D. (2002). "Critical values for the control pa-
candidate.data3=Math.floor(individual1.data3+F*(individual12.data3-    rameters of differential evolution algorithms". Proceed-
individual3.data3)) } //see if is bet-          ings of the 8th International Conference on Soft Computing
ter than original, if so replace          (MENDEL). Brno, Czech Republic. pp. 62–67.
if(fitnessFunction(original)<fitnessFunction(candidate)){      [9] Pedersen, M.E.H. (2010). Tuning & Simplifying Heuris-
population.remove(original) population.add(candidate) }          tical Optimization (PhD thesis). University of Southamp-
j++ } } //find best candidate solution i=0 var Individual          ton, School of Engineering Sciences, Computational En-
bestFitness=new Individual() while (i<populationSize)          gineering and Design Group.
{   var Individual individual=population.get(i)      [10] Pedersen, M.E.H. (2010). "Good parameters for differ-
if(fitnessFunction(bestFitness)<fitnessFunction(individual)){          ential evolution". Technical Report HL1002 (Hvass La-
bestFitness=individual } i++ } //your solution return          batories).
bestFitness } }

```

78.5 See also

- CMA-ES
- Artificial bee colony algorithm
- Evolution strategy
- Genetic algorithm
- Differential search algorithm ^[14]
- Biogeography-based optimization

78.6 References

- [1] Rocca, P.; Oliveri, G.; Massa, A. (2011). "Differential Evolution as Applied to Electromagnetics". *IEEE Antennas and Propagation Magazine* **53** (1): 38–49. doi:10.1109/MAP.2011.5773566.
- [2] Storn, R.; Price, K. (1997). "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces". *Journal of Global Optimization* **11**: 341–359. doi:10.1023/A:1008202821328.
- [3] Storn, R. (1996). "On the usage of differential evolution for function optimization". *Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS)*. pp. 519–523.
- [4] Price, K.; Storn, R.M.; Lampinen, J.A. (2005). *Differential Evolution: A Practical Approach to Global Optimization*. Springer. ISBN 978-3-540-20950-8.
- [5] Feoktistov, V. (2006). *Differential Evolution: In Search of Solutions*. Springer. ISBN 978-0-387-36895-5.
- [6] Chakraborty, U.K., ed. (2008), *Advances in Differential Evolution*, Springer, ISBN 978-3-540-68827-3

- [7] Liu, J.; Lampinen, J. (2002). "On setting the control parameter of the differential evolution method". *Proceedings of the 8th International Conference on Soft Computing (MENDEL)*. Brno, Czech Republic. pp. 11–18.
- [8] Zaharie, D. (2002). "Critical values for the control parameters of differential evolution algorithms". *Proceedings of the 8th International Conference on Soft Computing (MENDEL)*. Brno, Czech Republic. pp. 62–67.
- [9] Pedersen, M.E.H. (2010). *Tuning & Simplifying Heuristic Optimization* (PhD thesis). University of Southampton, School of Engineering Sciences, Computational Engineering and Design Group.
- [10] Pedersen, M.E.H. (2010). "Good parameters for differential evolution". *Technical Report HL1002* (Hvass Laboratories).
- [11] Zhang, X.; Jiang, X.; Scott, P.J. (2011). "A Minimax Fitting Algorithm for Ultra-Precision Aspheric Surfaces". *The 13th International Conference on Metrology and Properties of Engineering Surfaces*.
- [12] Liu, J.; Lampinen, J. (2005). "A fuzzy adaptive differential evolution algorithm". *Soft Computing* **9** (6): 448–462. doi:10.1007/s00500-004-0363-x.
- [13] Qin, A.K.; Suganthan, P.N. (2005). "Self-adaptive differential evolution algorithm for numerical optimization". *Proceedings of the IEEE congress on evolutionary computation (CEC)*. pp. 1785–1791.
- [14] Civicioglu, P. (2012). "Transforming geocentric cartesian coordinates to geodetic coordinates by using differential search algorithm". *Computers & Geosciences* **46**: 229–247. doi:10.1016/j.cageo.2011.12.011.
- [15] Brest, J.; Greiner, S.; Boskovic, B.; Mernik, M.; Zumer, V. (2006). "Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark functions". *IEEE Transactions on Evolutionary Computation* **10** (6): 646–657. doi:10.1109/tevc.2006.872133.
- [16] Zhang, Wen-Jun; Xie, Xiao-Feng (2003). DEPSO: hybrid particle swarm with differential evolution operator. *IEEE International Conference on Systems, Man, and Cybernetics (SMCC)*, Washington, DC, USA: 3816–3821.

78.7 External links

- [Storn's Homepage on DE](#) featuring source-code for several programming languages.
- [Fast DE Algorithm](#) A Fast Differential Evolution Algorithm using k-Nearest Neighbour Predictor.
- [MODE Application](#) Parameter Estimation of a Pressure Swing Adsorption Model for Air Separation Using Multi-objective Optimisation and Support Vector Regression Model.

Chapter 79

Divide and conquer algorithms

In computer science, **divide and conquer (D&C)** is an algorithm design paradigm based on multi-branched recursion. A divide and conquer algorithm works by recursively breaking down a problem into two or more subproblems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

This technique is the basis of efficient algorithms for all kinds of problems, such as sorting (e.g., quicksort, merge sort), multiplying large numbers (e.g. Karatsuba), syntactic analysis (e.g., top-down parsers), and computing the discrete Fourier transform (FFTs).

On the other hand, the ability to understand and design D&C algorithms is a skill that takes time to master. As when proving a **theorem by induction**, it is often necessary to replace the original problem by a more general or complicated problem in order to initialize the recursion, and there is no systematic method for finding the proper generalization. These D&C complications are seen when optimizing the calculation of a **Fibonacci number** with **efficient double recursion**.

The correctness of a divide and conquer algorithm is usually proved by **mathematical induction**, and its computational cost is often determined by solving **recurrence relations**.

79.1 Decrease and conquer

The name “divide and conquer” is sometimes applied also to algorithms that reduce each problem to only one subproblem, such as the **binary search** algorithm for finding a record in a sorted list (or its analog in **numerical computing**, the **bisection algorithm** for root finding).^[1] These algorithms can be implemented more efficiently than general divide-and-conquer algorithms; in particular, if they use **tail recursion**, they can be converted into simple **loops**. Under this broad definition, however, every algorithm that uses recursion or loops could be regarded as a “divide and conquer algorithm”. Therefore, some authors consider that the name “divide and conquer” should be used only when each problem may gen-

erate two or more subproblems.^[2] The name **decrease and conquer** has been proposed instead for the single-subproblem class.^[3]

An important application of decrease and conquer is in optimization, where if the search space is reduced (“pruned”) by a constant factor at each step, the overall algorithm has the same asymptotic complexity as the pruning step, with the constant depending on the pruning factor (by summing the geometric series); this is known as **prune and search**.

79.2 Early historical examples

Early examples of these algorithms are primarily decrease and conquer – the original problem is successively broken down into *single* subproblems, and indeed can be solved iteratively.

Binary search, a decrease and conquer algorithm where the subproblems are of roughly half the original size, has a long history. While a clear description of the algorithm on computers appeared in 1946 in an article by **John Mauchly**, the idea of using a sorted list of items to facilitate searching dates back at least as far as **Babylonia** in 200 BC.^[4] Another ancient decrease and conquer algorithm is the **Euclidean algorithm** to compute the **greatest common divisor** of two numbers (by reducing the numbers to smaller and smaller equivalent subproblems), which dates to several centuries BC.

An early example of a divide-and-conquer algorithm with multiple subproblems is **Gauss's** 1805 description of what is now called the **Cooley-Tukey fast Fourier transform (FFT)** algorithm,^[5] although he did not analyze its operation count quantitatively and FFTs did not become widespread until they were rediscovered over a century later.

An early two-subproblem D&C algorithm that was specifically developed for computers and properly analyzed is the **merge sort** algorithm, invented by **John von Neumann** in 1945.^[6]

Another notable example is the algorithm invented by **Anatolii A. Karatsuba** in 1960^[7] that could multiply two

n -digit numbers in $O(n^{\log_2 3})$ operations (in Big O notation). This algorithm disproved Andrey Kolmogorov's 1956 conjecture that $\Omega(n^2)$ operations would be required for that task.

As another example of a divide and conquer algorithm that did not originally involve computers, Knuth gives the method a post office typically uses to route mail: letters are sorted into separate bags for different geographical areas, each of these bags is itself sorted into batches for smaller sub-regions, and so on until they are delivered.^[4] This is related to a radix sort, described for punch-card sorting machines as early as 1929.^[4]

79.3 Advantages

79.3.1 Solving difficult problems

Divide and conquer is a powerful tool for solving conceptually difficult problems: all it requires is a way of breaking the problem into sub-problems, of solving the trivial cases and of combining sub-problems to the original problem. Similarly, decrease and conquer only requires reducing the problem to a single smaller problem, such as the classic Tower of Hanoi puzzle, which reduces moving a tower of height n to moving a tower of height $n - 1$.

79.3.2 Algorithm efficiency

The divide-and-conquer paradigm often helps in the discovery of efficient algorithms. It was the key, for example, to Karatsuba's fast multiplication method, the quicksort and mergesort algorithms, the Strassen algorithm for matrix multiplication, and fast Fourier transforms.

In all these examples, the D&C approach led to an improvement in the asymptotic cost of the solution. For example, if the base cases have constant-bounded size, the work of splitting the problem and combining the partial solutions is proportional to the problem's size n , and there are a bounded number p of subproblems of size $\sim n/p$ at each stage, then the cost of the divide-and-conquer algorithm will be $O(n \log n)$.

79.3.3 Parallelism

Divide and conquer algorithms are naturally adapted for execution in multi-processor machines, especially shared-memory systems where the communication of data between processors does not need to be planned in advance, because distinct sub-problems can be executed on different processors.

79.3.4 Memory access

Divide-and-conquer algorithms naturally tend to make efficient use of memory caches. The reason is that once a sub-problem is small enough, it and all its sub-problems can, in principle, be solved within the cache, without accessing the slower main memory. An algorithm designed to exploit the cache in this way is called *cache-oblivious*, because it does not contain the cache size(s) as an explicit parameter.^[8] Moreover, D&C algorithms can be designed for important algorithms (e.g., sorting, FFTs, and matrix multiplication) to be *optimal* cache-oblivious algorithms—they use the cache in a probably optimal way, in an asymptotic sense, regardless of the cache size. In contrast, the traditional approach to exploiting the cache is *blocking*, as in *loop nest optimization*, where the problem is explicitly divided into chunks of the appropriate size—this can also use the cache optimally, but only when the algorithm is tuned for the specific cache size(s) of a particular machine.

The same advantage exists with regards to other hierarchical storage systems, such as NUMA or virtual memory, as well as for multiple levels of cache: once a sub-problem is small enough, it can be solved within a given level of the hierarchy, without accessing the higher (slower) levels.

79.3.5 Roundoff control

In computations with rounded arithmetic, e.g. with floating point numbers, a divide-and-conquer algorithm may yield more accurate results than a superficially equivalent iterative method. For example, one can add N numbers either by a simple loop that adds each datum to a single variable, or by a D&C algorithm called pairwise summation that breaks the data set into two halves, recursively computes the sum of each half, and then adds the two sums. While the second method performs the same number of additions as the first, and pays the overhead of the recursive calls, it is usually more accurate.^[9]

79.4 Implementation issues

79.4.1 Recursion

Divide-and-conquer algorithms are naturally implemented as recursive procedures. In that case, the partial sub-problems leading to the one currently being solved are automatically stored in the procedure call stack. A recursive function is a function that is defined in terms of itself.

79.4.2 Explicit stack

Divide and conquer algorithms can also be implemented by a non-recursive program that stores the partial subproblems in some explicit data structure, such as a **stack**, **queue**, or **priority queue**. This approach allows more freedom in the choice of the sub-problem that is to be solved next, a feature that is important in some applications — e.g. in **breadth-first recursion** and the **branch and bound** method for function optimization. This approach is also the standard solution in programming languages that do not provide support for recursive procedures.

79.4.3 Stack size

In recursive implementations of D&C algorithms, one must make sure that there is sufficient memory allocated for the recursion stack, otherwise the execution may fail because of **stack overflow**. Fortunately, D&C algorithms that are time-efficient often have relatively small recursion depth. For example, the quicksort algorithm can be implemented so that it never requires more than $\log_2 n$ nested recursive calls to sort n items.

Stack overflow may be difficult to avoid when using recursive procedures, since many compilers assume that the recursion stack is a contiguous area of memory, and some allocate a fixed amount of space for it. Compilers may also save more information in the recursion stack than is strictly necessary, such as return address, unchanging parameters, and the internal variables of the procedure. Thus, the risk of stack overflow can be reduced by minimizing the parameters and internal variables of the recursive procedure, and/or by using an explicit stack structure.

79.4.4 Choosing the base cases

In any recursive algorithm, there is considerable freedom in the choice of the **base cases**, the small subproblems that are solved directly in order to terminate the recursion.

Choosing the smallest or simplest possible base cases is more elegant and usually leads to simpler programs, because there are fewer cases to consider and they are easier to solve. For example, an FFT algorithm could stop the recursion when the input is a single sample, and the quicksort list-sorting algorithm could stop when the input is the empty list; in both examples there is only one base case to consider, and it requires no processing.

On the other hand, efficiency often improves if the recursion is stopped at relatively large base cases, and these are solved non-recursively, resulting in a **hybrid algorithm**. This strategy avoids the overhead of recursive calls that do little or no work, and may also allow the use of specialized non-recursive algorithms that, for those base cases, are more efficient than explicit recursion. A general procedure for a simple hybrid recursive algorithm is *short-*

circuiting the base case, also known as **arm's-length recursion**. In this case whether the next step will result in the base case is checked before the function call, avoiding an unnecessary function call. For example, in a tree, rather than recursing to a child node and then checking if it is null, checking null before recursing; this avoids half the function calls in some algorithms on binary trees. Since a D&C algorithm eventually reduces each problem or subproblem instance to a large number of base instances, these often dominate the overall cost of the algorithm, especially when the splitting/joining overhead is low. Note that these considerations do not depend on whether recursion is implemented by the compiler or by an explicit stack.

Thus, for example, many library implementations of quicksort will switch to a simple loop-based **insertion sort** (or similar) algorithm once the number of items to be sorted is sufficiently small. Note that, if the empty list were the only base case, sorting a list with n entries would entail $n+1$ quicksort calls that would do nothing but return immediately. Increasing the base cases to lists of size 2 or less will eliminate most of those do-nothing calls, and more generally a base case larger than 2 is typically used to reduce the fraction of time spent in function-call overhead or stack manipulation.

Alternatively, one can employ large base cases that still use a divide-and-conquer algorithm, but implement the algorithm for predetermined set of fixed sizes where the algorithm can be completely unrolled into code that has no recursion, loops, or **conditionals** (related to the technique of **partial evaluation**). For example, this approach is used in some efficient FFT implementations, where the base cases are unrolled implementations of divide-and-conquer FFT algorithms for a set of fixed sizes.^[10] Source code generation methods may be used to produce the large number of separate base cases desirable to implement this strategy efficiently.^[10]

The generalized version of this idea is known as recursion “unrolling” or “coarsening” and various techniques have been proposed for automating the procedure of enlarging the base case.^[11]

79.4.5 Sharing repeated subproblems

For some problems, the branched recursion may end up evaluating the same sub-problem many times over. In such cases it may be worth identifying and saving the solutions to these overlapping subproblems, a technique commonly known as **memoization**. Followed to the limit, it leads to **bottom-up** divide-and-conquer algorithms such as **dynamic programming** and **chart parsing**.

79.5 See also

- Akra–Bazzi method

- Fork–join model
- Master theorem
- Mathematical induction
- MapReduce

79.6 References

- [1] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms* (MIT Press, 2000).
- [2] Brassard, G. and Bratley, P. Fundamental of Algorithms, Prentice-Hall, 1996.
- [3] Anany V. Levitin, *Introduction to the Design and Analysis of Algorithms* (Addison Wesley, 2002).
- [4] Donald E. Knuth, *The Art of Computer Programming: Volume 3, Sorting and Searching*, second edition (Addison-Wesley, 1998).
- [5] Heideman, M. T., D. H. Johnson, and C. S. Burrus, “Gauss and the history of the fast Fourier transform,” IEEE ASSP Magazine, 1, (4), 14–21 (1984)
- [6] Knuth, Donald (1998). *The Art of Computer Programming: Volume 3 Sorting and Searching*. p. 159. ISBN 0-201-89685-0.
- [7] Karatsuba, Anatolii A.; Yuri P. Ofman (1962). "Умножение многозначных чисел на автоматах". *Doklady Akademii Nauk SSSR* **146**: 293–294. Translated in *Physics-Doklady* **7**: 595–596. 1963. Missing or empty |title= (help)
- [8] M. Frigo; C. E. Leiserson; H. Prokop (1999). “Cache-oblivious algorithms”. *Proc. 40th Symp. on the Foundations of Computer Science*.
- [9] Nicholas J. Higham, “The accuracy of floating point summation”, *SIAM J. Scientific Computing* **14** (4), 783–799 (1993).
- [10] Frigo, M.; Johnson, S. G. (February 2005). “The design and implementation of FFTW3”. *Proceedings of the IEEE* **93** (2): 216–231. doi:10.1109/JPROC.2004.840301.
- [11] Radu Rugina and Martin Rinard, "Recursion unrolling for divide and conquer programs," in *Languages and Compilers for Parallel Computing*, chapter 3, pp. 34–48. *Lecture Notes in Computer Science* vol. 2017 (Berlin: Springer, 2001).

79.7 External links

Chapter 80

Dykstra's projection algorithm

Not to be confused with Dijkstra's algorithm.

Dykstra's algorithm is a method that computes a point in the intersection of convex sets, and is a variant of the alternating projection method (also called the projections onto convex sets method). In its simplest form, the method finds a point in the intersection of two convex sets by iteratively projecting onto each of the convex set; it differs from the alternating projection method in that there are intermediate steps. A parallel version of the algorithm was developed by Gaffke and Mathar.

The method is named after R. L. Dykstra who proposed it in the 1980s.

A key difference between Dykstra's algorithm and the standard alternating projection method occurs when there is more than one point in the intersection of the two sets. In this case, the alternating projection method gives some arbitrary point in this intersection, whereas Dykstra's algorithm gives a specific point: the projection of r onto the intersection, where r is the initial point used in the algorithm,

$$\|\bar{x} - r\|^2 \leq \|x - r\|^2, \text{ all for } x \in C \cap D,$$

where C, D are convex sets. This problem is equivalent to finding the projection of r onto the set $C \cap D$, which we denote by $\mathcal{P}_{C \cap D}$.

To use Dykstra's algorithm, one must know how to project onto the sets C and D separately.

First, consider the basic alternating projection (aka POCS) method (first studied, in the case when the sets C, D were linear subspaces, by John von Neumann^[1]), which initializes $x_0 = r$ and then generates the sequence

$$x_{k+1} = \mathcal{P}_C(\mathcal{P}_D(x_k))$$

Dykstra's algorithm is of a similar form, but uses additional auxiliary variables. Start with $x_0 = r, p_0 = q_0 = 0$ and update by

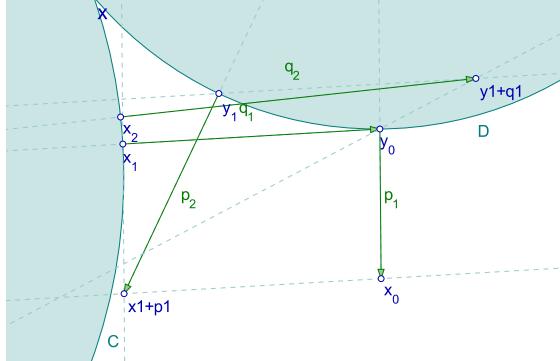
$$y_k = \mathcal{P}_D(x_k + p_k)$$

$$p_{k+1} = x_k + p_k - y_k$$

$$x_{k+1} = \mathcal{P}_C(y_k + q_k)$$

$$q_{k+1} = y_k + q_k - x_{k+1}.$$

Then the sequence (x_k) converges to the solution of the original problem. For convergence results and a modern perspective on the literature, see ^[2]



Dykstra's algorithm finds for each r the only $\bar{x} \in C \cap D$ such that:

80.2 References

- Boyle, J. P.; Dykstra, R. L. (1986). “A method for finding projections onto the intersection of convex sets in Hilbert spaces”. *Lecture Notes in Statistics* **37**: 28–47. doi:10.1007/978-1-4613-9940-7_3.
- Gaffke, N.; Mathar, R. (1989). “A cyclic projection algorithm via duality”. *Metrika* **36**: 29–54. doi:10.1007/bf02614077.

- [1] J. von Neumann, On rings of operators. Reduction theory, Ann. of Math. 50 (1949) 401–485 (a reprint of lecture notes first distributed in 1933).
- [2] P. L. Combettes and J.-C. Pesquet, “Proximal splitting methods in signal processing,” in: Fixed-Point Algorithms for Inverse Problems in Science and Engineering, (H. H. Bauschke, R. S. Burachik, P. L. Combettes, V. Elser, D. R. Luke, and H. Wolkowicz, Editors), pp. 185–212. Springer, New York, 2011

Chapter 81

Eagle strategy

Eagle strategy is a search strategy for solving nonlinear optimization problems, and this strategy was developed by Xin-she Yang and Suash Deb, based on the foraging behaviour of eagle species such as golden eagles.^[1]

In optimization, a common strategy is to search for the optimal solution starting from a set of initial guess solutions (either random and educated guess). In the case when the cost functions are multimodal with multiple local best solutions, the final solutions may heavily depend on the initial starting solutions. In order to minimize such dependence on initial random solutions, most modern algorithms, especially metaheuristic algorithms, are able to escape local optima by using some sophisticated random techniques. However, most of these algorithms are one-stage type; that is, once initialization is done, the search process continues until an algorithm stops. Running an algorithm many times from different initial solutions may occasionally improve the overall performance on average.

Eagle strategy improves this by using an iterative, interacting two-stage strategy to enhance the search efficiency by escaping the local optima and use initial solutions in different regions. It uses a slow search stage and a fast stage to simulate an eagle searching for prey tends to search on a large area and then quickly switches to a rapid chasing phase once a prey is in sight.^[2] In optimization, it uses a coarse search stage on a larger area in a search space in combination with an intensive faster search algorithm in the neighbourhood of promising solutions. Two stages interchanges and proceed iteratively.

As there are two stages in the strategy, each stage can employ different algorithms. For example, differential evolution can be used within eagle strategy.^[3] Studies show that such a combination is better than any of its components.^[4]

In the simplest case, when the first stage does not use any algorithm (just initialization), it essentially degenerates into a hill-climbing with random restart. However, this strategy could be potentially much more powerful if a good combination of different algorithms is used.

81.1 References

- [1] X. S. Yang and S. Deb, Eagle strategy using Levy walk and firefly algorithms for stochastic optimization, in: Nature Inspired Cooperative Strategies for Optimization (NICSO 2010) (Eds. J. R. Gonzalez et al.), Vol. 284, 101-111 (2010).
- [2] M. W. Collopy, Foraging behavior and success of golden eagles, *The Auk*, Vol. 100, 747-749 (1983).
- [3] X. S. Yang and S. Deb, Two-stage eagle strategy with differential evolution, *Int. J. Bio-Inspired Computation*, Vol. 4(1), pp. 1-5 (2012). <http://arxiv.org/abs/1203.6586>
- [4] A. H. Gandomi, X. S. Yang, S. Talatahari, S. Deb, Coupled eagle strategy and differential evolution for unconstrained and constrained global optimization, *Computers & Mathematics with Applications*, Vol. 63(1), 191-200 (2012). <http://dx.doi.org/10.1016/j.camwa.2011.11.010>

Chapter 82

Evolutionary programming

Evolutionary programming is one of the four major evolutionary algorithm paradigms. It is similar to genetic programming, but the structure of the program to be optimized is fixed, while its numerical parameters are allowed to evolve.

It was first used by Lawrence J. Fogel in the US in 1960 in order to use simulated evolution as a learning process aiming to generate artificial intelligence. Fogel used finite state machines as predictors and evolved them. Currently evolutionary programming is a wide evolutionary computing dialect with no fixed structure or (representation), in contrast with some of the other dialects. It is becoming harder to distinguish from evolutionary strategies.

Its main variation operator is mutation; members of the population are viewed as part of a specific species rather than members of the same species therefore each parent generates an offspring, using a ($\mu + \mu$) survivor selection.

82.1 See also

- Artificial intelligence
- Genetic algorithm
- Genetic operator

82.2 References

- Fogel, L.J., Owens, A.J., Walsh, M.J. (1966), *Artificial Intelligence through Simulated Evolution*, John Wiley.
- Fogel, L.J. (1999), *Intelligence through Simulated Evolution : Forty Years of Evolutionary Programming*, John Wiley.
- Eiben, A.E., Smith, J.E. (2003), *Introduction to Evolutionary Computing*, Springer. ISBN 3-540-40184-9

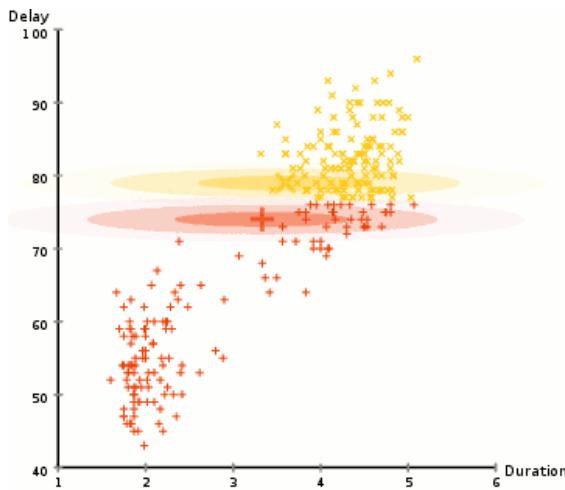
82.3 External links

- The Hitch-Hiker's Guide to Evolutionary Computation: What's Evolutionary Programming (EP)?
- Evolutionary Programming by Jason Brownlee (PhD)

Chapter 83

Expectation–maximization algorithm

In statistics, an **expectation–maximization (EM) algorithm** is an iterative method for finding maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables. The EM iteration alternates between performing an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the E step. These parameter-estimates are then used to determine the distribution of the latent variables in the next E step.



EM clustering of Old Faithful eruption data. The random initial model (which, due to the different scales of the axes, appears to be two very flat and wide spheres) is fit to the observed data. In the first iterations, the model changes substantially, but then converges to the two modes of the geyser. Visualized using ELKI.

83.1 History

The EM algorithm was explained and given its name in a classic 1977 paper by Arthur Dempster, Nan Laird, and Donald Rubin.^[1] They pointed out that the method had been “proposed many times in special circumstances” by earlier authors. In particular, a very de-

tailed treatment of the EM method for exponential families was published by Rolf Sundberg in his thesis and several papers^{[2][3][4]} following his collaboration with Per Martin-Löf and Anders Martin-Löf.^{[5][6][7][8][9][10][11]} The Dempster-Laird-Rubin paper in 1977 generalized the method and sketched a convergence analysis for a wider class of problems. Regardless of earlier inventions, the innovative Dempster-Laird-Rubin paper in the *Journal of the Royal Statistical Society* received an enthusiastic discussion at the Royal Statistical Society meeting with Sundberg calling the paper “brilliant”. The Dempster-Laird-Rubin paper established the EM method as an important tool of statistical analysis.

The convergence analysis of the Dempster-Laird-Rubin paper was flawed and a correct convergence analysis was published by C.F. Jeff Wu in 1983.^[12] Wu’s proof established the EM method’s convergence outside of the **exponential family**, as claimed by Dempster-Laird-Rubin.^[13]

83.2 Introduction

The EM algorithm is used to find the maximum likelihood parameters of a statistical model in cases where the equations cannot be solved directly. Typically these models involve **latent variables** in addition to unknown **parameters** and known data observations. That is, either there are **missing values** among the data, or the model can be formulated more simply by assuming the existence of additional unobserved data points. For example, a mixture model can be described more simply by assuming that each observed data point has a corresponding unobserved data point, or latent variable, specifying the mixture component that each data point belongs to.

Finding a maximum likelihood solution typically requires taking the **derivatives** of the **likelihood function** with respect to all the unknown values — viz. the parameters and the latent variables — and simultaneously solving the resulting equations. In statistical models with latent variables, this usually is not possible. Instead, the result is typically a set of interlocking equations in which the solution to the parameters requires the values of the latent

variables and vice versa, but substituting one set of equations into the other produces an unsolvable equation.

The EM algorithm proceeds from the observation that the following is a way to solve these two sets of equations numerically. One can simply pick arbitrary values for one of the two sets of unknowns, use them to estimate the second set, then use these new values to find a better estimate of the first set, and then keep alternating between the two until the resulting values both converge to fixed points. It's not obvious that this will work at all, but in fact it can be proven that in this particular context it does, and that the derivative of the likelihood is (arbitrarily close to) zero at that point, which in turn means that the point is either a maximum or a saddle point. In general there may be multiple maxima, and there is no guarantee that the global maximum will be found. Some likelihoods also have singularities in them, i.e. nonsensical maxima. For example, one of the “solutions” that may be found by EM in a mixture model involves setting one of the components to have zero variance and the mean parameter for the same component to be equal to one of the data points.

83.3 Description

Given a statistical model which generates a set \mathbf{X} of observed data, a set of unobserved latent data or **missing values** \mathbf{Z} , and a vector of unknown parameters $\boldsymbol{\theta}$, along with a likelihood function $L(\boldsymbol{\theta}; \mathbf{X}, \mathbf{Z}) = p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$, the **maximum likelihood estimate** (MLE) of the unknown parameters is determined by the **marginal likelihood** of the observed data

$$L(\boldsymbol{\theta}; \mathbf{X}) = p(\mathbf{X}|\boldsymbol{\theta}) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$$

However, this quantity is often intractable (e.g. if \mathbf{Z} is a sequence of events, so that the number of values grows exponentially with the sequence length, making the exact calculation of the sum extremely difficult).

The EM algorithm seeks to find the MLE of the marginal likelihood by iteratively applying the following two steps:

Expectation step (E step): Calculate the **expected value** of the **log likelihood** function, with respect to the **conditional distribution** of \mathbf{Z} given \mathbf{X} under the current estimate of the parameters $\boldsymbol{\theta}^{(t)}$:

$$Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) = E_{\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{(t)}} [\log L(\boldsymbol{\theta}; \mathbf{X}, \mathbf{Z})]$$

Maximization step (M step): Find the parameter that maximizes this quantity:

$$\boldsymbol{\theta}^{(t+1)} = \arg \max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$$

Note that in typical models to which EM is applied:

1. The observed data points \mathbf{X} may be discrete (taking values in a finite or countably infinite set) or continuous (taking values in an uncountably infinite set). There may in fact be a vector of observations associated with each data point.
2. The missing values (aka **latent variables**) \mathbf{Z} are discrete, drawn from a fixed number of values, and there is one latent variable per observed data point.
3. The parameters are continuous, and are of two kinds: Parameters that are associated with all data points, and parameters associated with a particular value of a latent variable (i.e. associated with all data points whose corresponding latent variable has a particular value).

However, it is possible to apply EM to other sorts of models.

The motivation is as follows. If we know the value of the parameters $\boldsymbol{\theta}$, we can usually find the value of the latent variables \mathbf{Z} by maximizing the log-likelihood over all possible values of \mathbf{Z} , either simply by iterating over \mathbf{Z} or through an algorithm such as the **Viterbi algorithm** for **hidden Markov models**. Conversely, if we know the value of the latent variables \mathbf{Z} , we can find an estimate of the parameters $\boldsymbol{\theta}$ fairly easily, typically by simply grouping the observed data points according to the value of the associated latent variable and averaging the values, or some function of the values, of the points in each group. This suggests an iterative algorithm, in the case where both $\boldsymbol{\theta}$ and \mathbf{Z} are unknown:

1. First, initialize the parameters $\boldsymbol{\theta}$ to some random values.
2. Compute the best value for \mathbf{Z} given these parameter values.
3. Then, use the just-computed values of \mathbf{Z} to compute a better estimate for the parameters $\boldsymbol{\theta}$. Parameters associated with a particular value of \mathbf{Z} will use only those data points whose associated latent variable has that value.
4. Iterate steps 2 and 3 until convergence.

The algorithm as just described monotonically approaches a local minimum of the cost function, and is commonly called **hard EM**. The **k-means** algorithm is an example of this class of algorithms.

However, one can do somewhat better: Rather than making a hard choice for \mathbf{Z} given the current parameter values and averaging only over the set of data points associated with a particular value of \mathbf{Z} , one can instead determine the probability of each possible value of \mathbf{Z} for

each data point, and then use the probabilities associated with a particular value of \mathbf{Z} to compute a **weighted average** over the entire set of data points. The resulting algorithm is commonly called *soft EM*, and is the type of algorithm normally associated with EM. The counts used to compute these weighted averages are called *soft counts* (as opposed to the *hard counts* used in a hard-EM-type algorithm such as *k*-means). The probabilities computed for \mathbf{Z} are **posterior probabilities** and are what is computed in the E step. The soft counts used to compute new parameter values are what is computed in the M step.

83.4 Properties

Speaking of an expectation (E) step is a bit of a **mismuter**. What is calculated in the first step are the fixed, data-dependent parameters of the function Q . Once the parameters of Q are known, it is fully determined and is maximized in the second (M) step of an EM algorithm.

Although an EM iteration does increase the observed data (i.e. marginal) likelihood function there is no guarantee that the sequence converges to a **maximum likelihood estimator**. For **multimodal distributions**, this means that an EM algorithm may converge to a **local maximum** of the observed data likelihood function, depending on starting values. There are a variety of heuristic or **metaheuristic** approaches for escaping a local maximum such as **random restart** (starting with several different random initial estimates $\theta^{(t)}$), or applying **simulated annealing** methods.

EM is particularly useful when the likelihood is an **exponential family**: the E step becomes the sum of expectations of sufficient statistics, and the M step involves maximizing a linear function. In such a case, it is usually possible to derive **closed form** updates for each step, using the Sundberg formula (published by Rolf Sundberg using unpublished results of Per Martin-Löf and Anders Martin-Löf).^{[3][4][7][8][9][10][11]}

The EM method was modified to compute **maximum a posteriori** (MAP) estimates for **Bayesian inference** in the original paper by Dempster, Laird, and Rubin.

There are other methods for finding maximum likelihood estimates, such as **gradient descent**, **conjugate gradient** or variations of the **Gauss–Newton** method. Unlike EM, such methods typically require the evaluation of first and/or second derivatives of the likelihood function.

83.5 Proof of correctness

Expectation-maximization works to improve $Q(\theta|\theta^{(t)})$ rather than directly improving $\log p(\mathbf{X}|\theta)$. Here we show that improvements to the former imply improvements to the latter.^[14]

For any \mathbf{Z} with non-zero probability $p(\mathbf{Z}|\mathbf{X}, \theta)$, we can

write

$$\log p(\mathbf{X}|\theta) = \log p(\mathbf{X}, \mathbf{Z}|\theta) - \log p(\mathbf{Z}|\mathbf{X}, \theta).$$

We take the expectation over values of \mathbf{Z} by multiplying both sides by $p(\mathbf{Z}|\mathbf{X}, \theta^{(t)})$ and summing (or integrating) over \mathbf{Z} . The left-hand side is the expectation of a constant, so we get:

$$\begin{aligned} \log p(\mathbf{X}|\theta) &= \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \theta^{(t)}) \log p(\mathbf{X}, \mathbf{Z}|\theta) - \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \theta^{(t)}) \log p(\mathbf{Z}|\mathbf{X}, \theta^{(t)}) \\ &= Q(\theta|\theta^{(t)}) + H(\theta|\theta^{(t)}), \end{aligned}$$

where $H(\theta|\theta^{(t)})$ is defined by the negated sum it is replacing. This last equation holds for any value of θ including $\theta = \theta^{(t)}$,

$$\log p(\mathbf{X}|\theta^{(t)}) = Q(\theta^{(t)}|\theta^{(t)}) + H(\theta^{(t)}|\theta^{(t)}),$$

and subtracting this last equation from the previous equation gives

$$\log p(\mathbf{X}|\theta) - \log p(\mathbf{X}|\theta^{(t)}) = Q(\theta|\theta^{(t)}) - Q(\theta^{(t)}|\theta^{(t)}) + H(\theta|\theta^{(t)}) - H(\theta^{(t)}|\theta^{(t)}).$$

However, Gibbs' inequality tells us that $H(\theta|\theta^{(t)}) \geq H(\theta^{(t)}|\theta^{(t)})$, so we can conclude that

$$\log p(\mathbf{X}|\theta) - \log p(\mathbf{X}|\theta^{(t)}) \geq Q(\theta|\theta^{(t)}) - Q(\theta^{(t)}|\theta^{(t)}).$$

In words, choosing θ to improve $Q(\theta|\theta^{(t)})$ beyond $Q(\theta^{(t)}|\theta^{(t)})$ will improve $\log p(\mathbf{X}|\theta)$ beyond $\log p(\mathbf{X}|\theta^{(t)})$ at least as much.

83.6 Alternative description

Under some circumstances, it is convenient to view the EM algorithm as two alternating maximization steps.^{[15][16]} Consider the function:

$$F(q, \theta) = \mathbb{E}_q[\log L(\theta; x, Z)] + H(q) = -D_{\text{KL}}(q \| p_{Z|X}(\cdot|x; \theta)) + \log L(\theta;$$

where q is an arbitrary probability distribution over the unobserved data z , $p_{Z|X}(\cdot|x; \theta)$ is the conditional distribution of the unobserved data given the observed data x , H is the **entropy** and D_{KL} is the **Kullback–Leibler divergence**.

Then the steps in the EM algorithm may be viewed as:

Expectation step: Choose q to maximize F :

$$q^{(t)} = * \arg \max_q F(q, \theta^{(t)})$$

Maximization step: Choose θ to maximize F :

$$\theta^{(t+1)} = * \arg \max_{\theta} F(q^{(t)}, \theta)$$

83.7 Applications

EM is frequently used for data clustering in machine learning and computer vision. In natural language processing, two prominent instances of the algorithm are the Baum-Welch algorithm and the inside-outside algorithm for unsupervised induction of probabilistic context-free grammars.

In psychometrics, EM is almost indispensable for estimating item parameters and latent abilities of item response theory models.

With the ability to deal with missing data and observe unidentified variables, EM is becoming a useful tool to price and manage risk of a portfolio.

The EM algorithm (and its faster variant Ordered subset expectation maximization) is also widely used in medical image reconstruction, especially in positron emission tomography and single photon emission computed tomography. See below for other faster variants of EM.

83.8 Filtering and smoothing EM algorithms

A Kalman filter is typically used for on-line state estimation and a minimum-variance smoother may be employed for off-line or batch state estimation. However, these minimum-variance solutions require estimates of the state-space model parameters. EM algorithms can be used for solving joint state and parameter estimation problems.

Filtering and smoothing EM algorithms arise by repeating the following two-step procedure:

E-step Operate a Kalman filter or a minimum-variance smoother designed with current parameter estimates to obtain updated state estimates.

M-step Use the filtered or smoothed state estimates within maximum-likelihood calculations to obtain updated parameter estimates.

Suppose that a Kalman filter or minimum-variance smoother operates on noisy measurements of a single-input-single-output system. An updated measurement noise variance estimate can be obtained from the maximum likelihood calculation

$$\hat{\sigma}_v^2 = \frac{1}{N} \sum_{k=1}^N (z_k - \hat{x}_k)^2$$

where \hat{x}_k are scalar output estimates calculated by a filter or a smoother from N scalar measurements z_k . Similarly, for a first-order auto-regressive process, an updated process noise variance estimate can be calculated by

$$\hat{\sigma}_w^2 = \frac{1}{N} \sum_{k=1}^N (\hat{x}_{k+1} - \hat{F}\hat{x}_k)^2$$

where \hat{x}_k and \hat{x}_{k+1} are scalar state estimates calculated by a filter or a smoother. The updated model coefficient estimate is obtained via

$$\hat{F} = \frac{\sum_{k=1}^N (\hat{x}_{k+1} - \hat{F}\hat{x}_k)}{\sum_{k=1}^N \hat{x}_k^2}$$

The convergence of parameter estimates such as those above are well studied.^{[17][18][19]}

83.9 Variants

A number of methods have been proposed to accelerate the sometimes slow convergence of the EM algorithm, such as those using conjugate gradient and modified Newton–Raphson techniques.^[20] Additionally EM can be used with constrained estimation techniques.

Expectation conditional maximization (ECM) replaces each M step with a sequence of conditional maximization (CM) steps in which each parameter θ_i is maximized individually, conditionally on the other parameters remaining fixed.^[21]

This idea is further extended in **generalized expectation maximization (GEM)** algorithm, in which one only seeks an increase in the objective function F for both the E step and M step under the alternative description.^[15]

It is also possible to consider the EM algorithm as a subclass of the **MM** (Majorize/Minimize or Minorize/Maximize, depending on context) algorithm,^[22] and therefore use any machinery developed in the more general case.

83.9.1 α -EM algorithm

The Q-function used in the EM algorithm is based on the log likelihood. Therefore, it is regarded as the log-EM algorithm. The use of the log likelihood can be generalized to that of the α -log likelihood ratio. Then, the α -log likelihood ratio of the observed data can be exactly expressed as equality by using the Q-function of the α -log likelihood

ratio and the α -divergence. Obtaining this Q-function is a generalized E step. Its maximization is a generalized M step. This pair is called the α -EM algorithm [23] which contains the log-EM algorithm as its subclass. Thus, the α -EM algorithm by Yasuo Matsuyama is an exact generalization of the log-EM algorithm. No computation of gradient or Hessian matrix is needed. The α -EM shows faster convergence than the log-EM algorithm by choosing an appropriate α . The α -EM algorithm leads to a faster version of the Hidden Markov model estimation algorithm α -HMM. [24]

83.10 Relation to variational Bayes methods

EM is a partially non-Bayesian, maximum likelihood method. Its final result gives a probability distribution over the latent variables (in the Bayesian style) together with a point estimate for θ (either a maximum likelihood estimate or a posterior mode). We may want a fully Bayesian version of this, giving a probability distribution over θ as well as the latent variables. In fact the Bayesian approach to inference is simply to treat θ as another latent variable. In this paradigm, the distinction between the E and M steps disappears. If we use the factorized Q approximation as described above (variational Bayes), we may iterate over each latent variable (now including θ) and optimize them one at a time. There are now k steps per iteration, where k is the number of latent variables. For graphical models this is easy to do as each variable's new Q depends only on its Markov blanket, so local message passing can be used for efficient inference.

83.11 Geometric interpretation

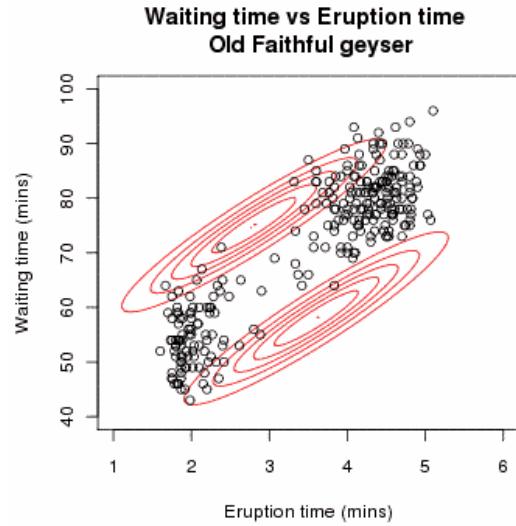
For more details on this topic, see [Information geometry](#).

In [information geometry](#), the E step and the M step are interpreted as projections under dual affine connections, called the e-connection and the m-connection; the Kullback–Leibler divergence can also be understood in these terms.

83.12 Examples

83.12.1 Gaussian mixture

Let $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$ be a sample of n independent observations from a [mixture](#) of two [multivariate normal distributions](#) of dimension d , and let $\mathbf{z} = (z_1, z_2, \dots, z_n)$ be the latent variables that determine the component from which the observation originates.^[16]



An animation demonstrating the EM algorithm fitting a two component Gaussian [mixture model](#) to the [Old Faithful](#) dataset. The algorithm steps through from a random initialization to convergence.

$$X_i | (Z_i = 1) \sim \mathcal{N}_d(\boldsymbol{\mu}_1, \Sigma_1) \text{ and } X_i | (Z_i = 2) \sim \mathcal{N}_d(\boldsymbol{\mu}_2, \Sigma_2)$$

where

$$\begin{aligned} P(Z_i = 1) &= \tau_1 \text{ and } P(Z_i = 2) = \tau_2 = \\ &1 - \tau_1 \end{aligned}$$

The aim is to estimate the unknown parameters representing the “mixing” value between the Gaussians and the means and covariances of each:

$$\theta = (\boldsymbol{\tau}, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \Sigma_1, \Sigma_2)$$

where the incomplete-data likelihood function is

$$L(\theta; \mathbf{x}, \mathbf{z}) = P(\mathbf{x}, \mathbf{z} | \theta) = \prod_{i=1}^n \sum_{j=1}^2 \tau_j f(\mathbf{x}_i; \boldsymbol{\mu}_j, \Sigma_j)$$

and the complete-data likelihood function is

$$L(\theta; \mathbf{x}, \mathbf{z}) = P(\mathbf{x}, \mathbf{z} | \theta) = \prod_{i=1}^n \sum_{j=1}^2 \mathbb{I}(z_i = j) f(\mathbf{x}_i; \boldsymbol{\mu}_j, \Sigma_j)$$

or

$$L(\theta; \mathbf{x}, \mathbf{z}) = \exp \left\{ \sum_{i=1}^n \sum_{j=1}^2 \mathbb{I}(z_i = j) \left[-\frac{1}{2} \log |\Sigma_j| - \frac{1}{2} (\mathbf{x}_i - \boldsymbol{\mu}_j)^\top \Sigma_j^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_j) \right] \right\}$$

where \mathbb{I} is an indicator function and f is the probability density function of a multivariate normal.

To see the last equality, note that for each i all indicators $\mathbb{I}(z_i = j)$ are equal to zero, except for one which is equal to one. The inner sum thus reduces to a single term.

E step

Given our current estimate of the parameters $\theta^{(t)}$, the conditional distribution of the Z_i is determined by Bayes theorem to be the proportional height of the normal density weighted by τ :

$$T_{j,i}^{(t)} := P(Z_i = j | X_i = \mathbf{x}_i; \theta^{(t)}) = \frac{\tau_j^{(t)} f(\mathbf{x}_i; \boldsymbol{\mu}_j^{(t)}, \Sigma_j^{(t)})}{\tau_1^{(t)} f(\mathbf{x}_i; \boldsymbol{\mu}_1^{(t)}, \Sigma_1^{(t)}) + \tau_2^{(t)} f(\mathbf{x}_i; \boldsymbol{\mu}_2^{(t)}, \Sigma_2^{(t)})}$$

This has the same form as a weighted MLE for a normal distribution, so

These are called the “membership probabilities” which are normally considered the output of the E step (although this is not the Q function of below).

Note that this E step corresponds with the following function for Q:

$$\begin{aligned} Q(\theta | \theta^{(t)}) &= E[\log L(\theta; \mathbf{x}, \mathbf{Z})] \\ &= E[\log \prod_{i=1}^n L(\theta; \mathbf{x}_i, \mathbf{z}_i)] \\ &= E[\sum_{i=1}^n \log L(\theta; \mathbf{x}_i, \mathbf{z}_i)] \\ &= \sum_{i=1}^n E[\log L(\theta; \mathbf{x}_i, \mathbf{z}_i)] \\ &= \sum_{i=1}^n \sum_{j=1}^2 T_{j,i}^{(t)} [\log \tau_j - \frac{1}{2} \log |\Sigma_j| - \frac{1}{2} (\mathbf{x}_i - \boldsymbol{\mu}_j^{(t)})^\top \Sigma_j^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_j^{(t)})] \end{aligned}$$

This does not need to be calculated, because in the M step we only require the terms depending on τ when we maximize for τ , or only the terms depending on $\boldsymbol{\mu}$ if we maximize for $\boldsymbol{\mu}$.

M step

The fact that $Q(\theta | \theta^{(t)})$ is quadratic in form means that determining the maximizing values of θ is relatively straightforward. Note that τ , $(\boldsymbol{\mu}_1, \Sigma_1)$ and $(\boldsymbol{\mu}_2, \Sigma_2)$ may all be maximized independently since they all appear in separate linear terms.

To begin, consider τ , which has the constraint $\tau_1 + \tau_2 = 1$:

$$\begin{aligned} \tau^{(t+1)} &= \arg \max_{\tau} Q(\theta | \theta^{(t)}) \\ &= \arg \max_{\tau} \left\{ \left[\sum_{i=1}^n T_{1,i}^{(t)} \right] \log \tau_1 + \left[\sum_{i=1}^n T_{2,i}^{(t)} \right] \log \tau_2 \right\} \end{aligned}$$

This has the same form as the MLE for the binomial distribution, so

$$\tau_j^{(t+1)} = \frac{\sum_{i=1}^n T_{j,i}^{(t)}}{\sum_{i=1}^n (T_{1,i}^{(t)} + T_{2,i}^{(t)})} = \frac{1}{n} \sum_{i=1}^n T_{j,i}^{(t)}$$

For the next estimates of $(\boldsymbol{\mu}_1, \Sigma_1)$:

$$(\boldsymbol{\mu}_1^{(t+1)}, \Sigma_1^{(t+1)}) = \arg \max_{\boldsymbol{\mu}_1, \Sigma_1} Q(\theta | \theta^{(t)})$$

$$= \arg \max_{\boldsymbol{\mu}_1, \Sigma_1} \sum_{i=1}^n T_{1,i}^{(t)} \left\{ -\frac{1}{2} \log |\Sigma_1| - \frac{1}{2} (\mathbf{x}_i - \boldsymbol{\mu}_1)^\top \Sigma_1^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_1) \right\}$$

$$\begin{aligned} \boldsymbol{\mu}_1^{(t+1)} &= \frac{\sum_{i=1}^n T_{1,i}^{(t)} \mathbf{x}_i}{\sum_{i=1}^n T_{1,i}^{(t)}} \quad \text{and} \quad \Sigma_1^{(t+1)} = \\ &\frac{\sum_{i=1}^n T_{1,i}^{(t)} (\mathbf{x}_i - \boldsymbol{\mu}_1^{(t+1)}) (\mathbf{x}_i - \boldsymbol{\mu}_1^{(t+1)})^\top}{\sum_{i=1}^n T_{1,i}^{(t)}} \end{aligned}$$

and, by symmetry

$$\begin{aligned} \boldsymbol{\mu}_2^{(t+1)} &= \frac{\sum_{i=1}^n T_{2,i}^{(t)} \mathbf{x}_i}{\sum_{i=1}^n T_{2,i}^{(t)}} \quad \text{and} \quad \Sigma_2^{(t+1)} = \\ &\frac{\sum_{i=1}^n T_{2,i}^{(t)} (\mathbf{x}_i - \boldsymbol{\mu}_2^{(t+1)}) (\mathbf{x}_i - \boldsymbol{\mu}_2^{(t+1)})^\top}{\sum_{i=1}^n T_{2,i}^{(t)}}. \end{aligned}$$

Termination

Conclude the iterative process if $\log L(\theta^t; \mathbf{x}, \mathbf{Z}) \leq \log L(\theta^{(t-1)}; \mathbf{x}, \mathbf{Z}) + \epsilon$ for ϵ below some preset threshold.

Generalization

The algorithm illustrated above can be generalized for mixtures of more than two multivariate normal distributions.

83.12.2 Truncated and censored regression

The EM algorithm has been implemented in the case where there is an underlying linear regression model explaining the variation of some quantity, but where the values actually observed are censored or truncated versions of those represented in the model.^[25] Special cases of this model include censored or truncated observations from a single normal distribution.^[25]

83.13 See also

- Density estimation
- Total absorption spectroscopy
- The EM algorithm can be viewed as a special case of the majorize-minimization (MM) algorithm.^[26]

83.14 Further reading

- Robert Hogg, Joseph McKean and Allen Craig. *Introduction to Mathematical Statistics*. pp. 359–364. Upper Saddle River, NJ: Pearson Prentice Hall, 2005.
- The on-line textbook: Information Theory, Inference, and Learning Algorithms, by David J.C. MacKay includes simple examples of the EM algorithm such as clustering using the soft k -means algorithm, and emphasizes the variational view of the EM algorithm, as described in Chapter 33.7 of version 7.2 (fourth edition).
- Dellaert, Frank. “The Expectation Maximization Algorithm”. CiteSeerX: 10.1.1.9.9735, gives an easier explanation of EM algorithm in terms of lowerbound maximization.
- Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Springer. ISBN 0-387-31073-8.
- M. R. Gupta and Y. Chen (2010). *Theory and Use of the EM Algorithm*. doi:10.1561/2000000034. A well-written short book on EM, including detailed derivation of EM for GMMs, HMMs, and Dirichlet.
- Bilmes, Jeff. “A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models”. CiteSeerX: 10.1.1.28.613, includes a simplified derivation of the EM equations for Gaussian Mixtures and Gaussian Mixture Hidden Markov Models.
- Variational Algorithms for Approximate Bayesian Inference, by M. J. Beal includes comparisons of EM to Variational Bayesian EM and derivations of several models including Variational Bayesian HMMs (chapters).
- The Expectation Maximization Algorithm: A short tutorial, A self-contained derivation of the EM Algorithm by Sean Borman.
- The EM Algorithm, by Xiaojin Zhu.
- EM algorithm and variants: an informal tutorial by Alexis Roche. A concise and very clear description of EM and many interesting variants.

- Einicke, G.A. (2012). *Smoothing, Filtering and Prediction: Estimating the Past, Present and Future*. Rijeka, Croatia: Intech. ISBN 978-953-307-752-9.

83.15 References

- [1] Dempster, A.P.; Laird, N.M.; Rubin, D.B. (1977). “Maximum Likelihood from Incomplete Data via the EM Algorithm”. *Journal of the Royal Statistical Society, Series B* **39** (1): 1–38. JSTOR 2984875. MR 0501537.
- [2] Sundberg, Rolf (1974). “Maximum likelihood theory for incomplete data from an exponential family”. *Scandinavian Journal of Statistics* **1** (2): 49–58. JSTOR 4615553. MR 381110.
- [3] Rolf Sundberg. 1971. *Maximum likelihood theory and applications for distributions generated when observing a function of an exponential family variable*. Dissertation, Institute for Mathematical Statistics, Stockholm University.
- [4] Sundberg, Rolf (1976). “An iterative method for solution of the likelihood equations for incomplete data from exponential families”. *Communications in Statistics – Simulation and Computation* **5** (1): 55–64. doi:10.1080/03610917608812007. MR 443190.
- [5] See the acknowledgement by Dempster, Laird and Rubin on pages 3, 5 and 11.
- [6] G. Kulldorff. 1961. *Contributions to the theory of estimation from grouped and partially grouped samples*. Almqvist & Wiksell.
- [7] Anders Martin-Löf. 1963. “Utvärdering av livslängder i subnanosekundsområdet” (“Evaluation of sub-nanosecond lifetimes”). (“Sundberg formula”)
- [8] Per Martin-Löf. 1966. *Statistics from the point of view of statistical mechanics*. Lecture notes, Mathematical Institute, Aarhus University. (“Sundberg formula” credited to Anders Martin-Löf).
- [9] Per Martin-Löf. 1970. *Statistika Modeller (Statistical Models): Anteckningar från seminarier läsåret 1969–1970 (Notes from seminars in the academic year 1969–1970), with the assistance of Rolf Sundberg*. Stockholm University. (“Sundberg formula”)
- [10] Martin-Löf, P. The notion of redundancy and its use as a quantitative measure of the deviation between a statistical hypothesis and a set of observational data. With a discussion by F. Abildgård, A. P. Dempster, D. Basu, D. R. Cox, A. W. F. Edwards, D. A. Sprott, G. A. Barnard, O. Barndorff-Nielsen, J. D. Kalbfleisch and G. Rasch and a reply by the author. *Proceedings of Conference on Foundational Questions in Statistical Inference* (Aarhus, 1973), pp. 1–42. Memoirs, No. 1, Dept. Theoret. Statist., Inst. Math., Univ. Aarhus, Aarhus, 1974.
- [11] Martin-Löf, Per The notion of redundancy and its use as a quantitative measure of the discrepancy between a statistical hypothesis and a set of observational data. *Scand. J. Statist.* **1** (1974), no. 1, 3–18.

- [12] Wu, C. F. Jeff (1983). “On the Convergence Properties of the EM Algorithm”. *The Annals of Statistics* (Institute of Mathematical Statistics) **11** (1): 95–103. Retrieved 11 December 2014.
- [13] Wu, C. F. Jeff (Mar 1983). “On the Convergence Properties of the EM Algorithm”. *Annals of Statistics* **11** (1): 95–103. doi:10.1214/aos/1176346060. JSTOR 2240463. MR 684867.
- [14] Little, Roderick J.A.; Rubin, Donald B. (1987). *Statistical Analysis with Missing Data*. Wiley Series in Probability and Mathematical Statistics. New York: John Wiley & Sons. pp. 134–136. ISBN 0-471-80254-9.
- [15] Neal, Radford; Hinton, Geoffrey (1999). Michael I. Jordan, ed. “A view of the EM algorithm that justifies incremental, sparse, and other variants”. *Learning in Graphical Models* (Cambridge, MA: MIT Press): 355–368. ISBN 0-262-60032-3. Retrieved 2009-03-22.
- [16] Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome (2001). “8.5 The EM algorithm”. *The Elements of Statistical Learning*. New York: Springer. pp. 236–243. ISBN 0-387-95284-5.
- [17] Einicke, G.A.; Malos, J.T.; Reid, D.C.; Hainsworth, D.W. (January 2009). “Riccati Equation and EM Algorithm Convergence for Inertial Navigation Alignment”. *IEEE Trans. Signal Processing* **57** (1): 370–375. doi:10.1109/TSP.2008.2007090.
- [18] Einicke, G.A.; Falco, G.; Malos, J.T. (May 2010). “EM Algorithm State Matrix Estimation for Navigation”. *IEEE Signal Processing Letters* **17** (5): 437–440. Bibcode:2010ISPL...17..437E. doi:10.1109/LSP.2010.2043151.
- [19] Einicke, G.A.; Falco, G.; Dunn, M.T.; Reid, D.C. (May 2012). “Iterative Smoother-Based Variance Estimation”. *IEEE Signal Processing Letters* **19** (5): 275–278. Bibcode:2012ISPL...19..275E. doi:10.1109/LSP.2012.2190278.
- [20] Jamshidian, Mortaza; Jennrich, Robert I. (1997). “Acceleration of the EM Algorithm by using Quasi-Newton Methods”. *Journal of the Royal Statistical Society, Series B* **59** (2): 569–587. doi:10.1111/1467-9868.00083. MR 1452026.
- [21] Meng, Xiao-Li; Rubin, Donald B. (1993). “Maximum likelihood estimation via the ECM algorithm: A general framework”. *Biometrika* **80** (2): 267–278. doi:10.1093/biomet/80.2.267. MR 1243503.
- [22] Hunter DR and Lange K (2004), A Tutorial on MM Algorithms, *The American Statistician*, 58: 30-37
- [23] Matsuyama, Yasuo (2003). “The α -EM algorithm: Surrogate likelihood maximization using α -logarithmic information measures”. *IEEE Transactions on Information Theory* **49** (3): 692–706. doi:10.1109/TIT.2002.808105.
- [24] Matsuyama, Yasuo (2011). “Hidden Markov model estimation based on alpha-EM algorithm: Discrete and continuous alpha-HMMs”. *International Joint Conference on Neural Networks*: 808–816.
- [25] Wolynetz, M.S. (1979). “Maximum likelihood estimation in a linear model from confined and censored normal data”. *Journal of the Royal Statistical Society, Series C* **28** (2): 195–206.
- [26] Lange, Kenneth. “The MM Algorithm”.

83.16 External links

- Various 1D, 2D and 3D demonstrations of EM together with Mixture Modeling are provided as part of the paired SOCR activities and applets. These applets and activities show empirically the properties of the EM algorithm for parameter estimation in diverse settings.
- k-MLE: A fast algorithm for learning statistical mixture models
- Class hierarchy in C++ (GPL) including Gaussian Mixtures
- Fast and clean C implementation of the Expectation Maximization (EM) algorithm for estimating Gaussian Mixture Models (GMMs).

Chapter 84

Extremal optimization

Extremal optimization (EO) is an optimization heuristic inspired by the Bak–Sneppen model of self-organized criticality from the field of statistical physics. This heuristic was designed initially to address combinatorial optimization problems such as the travelling salesman problem and spin glasses, although the technique has been demonstrated to function in optimization domains.

84.1 Relation to self-organized criticality

Self-organized criticality (SOC) is a statistical physics concept to describe a class of dynamical systems that have a critical point as an attractor. Specifically, these are non-equilibrium systems that evolve through avalanches of change and dissipations that reach up to the highest scales of the system. SOC is said to govern the dynamics behind some natural systems that have these burst-like phenomena including landscape formation, earthquakes, evolution, and the granular dynamics of rice and sand piles. Of special interest here is the Bak–Sneppen model of SOC, which is able to describe evolution via punctuated equilibrium (extinction events) – thus modelling evolution as a self-organised critical process.

84.2 Relation to computational complexity

Another piece in the puzzle is work on computational complexity, specifically that critical points have been shown to exist in NP-complete problems, where near-optimum solutions are widely dispersed and separated by barriers in the search space causing local search algorithms to get stuck or severely hampered. It was the evolutionary self-organised criticality model by Bak and Sneppen and the observation of critical points in combinatorial optimisation problems that lead to the development of Extremal Optimization by Stefan Boettcher and Allon Percus.

84.3 The technique

EO was designed as a local search algorithm for combinatorial optimization problems. Unlike genetic algorithms, which work with a population of candidate solutions, EO evolves a single solution and makes local modifications to the worst components. This requires that a suitable representation be selected which permits individual solution components to be assigned a quality measure (“fitness”). This differs from holistic approaches such as ant colony optimization and evolutionary computation that assign equal-fitness to all components of a solution based upon their collective evaluation against an objective function. The algorithm is initialized with an initial solution, which can be constructed randomly, or derived from another search process.

The technique is a fine-grained search, and superficially resembles a hill climbing (local search) technique. A more detailed examination reveals some interesting principles, which may have applicability and even some similarity to broader population-based approaches (evolutionary computation and artificial immune system). The governing principle behind this algorithm is that of improvement through selectively removing low-quality components and replacing them with a randomly selected component. This is obviously at odds with genetic algorithms, the quintessential evolutionary computation algorithm that selects good solutions in an attempt to make better solutions.

The resulting dynamics of this simple principle is firstly a robust hill climbing search behaviour, and secondly a diversity mechanism that resembles that of multiple-restart search. Graphing holistic solution quality over time (algorithm iterations) shows periods of improvement followed by quality crashes (avalanche) very much in the manner as described by punctuated equilibrium. It is these crashes or dramatic jumps in the search space that permit the algorithm to escape local optima and differentiate this approach from other local search procedures. Although such punctuated-equilibrium behaviour can be “designed” or “hard-coded”, it should be stressed that this is an emergent effect of the negative-component-selection principle fundamental to the algorithm.

EO has primarily been applied to combinatorial problems such as graph partitioning and the travelling salesman problem, as well as problems from statistical physics such as spin glasses.

84.4 Variations on the theme and applications

Generalised extremal optimization (GEO) was developed to operate on bit strings where component quality is determined by the absolute rate of change of the bit, or the bits contribution to holistic solution quality. This work includes application to standard function optimisation problems as well as engineering problem domains. Another similar extension to EO is Continuous Extremal Optimization (CEO).

EO has been applied to image rasterization as well as used as a local search after using ant colony optimization. EO has been used to identify structures in complex networks. EO has been used on a multiple target tracking problem. Finally, some work has been done on investigating the probability distribution used to control selection.

84.5 References

- Per Bak, Chao Tang, and Kurt Wiesenfeld, “Self-organized criticality: An explanation of the 1/f noise”, *Physical Review Letters* **59**, 381–384 (1987)
- Per Bak and Kim Sneppen, “Punctuated equilibrium and criticality in a simple model of evolution”, *Physical Review Letters* **71**, 4083–4086 (1993)
- P Cheeseman, B Kanefsky, WM Taylor, “Where the really hard problems are”, *Proceedings of the 12th IJCAI*, (1991)
- G Istrate, “Computational complexity and phase transitions”, *Proceedings. 15th Annual IEEE Conference on Computational Complexity*, 104–115 (2000)
- Stefan Boettcher, Allon G. Percus, “Extremal Optimization: Methods derived from Co-Evolution”, *Proceedings of the Genetic and Evolutionary Computation Conference* (1999)
- Stefan Boettcher, “Extremal optimization of graph partitioning at the percolation threshold”, *J. Phys. A: Math. Gen.* **32**, 5201–5211 (1999)
- S Boettcher, A Percus, “Nature’s Way of Optimizing”, *Artif. Intel.* **119**, (2000) 275
- S Boettcher, “Extremal Optimization – Heuristics via Co-Evolutionary Avalanches”, *Computing in Science & Engineering* **2**, pp. 75–82, 2000
- Stefan Boettcher and Allon G. Percus, “Optimization with Extremal Dynamics”, *Phys. Rev. Lett.* **86**, 5211–5214 (2001)
- Jesper Dall and Paolo Sibani, “Faster Monte Carlo Simulations at Low Temperatures. The Waiting Time Method”, *Computer Physics Communication* **141** (2001) 260–267
- Stefan Boettcher and Michelangelo Grigni, “Jamming Model for the Extremal Optimization Heuristic”, *J. Phys. A: Math. Gen.* **35**, 1109–1123 (2002)
- Souham Meshoul and Mohamed Batouche, “Robust Point Correspondence for Image Registration Using Optimization with Extremal Dynamics”, *Lecture Notes in Computer Science* **2449**, 330–337 (2002)
- Roberto N. Onody and Paulo A. de Castro, “Self-Organized Criticality, Optimization and Biodiversity”, *Int. J. Mod. Phys. C* **14**, 911–916 (2002)
- Stefan Boettcher and Allon G. Percus, “Extremal Optimization at the Phase Transition of the 3-Coloring Problem”, *Phys. Rev. E* **69**, 066703 (2004)
- A. Alan Middleton, “Improved extremal optimization for the Ising spin glass”, *Phys. Rev. E* **69**, 055701 (2004)
- F. Heilmann, K. H. Hoffmann and P. Salamon, “Best possible probability distribution over extremal optimization ranks”, *Europhys. Lett.* **66**, pp. 305–310 (2004)
- Pontus Svensson, “Extremal optimization for sensor report pre-processing”, *Proc SPIE* **5429**, 162–171 (2004)
- Tao Zhou, Wen-Jie Bai, Long-Jiu Cheng, Bing-Hong Wang, “Continuous extremal optimization for Lennard–Jones Clusters”, *Phys. Rev. E* **72**, 016702 (2004)
- Jordi Duch and Alex Arenas, “Community detection in complex networks using extremal optimization”, *Phys. Rev. E* **72**, 027104 (2005)
- E. Ahmed and M.F. Elettreby, “On combinatorial optimization motivated by biology”, *Applied Mathematics and Computation*, Volume 172, Issue 1, 1 January 2006, Pages 40–48

84.6 Web resources

- Stefan Boettcher’s home page which includes an excellent explanation of the technique and demonstration applets

- Allon Percus home page
- A good introduction to EO with lots of linked references
- a general summary on global optimization, including a short introduction to EO

84.7 See also

- Simulated Annealing
- Genetic Algorithm

Chapter 85

Fernandez's method

Fernandez method is a method which is used in the Multiprocessing scheduling algorithm. It is also represented by FB. It is actually used to improve the quality of the lower bounding schemes which are adopted by Branch and Bound algorithms for solving Multiprocessor scheduling problem. Fernandez problem derives a better lower bound than HF, and propose a quadratic-time algorithm from calculating the bound. It is known that a straightforward calculation of FB takes $O(n^3)$ time, since it must examine $O(n^2)$ combinations each of which takes $O(n)$ time in the worst case.

85.1 Further reading

- *A Comparison of List Scheduling for Parallel Processing Systems*

Chapter 86

Firefly algorithm

The **firefly algorithm (FA)** is a metaheuristic algorithm, inspired by the flashing behaviour of fireflies. The primary purpose for a firefly's flash is to act as a signal system to attract other fireflies. Xin-She Yang formulated this firefly algorithm by assuming:^[1]

1. All fireflies are unisexual, so that one firefly will be attracted to all other fireflies;
2. Attractiveness is proportional to their brightness, and for any two fireflies, the less bright one will be attracted by (and thus move to) the brighter one; however, the brightness can decrease as their distance increases;
3. If there are no fireflies brighter than a given firefly, it will move randomly.

The brightness should be associated with the objective function.

Firefly algorithm is a nature-inspired metaheuristic optimization algorithm.

86.1 Algorithm description

The pseudo code can be summarized as:

```
Begin 1) Objective function: f(x), x = (x1, x2, ..., xd) ; 2) Generate an initial population of fireflies xi (i = 1, 2, ..., n) ; 3) Formulate light intensity I so that it is associated with f(x) (for example, for maximization problems, I ∝ f(x) or simply I = f(x) ; 4) Define absorption coefficient γ While (t < MaxGeneration) for i = 1 : n (all n fireflies) for j = 1 : n (n fireflies) if (Ij > Ii), move firefly i towards j; end if Vary attractiveness with distance r via exp(-γ r) ; Evaluate new solutions and update light intensity; end for j end for i Rank fireflies and find the current best; end while Post-processing the results and visualization; end
```

The main update formula for any pair of two fireflies x_i and x_j is

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \beta \exp[-\gamma r_{ij}^2] (\mathbf{x}_j^t - \mathbf{x}_i^t) + \alpha_t \epsilon_t$$

where α_t is a parameter controlling the step size, while ϵ_t is a vector drawn from a Gaussian or other distribution.

It can be shown that the limiting case $\gamma \rightarrow 0$ corresponds to the standard Particle Swarm Optimization (PSO). In fact, if the inner loop (for j) is removed and the brightness I_j is replaced by the current global best g^* , then FA essentially becomes the standard PSO.

86.2 Implementation Guides

The γ should be related to the scales of design variables. Ideally, the β term should be order one, which requires that γ should be linked with scales. For example, one possible choice is to use $\gamma = 1/\sqrt{L}$ where L is the average scale of the problem. In case of scales vary significantly, γ can be considered as a vector to suit different scales in different dimensions. Similarly, α_t should also be linked with scales. For example, $\alpha_t \leftarrow 0.01L\alpha_t$. It is worth pointing out the above description does not include the randomness reduction. In fact, in actual implementation by most researchers, the motion of the fireflies is gradually reduced by an annealing-like randomness reduction via $\alpha = \alpha_0\delta^t$ where $0 < \delta < 1$ (e.g. $\delta = 0.97$).^[2] In some difficult problem, it may be helpful if you increase α_t at some stages, then reduce it when necessary. This non-monotonic variation of α_t will enable the algorithm to escape any local optima when in the unlikely case it might get stuck if randomness is reduced too quickly.

Parametric studies show that n (number of fireflies) should be about 15 to 40 for most problems.^[3] A python implementation is also available, though with limited functionalities.^[4]

Recent studies shows that the firefly algorithm is very efficient,^[5] and could outperform other metaheuristic algorithms including particle swarm optimization.^[6] Most metaheuristic algorithms may have difficulty in dealing with stochastic test functions, and it seems that firefly algorithm can deal with stochastic test functions^[7]

very efficiently. In addition, FA is also better for dealing with noisy optimization problems with ease of implementation.^{[8][9]}

Chatterjee et al.^[10] showed that the firefly algorithm can be superior to particle swarm optimization in their applications, the effectiveness of the firefly algorithm was further tested in later studies. In addition, firefly algorithm can efficiently solve non-convex problems with complex nonlinear constraints.^{[11][12]} Further improvement on the performance is also possible with promising results.^{[13][14]}

86.3 Variants of Firefly Algorithm

A recent, comprehensive review showed that the firefly algorithm and its variants have been used in almost area of science^[15] There are more than twenty variants:

86.3.1 Discrete Firefly Algorithm (DFA)

A discrete version of Firefly Algorithm, namely, Discrete Firefly Algorithm (DFA) proposed recently by M. K. Sayadi, R. Ramezanian and N. Ghaffari-Nasab can efficiently solve NP-hard scheduling problems.^[16] DFA outperforms existing algorithms such as the ant colony algorithm.

For image segmentation, the FA-based method is far more efficient to Otsu's method and recursive Otsu.^[17] Meanwhile, a good implementation of a discrete firefly algorithm for QAP problems has been carried out by Durkota.^[18]

86.3.2 Multiobjective FA

An important study of FA was carried out by Apostolopoulos and Vlachos,^[19] which provides a detailed background and analysis over a wide range of test problems including multiobjective load dispatch problem.

86.3.3 Lagrangian FA

An interesting, Lagrangian firefly algorithm is proposed to solve power system optimization unit commitment problems.^[20]

86.3.4 Chaotic FA

A chaotic firefly algorithm (CFA) was developed and found to outperform the previously best-known solutions available.^[21]

86.3.5 Hybrid Algorithms

A hybrid intelligent scheme has been developed by combining the firefly algorithm with the ant colony optimization.^[22]

86.3.6 Firefly Algorithm Based Memetic Algorithm

A firefly algorithm (FA) based memetic algorithm (FA-MA) is proposed to appropriately determine the parameters of SVR forecasting model for electricity load forecasting. In the proposed FA-MA algorithm, the FA algorithm is applied to explore the solution space, and the pattern search is used to conduct individual learning and thus enhance the exploitation of FA.^[23]

86.3.7 Parallel Firefly Algorithm with Predation (pFAP)

An implementation for shared memory environments with the addition of a predation mechanism that helps the method to escape local optimum.^[24]

86.3.8 Modified Firefly Algorithm

Many variants and modifications are done to increase its performance. A particular example will be modified firefly algorithm by Tilahun and Ong.,^[25] in which the updating process of the brightest firefly is modified to keep the best result throughout the iterations.

86.4 Applications

86.4.1 Digital Image Compression and Image Processing

Very recently, an FF-LBG algorithm for vector quantization of digital image compression was based on the firefly algorithm, which proves to be faster than other algorithms such as PSO-LBG and HBMO-LBG (particle swarm optimization and honey-bee mating optimization; variations on the Linde–Buzo–Gray algorithm).^{[26] [27]} For minimum cross entropy thresholding, firefly-based algorithm uses the least computation time^[28] Also, for gel electrophoresis images, FA-based method is very efficient.^[29]

86.4.2 Eigenvalue optimization

Eigenvalue optimization of isospectral systems has solved by FA and multiple optimum points have been found efficiently.^[30]

86.4.3 Nanoelectronic Integrated Circuit and System Design

The multiobjective firefly algorithm (MOFA) has been used for the design optimization of a 90 nm CMOS based operational amplifier (OP-AMP) which could perform simultaneous power minimization and slew rate maximization within 500 iterations.^[31]

86.4.4 Feature selection and fault detection

Feature selection can be also carried out successfully using firefly algorithm.^[32] Real-time fault identification in large systems becomes viable, based on the recent work on fault identification with binary adaptive firefly optimization.^[33]

86.4.5 Antenna Design

Firefly algorithms outperforms ABC for optimal design of linear array of isotropic sources^[34] and digital controllable array antenna.^[35] It has found applications in synthesis of satellite footprint patterns as well.^[36]

86.4.6 Structural Design

For mixed-variable problems, many optimization algorithms may struggle. However, firefly algorithm can efficiently solve optimization problems with mixed variables.^[37]

86.4.7 Scheduling and TSP

Firefly-based algorithms for scheduling task graphs and job shop scheduling requires less computing than all other metaheuristics.^{[38][39]} A binary firefly algorithm has been developed to tackle the knapsack cryptosystem efficiently.^[40] Recently, an evolutionary discrete FA has been developed for solving travelling salesman problems^[41] Further improvement in performance can be obtained by using preferential directions in firefly movements.

86.4.8 Semantic Web Composition

A hybrid FA has been developed by Pop et al. for selecting optimal solution in semantic web service composition.^[42]

86.4.9 Chemical Phase equilibrium

For phase equilibrium calculations and stability analysis, FA was found to be the most reliable compared with other techniques.^[43]

86.4.10 Clustering

Performance study for clustering also suggested that firefly algorithm is very efficient.^[44]

86.4.11 Dynamic Problems

Firefly algorithm can solve optimization problems in dynamic environments very efficiently.^{[45][46]}

86.4.12 Rigid Image Registration Problems

Firefly algorithm can solve the rigid image registration problems more efficient than genetic algorithm, particle swarm optimization, and artificial bee colony^[47]

86.4.13 Protein Structure Prediction

Prediction of protein structures is NP-hard, and a recent study by Maher et al.^[48] shows that firefly-based methods can speed up the predictions. Firefly algorithm can solve two dimensional HP model. In their experiment, they took 14 sequences of different chain lengths from 18 to 100 as the dataset and compared the FA with standard genetic algorithm and immune genetic algorithm. The averaged energy convergence results show that FA achieves the lowest values.^[49]

86.4.14 Parameter Optimization of SVM

Firefly algorithm (FA) is applied to determine the paraemters of MSVR (Multiple-output support vector regression) in interval-valued stock price index forecasting.^[50]

Meanwhile, a firefly algorithm (FA) based memetic algorithm (FA-MA) is proposed to appropriately determine the parameters of SVR forecasting model for electricity load forecasting. In the proposed FA-MA algorithm, the FA algorithm is applied to explore the solution space, and the pattern search is used to conduct individual learning and thus enhance the exploitation of FA.^[51]

86.4.15 IK-FA, Solving Inverse Kinematics using FA

FA, heuristic is used as inverse kinematics solver. The proposal is called IK-FA, for inverse Kinematics using Firefly Algorithm. Inverse kinematic consists in finding a valuable joints solution allowing achieving a specific end segment position. The proposed method used a forward kinematics model, the FA heuristic, a fitness function and a set of motions constraints, to solve inverse kinematics.^[52]

86.5 See also

- Evolutionary multi-modal optimization
- Glowworm swarm optimization (GSO)

86.6 References

- [1] Yang, X. S. (2008). *Nature-Inspired Metaheuristic Algorithms*. Frome: Luniver Press. ISBN 1-905986-10-6.
- [2] http://www.mathworks.com/matlabcentral/fileexchange/29693-firefly-algorithm/content/fa_mincon.m
- [3] A simple demo Matlab code is available
- [4] <https://code.google.com/p/csc6810project/>
- [5] Yang, X. S. (2009). “Firefly algorithms for multimodal optimization”. *Stochastic Algorithms: Foundations and Applications, SAGA 2009*. Lecture Notes in Computer Sciences **5792**. pp. 169–178. arXiv:1003.1466.
- [6] Lukasik, S.; Zak, S. (2009). *Firefly algorithm for continuous constrained optimization task*. *ICCCI 2009, Lecture Notes in Artificial Intelligence (Eds. N. T. Ngugen, R. Kowalczyk, S. M. Chen)* **5796**. pp. 97–100.
- [7] Yang, X.-S. (2010). “Firefly algorithm, stochastic test functions and design optimisation”. *Int. J. Bio-inspired Computation* **2** (2): 78–84. arXiv:1003.1409.
- [8] Chai-ead, N.; Aungkulanon, P.; Luangpaiboon, P. (2011). “Bees and firefly algorithms for noisy non-linear optimisation problems”. *Prof. Int. Multiconference of Engineers and Computer Scientists 2011* **2**: 1449–1454.
- [9] Aungkulanon, P.; Chai-ead, N.; Luangpaiboon, P. (2011). “Simulated manufacturing process improvement via particle swarm optimisation and firefly algorithms”. *Prof. Int. Multiconference of Engineers and Computer Scientists 2011* **2**: 1123–1128.
- [10] A. Chatterjee, G. K. Mahanti, and A. Chatterjee, Design of a fully digital controlled reconfigurable switched beam conconcentric ring array antenna using firefly and particle swarm optimization algorithm, Progress in Electromagnetic Research B, Vol. 36, 113-131(2012)
- [11] X. S. Yang, S. S. Hosseini, A. H. Gandomi, Firefly algorithm for solving non-convex economic dispatch problems with valve loading effect, Applied Soft Computing, Vol. 12(3), 1180-1186(2012)
- [12] A. Abdullah, S. Deris, M. S. Mohamad and S. Z. M. Hashim, A new hybrid firefly algorithm for complex and nonlinear problem, in: Distributed Computing and Artificial Intelligence, Advances in Intelligent and Soft Computing, 2012, Volume 151/2012, 673-680, doi:10.1007/978-3-642-28765-7_81
- [13] S. M. Farahani, A. A. Abshouri, B. Nasiri, M. R. Meybodi, Some hybrid models to improve firefly algorithm performance, Int. J. Artificial Intelligence, Vol. 8 S(12), 97-117 (2012)
- [14] B. Nasiri, M. R. Meybodi, Speciation-based firefly algorithm for optimization in dynamic environments, Int. J. Artificial Intelligence, Vol. 8 (S12), 118-132 (2012)
- [15] I. Fister, I. Fister Jr., X. S. Yang, J. Brest, A comprehensive review of firefly algorithms, Swarm and Evolutionary Computation, vol. 13, no. 1, pp. 34-46 (2013).
- [16] Sayadi, M. K.; Ramezanian, R.; Ghaffari-Nasab, N. (2010). “A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems”. *Int. J. of Industrial Engineering Computations* **1**: 1–10.
- [17] T. Hassanzadeh, H. Vojodi and A. M. E. Moghadam, An image segmentation approach based on maximum variance intra-cluster method and firefly algorithm, in: Proc. of 7th Int. Conf. on Natural Computation (ICNC), pp. 1817-1821 (2011).
- [18] K. Durkota, Implementation of a discrete firefly algorithm for the QAP problem within the sage framework, BSc thesis, Czech Technical University, (2011). <http://cyber.felk.cvut.cz/research/theses/papers/189.pdf>
- [19] Apostolopoulos, T.; Vlachos, A. (2011). “Application of the Firefly Algorithm for Solving the Economic Emissions Load Dispatch Problem”. *International Journal of Combinatorics* **2011**: Article ID 523806.
- [20] Rampriya B., Mahadevan K. and Kannan S., Unit commitment in deregulated power system using Lagrangian firefly algorithm, Proc. of IEEE Int. Conf. on Communication Control and Computing Technologies (ICCCCT), pp. 389-393 (2010).
- [21] L. dos Santos Coelho, D. L. de Andrade Bernert, V. C. Mariani, a chaotic firefly algorithm applied to reliability-redundancy optimization, in: 2011 IEEE Congress on Evolutionary Computation (CEC'11), pp. 517-521 (2011).
- [22] G. Giannakouris, V. Vassiliadis and G. Dounias, Experimental study on a hybrid nature-inspired algorithm for financial portfolio optimization, SETN 2010, LNAI 6040, pp. 101-111 (2010).
- [23] Zhongyi Hu, Yukun Bao, and Tao Xiong, Electricity Load Forecasting using Support Vector Regression with Memetic Algorithms, The Scientific World Journal, 2014, <http://www.hindawi.com/journals/tswj/aip/292575/>

- [24] E. F. P. Luz, H. F. Campos Velho, J. C. Becceneri, Firefly Algorithm with Predation: A parallel implementation applied to inverse heat conduction problem, in: Proc. of 10th World Congress on Computational Mechanics (WCCM 2012), (2012).
- [25] S. L. Tilahun, H. C. Ong, Modified Firefly Algorithm, Journal of Applied Mathematics, Volume 2012 (2012), Article ID 467631 .
- [26] Horng M.-H. and Jiang T. W., The codebook design of image vector quantization based on the firefly algorithm, in: Computational Collective Intelligence, Technologies and Applications, LNCS, Vol. 6423, pp. 438-447 (2010).
- [27] M.-H. Horng, vector quantization using the firefly algorithm for image compression, Expert Systems with Applications, Vol. 38, (article in press) 12 Aug. (2011).
- [28] M.-H. Horng and R.-J Liou, Multilevel minimum cross entropy threshold selection based on the firefly algorithm, Expert Systems with Applications, Vol. 38, Issue 12, 14805-14811 (2011).
- [29] M. H. M. Noor, A. R. Ahmad, Z. Hussain, K. A. Ahmad, A. R. Ainihayati, Multilevel thresholding of gel electrophoresis images using firefly algorithm, in: Proceedings of Control System, Computing and Engineering (ICC-SCE2011), pp. 18-21 (2011).
- [30] R. Dutta, R. Ganguli and V. Mani, Exploring isospectral spring-mass systems with firefly algorithm, Proc. Roy. Soc. A., Vol. 467, (2011)<http://rspa.royalsocietypublishing.org/content/early/2011/06/16/rspa.2011.0119.abstract>
- [31] G. Zheng, S. P. Mohanty, E. Kougiannos, and O. Okobiah, “iVAMS: Intelligent Metamodel-Integrated Verilog-AMS for Circuit-Accurate System-Level Mixed-Signal Design Exploration”, in Proceedings of the 24th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP), 2013, pp. 75–78.
- [32] H. Banati and M. Bajaj, Firefly based feature selection approach, Int. J. Computer Science Issues, vol. 8, No. 2, 473-480 (2011).
- [33] R. Falcon, M. Almeida and A. Nayak, Fault identification with binary adaptive fireflies in parallel and distributed systems, IEEE Congress on Evolutionary Computation, (2011).
- [34] B. Basu and G. K. Mahanti, Firefly and artificial bees colony algorithm for synthesis of scanned and broadside linear array antenna, Progress in Electromagnetic Research B., Vol. 32, 169-190 (2011).
- [35] A. Chatterjee, G. K. Mahanti, and A. Chatterjee, Design of a fully digital controlled reconfigurable switched beam conconcentric ring array antenna using firefly and particle swarm optimization algorithm, Progress in Electromagnetic Research B, Vol. 36, 113-131(2012)
- [36] Anirban Chatterjee, Gautam Kumar Mahanti and Gourab Ghatak, Synthesis of satellite footprint patterns from rectangular planar array antenna by using swarm-based optimization algorithms, Int. J. Satell. Commun. Network. 2014; 32:25–47
- [37] A. H. Gandomi, X. S. Yang, A. H. Alavi, Mixed variable structural optimization using firefly algorithm, Computers and Structures, Vol. 89, No. 23-24, pp. 2325-2336 (2011). doi:10.1016/j.compstruc.2011.08.002
- [38] U. Höning, A firefly algorithm-based approach for scheduling task graphs in homogeneous systems, Proceeding Informatics, doi:10.2316/P.2010.724-033, 724 (2010).
- [39] A. Khadwilard, S. Chansombat, T. Thepphakorn, P. Thapatsuwan, W. Chainat, P. Pongcharoen, Application of firefly algorithm and its parameter setting for job shop scheduling, First Symposium on Hands-On Research and Development, (2011).
- [40] S. Palit, S. Sinha, M. Molla, A. Khanra, M. Kule, A cryptanalytic attack on the knapsack cryptosystem using binary Firefly algorithm, in: 2nd Int. Conference on Computer and Communication Technology (ICCCT), 15-17 Sept 2011, India, pp. 428-432 (2011).
- [41] G. K. Jati and S. Suyanto, Evolutionary discrete firefly algorithm for travelling salesman problem, ICAIS2011, Lecture Notes in Artificial Intelligence (LNAI 6943), pp.393-403 (2011).
- [42] C. B. Pop, V. R. Chifu, I. Salomie, R. B. Baico, M. Dinsooreanu, G. Copil, A hybrid firefly-inspired approach for optimal semantic web service composition, in: Proc. of 2nd Workshop on Software Services: Cloud Computing and Applications, June, 2011.
- [43] S. E. Fateen, A. Bonilla-Petrociolet, G. P. Rangaiah, Evaluation of covariance matrix adaptation evolution strategy, shuffled complex evolution and firefly algorithms for phase stability, phase equilibrium and chemical equilibrium problems, Chemical Engineering Research and Design, May (2012). <http://dx.doi.org/10.1016/j.cherd.2012.04.011>
- [44] J. Senthilnath, S. N. Omkar and V. Mani, Clustering using firefly algorithm: Performance study, Swarm and Evolutionary Computation, June (2011). doi:10.1016/j.swevo.2011.06.003
- [45] S. M. Farahani, B. Nasiri and M. R. Meybodi, A multi-swarm based firefly algorithm in dynamic environments, Third Int. Conference on Signal Processing Systems (IC-SPS2011), Aug 27-28, Yantai, China, pp. 68-72 (2011)
- [46] A. A. Abshouri, M. R. Meybodi and A. Bakhtiyari, New firefly algorithm based on multiswarm and learning automata in dynamic environments, Third Int. Conference on Signal Processing Systems (IC-SPS2011), Aug 27-28, Yantai, China, pp. 73-77 (2011).
- [47] Yudong Zhang and Lenan Wu, A Novel Method for Rigid Image Registration based on Firefly Algorithm, International Journal of Research and Reviews in Soft and Intelligent Computing, vol.2, no.2, pp. 141-146 (2012).
- [48] B. Maher, A. Albrecht, M. Loomes, X. S. Yang, K. Steinhofel, A firefly-inspired method for protein structure prediction in lattice models, Biomolecules, vol. 4, no. 1, pp. 56-75 (2014).

- [49] Yudong, Zhang; Lenan, Wu; Shuihua, Wang (2013). “Solving Two-Dimensional HP model by Firefly Algorithm and Simplified Energy Function”. *Mathematical Problems in Engineering* **2013**. doi:10.1155/2013/398141.
- [50] Tao Xiong, Yukun Bao, Zhongyi Hu: Multiple-output support vector regression with a firefly algorithm for interval-valued stock price index forecasting. *Knowl.-Based Syst.* **55**: 87-100 (2014), <http://www.sciencedirect.com/science/article/pii/S0950705113003237>
- [51] Zhongyi Hu, Yukun Bao, and Tao Xiong, Electricity Load Forecasting using Support Vector Regression with Memetic Algorithms, *The Scientific World Journal*, 2014, <http://www.hindawi.com/journals/tswj/aip/292575/>
- [52] Rokbani, Nizar, et al. “IK-FA, Inverse Kinematics Using Firefly Algorithm with Application to Biped Gait Generation, International Conference on Control, Engineering & Information Technology (CEIT’14), Tunisia, 2014

86.7 External links

- Firefly Algorithm implemented in Python
- Firefly Algorithm in C/C++
- Firefly Algorithm in Matlab or Octave

Chapter 87

Fourier–Motzkin elimination

Fourier–Motzkin elimination, also known as the **FME method**, is a mathematical algorithm for eliminating variables from a system of linear inequalities. It can output real solutions.

The algorithm is named after Joseph Fourier and Theodore Motzkin.

87.1 Elimination

The elimination of a set of variables, say V , from a system of relations (here linear inequalities) refers to the creation of another system of the same sort, but without the variables in V , such that both systems have the same solutions over the remaining variables.

If all variables are eliminated from a system of linear inequalities, then one obtains a system of constant inequalities. It is then trivial to decide whether the resulting system is true or false. It is true if and only if the original system has solutions. As a consequence, elimination of all variables can be used to detect whether a system of inequalities has solutions or not.

Consider a system S of n inequalities with r variables x_1 to x_r , with x_r the variable to be eliminated. The linear inequalities in the system can be grouped into three classes depending on the sign (positive, negative or null) of the coefficient for x_r .

- those inequalities that are of the form $x_r \geq b_i - \sum_{k=1}^{r-1} a_{ik}x_k$; denote these by $x_r \geq A_j(x_1, \dots, x_{r-1})$, for j ranging from 1 to n_A where n_A is the number of such inequalities;
- those inequalities that are of the form $x_r \leq b_i - \sum_{k=1}^{r-1} a_{ik}x_k$; denote these by $x_r \leq B_j(x_1, \dots, x_{r-1})$, for j ranging from 1 to n_B where n_B is the number of such inequalities;
- those inequalities in which x_r plays no role, grouped into a single conjunction ϕ .

The original system is thus equivalent to

$$\max(A_1(x_1, \dots, x_{r-1}), \dots, A_{n_A}(x_1, \dots, x_{r-1})) \leq x_r \leq \min(B_1(x_1, \dots, x_{r-1}), \dots, B_{n_B}(x_1, \dots, x_{r-1}))$$

Elimination consists in producing a system equivalent to $\exists x_r S$. Obviously, this formula is equivalent to

$$\max(A_1(x_1, \dots, x_{r-1}), \dots, A_{n_A}(x_1, \dots, x_{r-1})) \leq \min(B_1(x_1, \dots, x_{r-1}), \dots, B_{n_B}(x_1, \dots, x_{r-1}))$$

The inequality

$$\max(A_1(x_1, \dots, x_{r-1}), \dots, A_{n_A}(x_1, \dots, x_{r-1})) \leq \min(B_1(x_1, \dots, x_{r-1}), \dots, B_{n_B}(x_1, \dots, x_{r-1}))$$

is equivalent to $n_A n_B$ inequalities $A_i(x_1, \dots, x_{r-1}) \leq B_j(x_1, \dots, x_{r-1})$, for $1 \leq i \leq n_A$ and $1 \leq j \leq n_B$.

We have therefore transformed the original system into another system where x_r is eliminated. Note that the output system has $(n - n_A - n_B) + n_A n_B$ inequalities. In particular, if $n_A = n_B = n/2$, then the number of output inequalities is $n^2/4$.

87.2 Complexity

Running an elimination step over n inequalities can result in at most $n^2/4$ inequalities in the output, thus running d successive steps can result in at most $4(n/4)^{2^d}$, a double exponential complexity. This is due to the algorithm producing many unnecessary constraints (constraints that are implied by other constraints). The number of necessary constraints grows as a single exponential.^[1] Unnecessary constraints may be detected using linear programming.

87.3 See also

- Real closed field: the cylindrical algebraic decomposition algorithm performs quantifier elimination over polynomial inequalities, not just linear.

87.4 References

[1] David Monniaux, *Quantifier elimination by lazy model elimination*, Computer Aided Verification (CAV) 2010.

- Fourier, Joseph (1827). "Histoire de l'Académie, partie mathématique (1824)". *Mémoires de l'Académie des sciences de l'Institut de France* 7. Gauthier-Villars.
- Schrijver, Alexander (1998). *Theory of Linear and Integer Programming*. John Wiley & sons. pp. 155–156. ISBN 0-471-98232-6.
- Keßler, Christoph W. "Parallel Fourier–Motzkin Elimination". *Universität Trier*. CiteSeerX: 10.1.1.54.657.
- Williams, H. P. (1986). "Fourier's Method of Linear Programming and its Dual". *American Mathematical Monthly* 93 (9): 681–695. doi:10.2307/2322281.

87.5 External links

- Lectures on Convex Sets, notes by Niels Lauritzen, at Aarhus University, March 2010.

Chapter 88

Generalized iterative scaling

In statistics, **generalized iterative scaling (GIS)** and **improved iterative scaling (IIS)** are two early algorithms used to fit log-linear models,^[1] notably multinomial logistic regression (MaxEnt) classifiers and extensions of it such as MaxEnt Markov models^[2] and conditional random fields. These algorithms have been largely surpassed by gradient-based methods such as L-BFGS^[3] and coordinate descent algorithms.^[4]

88.1 See also

- Expectation-maximization

88.2 References

- [1] Darroch, J.N. and Ratcliff, D. (1972). “Generalized iterative scaling for log-linear models”. *The Annals of Mathematical Statistics* (Institute of Mathematical Statistics) **43** (5): 1470–1480. doi:10.1214/aoms/1177692379.
- [2] McCallum, Andrew; Freitag, Dayne; Pereira, Fernando (2000). “Maximum Entropy Markov Models for Information Extraction and Segmentation”. *Proc. ICML 2000*. pp. 591–598.
- [3] Malouf (2002). *A comparison of algorithms for maximum entropy parameter estimation*. Sixth Conf. on Natural Language Learning (CoNLL). pp. 49–55. |first1=missing |last1= in Authors list (help)
- [4] Yu, Hsiang-Fu; Huang, Fang-Lan; Lin, Chih-Jen (2011). “Dual coordinate descent methods for logistic regression and maximum entropy models”. *Machine Learning* **85**: 41–75. doi:10.1007/s10994-010-5221-8.

Chapter 89

Genetic algorithm



The 2006 NASA ST5 spacecraft antenna. This complicated shape was found by an evolutionary computer design program to create the best radiation pattern.

In the field of artificial intelligence, a **genetic algorithm (GA)** is a search heuristic that mimics the process of natural selection. This heuristic (also sometimes called a metaheuristic) is routinely used to generate useful solutions to optimization and search problems.^[1] Genetic algorithms belong to the larger class of **evolutionary algorithms (EA)**, which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

Genetic algorithms find application in bioinformatics, phylogenetics, computational science, engineering, economics, chemistry, manufacturing, mathematics, physics, pharmacometrics and other fields.

89.1 Methodology

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.^[2]

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a *generation*. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

A typical genetic algorithm requires:

1. a genetic representation of the solution domain,
2. a fitness function to evaluate the solution domain.

A standard representation of each candidate solution is as an array of bits.^[2] Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in **genetic programming** and graph-form representations are explored in **evolutionary programming**; a mix of both linear chromosomes and trees is explored in **gene expression programming**.

Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

89.1.1 Initialization

The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Often, the initial population is generated randomly, allowing the entire range of possible solutions (the *search space*). Occasionally, the solutions may be “seeded” in areas where optimal solutions are likely to be found.

89.1.2 Selection

Main article: Selection (genetic algorithm)

During each successive generation, a proportion of the existing population is **selected** to breed a new generation. Individual solutions are selected through a *fitness-based* process, where **fitter** solutions (as measured by a **fitness function**) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as the former process may be very time-consuming.

The fitness function is defined over the genetic representation and measures the *quality* of the represented solution. The fitness function is always problem dependent. For instance, in the **knapsack problem** one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The *fitness* of the solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise.

In some problems, it is hard or even impossible to define the fitness expression; in these cases, a simulation may be used to determine the fitness function value of a phenotype (e.g. **computational fluid dynamics** is used to determine the air resistance of a vehicle whose shape is encoded as the phenotype), or even **interactive genetic algorithms** are used.

89.1.3 Genetic operators

Main articles: Crossover (genetic algorithm) and Mutation (genetic algorithm)

The next step is to generate a second generation population of solutions from those selected through a combination of **genetic operators**: **crossover** (also called **recombination**), and **mutation**.

For each new solution to be produced, a pair of “parent” solutions is selected for breeding from the pool selected previously. By producing a “child” solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its “parents”. New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated. Although reproduction methods that are based on the use of two parents are more “biology inspired”, some research^{[3][4]} suggests that more than two “parents” generate higher quality chromosomes.

These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions. These less fit solutions ensure genetic diversity within the genetic pool of the parents and therefore ensure the genetic diversity of the subsequent generation of children.

Opinion is divided over the importance of crossover versus mutation. There are many references in Fogel (2006) that support the importance of mutation-based search.

Although crossover and mutation are known as the main genetic operators, it is possible to use other operators such as regrouping, colonization-extinction, or migration in genetic algorithms.^[5]

It is worth tuning parameters such as the **mutation probability**, **crossover probability** and **population size** to find reasonable settings for the problem class being worked on. A very small mutation rate may lead to **genetic drift** (which is **non-ergodic** in nature). A recombination rate that is too high may lead to premature convergence of the genetic algorithm. A mutation rate that is too high may lead to loss of good solutions, unless **elitist selection** is employed.

89.1.4 Termination

This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached

- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of the above

89.2 The building block hypothesis

Genetic algorithms are simple to implement, but their behavior is difficult to understand. In particular it is difficult to understand why these algorithms frequently succeed at generating solutions of high fitness when applied to practical problems. The building block hypothesis (BBH) consists of:

1. A description of a heuristic that performs adaptation by identifying and recombining “building blocks”, i.e. low order, low defining-length **schemata** with above average fitness.
2. A hypothesis that a genetic algorithm performs adaptation by implicitly and efficiently implementing this heuristic.

Goldberg describes the heuristic as follows:

“Short, low order, and highly fit schemata are sampled, recombined [crossed over], and resampled to form strings of potentially higher fitness. In a way, by working with these particular schemata [the building blocks], we have reduced the complexity of our problem; instead of building high-performance strings by trying every conceivable combination, we construct better and better strings from the best partial solutions of past samplings.

“Because highly fit schemata of low defining length and low order play such an important role in the action of genetic algorithms, we have already given them a special name: building blocks. Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low-order, high-performance schemata, or building blocks.”^[6]

89.3 Limitations

There are limitations of the use of a genetic algorithm compared to alternative optimization algorithms:

- Repeated **fitness function** evaluation for complex problems is often the most prohibitive and limiting segment of artificial evolutionary algorithms. Finding the optimal solution to complex high-dimensional, multimodal problems often requires very expensive **fitness function** evaluations. In real world problems such as structural optimization problems, a single function evaluation may require several hours to several days of complete simulation. Typical optimization methods can not deal with such types of problem. In this case, it may be necessary to forgo an exact evaluation and use an **approximated fitness** that is computationally efficient. It is apparent that amalgamation of **approximate models** may be one of the most promising approaches to convincingly use GA to solve complex real life problems.

- Genetic algorithms do not scale well with complexity. That is, where the number of elements which are exposed to mutation is large there is often an exponential increase in search space size. This makes it extremely difficult to use the technique on problems such as designing an engine, a house or plane. In order to make such problems tractable to evolutionary search, they must be broken down into the simplest representation possible. Hence we typically see evolutionary algorithms encoding designs for fan blades instead of engines, building shapes instead of detailed construction plans, airfoils instead of whole aircraft designs. The second problem of complexity is the issue of how to protect parts that have evolved to represent good solutions from further destructive mutation, particularly when their fitness assessment requires them to combine well with other parts.
- The “better” solution is only in comparison to other solutions. As a result, the stop criterion is not clear in every problem.
- In many problems, GAs may have a tendency to converge towards local optima or even arbitrary points rather than the global optimum of the problem. This means that it does not “know how” to sacrifice short-term fitness to gain longer-term fitness. The likelihood of this occurring depends on the shape of the **fitness landscape**: certain problems may provide an easy ascent towards a global optimum, others may make it easier for the function to find the local optima. This problem may be alleviated by using a different fitness function, increasing the rate of mutation, or by using selection techniques that maintain a diverse population of solutions,^[7] although the **No Free Lunch theorem**^[8] proves that there is no general solution to this problem. A common technique to maintain diversity is to impose a “niche penalty”, wherein, any group of individuals of sufficient similarity (niche radius) have a penalty added, which will reduce the representation of that group in

subsequent generations, permitting other (less similar) individuals to be maintained in the population. This trick, however, may not be effective, depending on the landscape of the problem. Another possible technique would be to simply replace part of the population with randomly generated individuals, when most of the population is too similar to each other. Diversity is important in genetic algorithms (and [genetic programming](#)) because crossing over a homogeneous population does not yield new solutions. In [evolution strategies](#) and [evolutionary programming](#), diversity is not essential because of a greater reliance on mutation.

- Operating on dynamic data sets is difficult, as genomes begin to converge early on towards solutions which may no longer be valid for later data. Several methods have been proposed to remedy this by increasing genetic diversity somehow and preventing early convergence, either by increasing the probability of mutation when the solution quality drops (called *triggered hypermutation*), or by occasionally introducing entirely new, randomly generated elements into the gene pool (called *random immigrants*). Again, [evolution strategies](#) and [evolutionary programming](#) can be implemented with a so-called “comma strategy” in which parents are not maintained and new parents are selected only from offspring. This can be more effective on dynamic problems.
- GAs cannot effectively solve problems in which the only fitness measure is a single right/wrong measure (like [decision problems](#)), as there is no way to converge on the solution (no hill to climb). In these cases, a random search may find a solution as quickly as a GA. However, if the situation allows the success/failure trial to be repeated giving (possibly) different results, then the ratio of successes to failures provides a suitable fitness measure.
- For specific optimization problems and problem instances, other optimization algorithms may be more efficient than genetic algorithms in terms of speed of convergence. Alternative and complementary algorithms include [evolution strategies](#), [evolutionary programming](#), [simulated annealing](#), [Gaussian adaptation](#), [hill climbing](#), and [swarm intelligence](#) (e.g.: [ant colony optimization](#), [particle swarm optimization](#)) and methods based on [integer linear programming](#). The suitability of genetic algorithms is dependent on the amount of knowledge of the problem; well known problems often have better, more specialized approaches.

89.4 Variants

89.4.1 Chromosome representation

The simplest algorithm represents each chromosome as a [bit string](#). Typically, numeric parameters can be represented by [integers](#), though it is possible to use [floating point representations](#). The floating point representation is natural to [evolution strategies](#) and [evolutionary programming](#). The notion of real-valued genetic algorithms has been offered but is really a misnomer because it does not really represent the building block theory that was proposed by [John Henry Holland](#) in the 1970s. This theory is not without support though, based on theoretical and experimental results (see below). The basic algorithm performs crossover and mutation at the bit level. Other variants treat the chromosome as a list of numbers which are indexes into an instruction table, nodes in a [linked list](#), [hashes](#), [objects](#), or any other imaginable [data structure](#). Crossover and mutation are performed so as to respect data element boundaries. For most data types, specific variation operators can be designed. Different chromosomal data types seem to work better or worse for different specific problem domains.

When bit-string representations of integers are used, [Gray coding](#) is often employed. In this way, small changes in the integer can be readily effected through mutations or crossovers. This has been found to help prevent premature convergence at so called *Hamming walls*, in which too many simultaneous mutations (or crossover events) must occur in order to change the chromosome to a better solution.

Other approaches involve using arrays of real-valued numbers instead of bit strings to represent chromosomes. Results from the theory of schemata suggest that in general the smaller the alphabet, the better the performance, but it was initially surprising to researchers that good results were obtained from using real-valued chromosomes. This was explained as the set of real values in a finite population of chromosomes as forming a *virtual alphabet* (when selection and recombination are dominant) with a much lower cardinality than would be expected from a floating point representation.^{[9][10]}

An expansion of the Genetic Algorithm accessible problem domain can be obtained through more complex encoding of the solution pools by concatenating several types of heterogeneously encoded genes into one chromosome.^[11] This particular approach allows for solving optimization problems that require vastly disparate definition domains for the problem parameters. For instance, in problems of cascaded controller tuning, the internal loop controller structure can belong to a conventional regulator of three parameters, whereas the external loop could implement a linguistic controller (such as a fuzzy system) which has an inherently different description. This particular form of encoding requires a specialized crossover mechanism that recombines the chromosome by section, and it is a useful tool for the modelling and simulation of complex adaptive systems, especially

evolution processes.

89.4.2 Elitism

A practical variant of the general process of constructing a new population is to allow the best organism(s) from the current generation to carry over to the next, unaltered. This strategy is known as *elitist selection* and guarantees that the solution quality obtained by the GA will not decrease from one generation to the next.^[12]

89.4.3 Parallel implementations

Parallel implementations of genetic algorithms come in two flavours. Coarse-grained parallel genetic algorithms assume a population on each of the computer nodes and migration of individuals among the nodes. Fine-grained parallel genetic algorithms assume an individual on each processor node which acts with neighboring individuals for selection and reproduction. Other variants, like genetic algorithms for online optimization problems, introduce time-dependence or noise in the fitness function.

89.4.4 Adaptive GAs

Genetic algorithms with adaptive parameters (adaptive genetic algorithms, AGAs) is another significant and promising variant of genetic algorithms. The probabilities of crossover (*pc*) and mutation (*pm*) greatly determine the degree of solution accuracy and the convergence speed that genetic algorithms can obtain. Instead of using fixed values of *pc* and *pm*, AGAs utilize the population information in each generation and adaptively adjust the *pc* and *pm* in order to maintain the population diversity as well as to sustain the convergence capacity. In AGA (adaptive genetic algorithm),^[13] the adjustment of *pc* and *pm* depends on the fitness values of the solutions. In CAGA (clustering-based adaptive genetic algorithm),^[14] through the use of clustering analysis to judge the optimization states of the population, the adjustment of *pc* and *pm* depends on these optimization states. It can be quite effective to combine GA with other optimization methods. GA tends to be quite good at finding generally good global solutions, but quite inefficient at finding the last few mutations to find the absolute optimum. Other techniques (such as simple hill climbing) are quite efficient at finding absolute optimum in a limited region. Alternating GA and hill climbing can improve the efficiency of GA while overcoming the lack of robustness of hill climbing.

This means that the rules of genetic variation may have a different meaning in the natural case. For instance – provided that steps are stored in consecutive order – crossing over may sum a number of steps from maternal DNA adding a number of steps from paternal DNA and so on.

This is like adding vectors that more probably may follow a ridge in the phenotypic landscape. Thus, the efficiency of the process may be increased by many orders of magnitude. Moreover, the *inversion operator* has the opportunity to place steps in consecutive order or any other suitable order in favour of survival or efficiency. (See for instance^[15] or example in travelling salesman problem, in particular the use of an *edge recombination operator*.)

A variation, where the population as a whole is evolved rather than its individual members, is known as gene pool recombination.

A number of variations have been developed to attempt to improve performance of GAs on problems with a high degree of fitness epistasis, i.e. where the fitness of a solution consists of interacting subsets of its variables. Such algorithms aim to learn (before exploiting) these beneficial phenotypic interactions. As such, they are aligned with the Building Block Hypothesis in adaptively reducing disruptive recombination. Prominent examples of this approach include the mGA,^[16] GEMGA^[17] and LLGA.^[18]

89.5 Problem domains

Problems which appear to be particularly appropriate for solution by genetic algorithms include *timetabling* and *scheduling* problems, and many scheduling software packages are based on GAs. GAs have also been applied to *engineering*.^[19] Genetic algorithms are often applied as an approach to solve *global optimization* problems.

As a general rule of thumb genetic algorithms might be useful in problem domains that have a complex *fitness landscape* as mixing, i.e., *mutation* in combination with *crossover*, is designed to move the population away from *local optima* that a traditional *hill climbing* algorithm might get stuck in. Observe that commonly used crossover operators cannot change any uniform population. Mutation alone can provide ergodicity of the overall genetic algorithm process (seen as a *Markov chain*).

Examples of problems solved by genetic algorithms include: mirrors designed to funnel sunlight to a solar collector,^[20] antennae designed to pick up radio signals in space,^[21] and walking methods for computer figures.^[22]

In his *Algorithm Design Manual*, Skiena advises against genetic algorithms for any task:

[I]t is quite unnatural to model applications in terms of genetic operators like mutation and crossover on bit strings. The pseudobiology adds another level of complexity between you and your problem. Second, genetic algorithms take a very long time on nontrivial problems. [...] [T]he analogy with evolution—where significant progress require millions of years—can be quite appropriate.

[...]

I have never encountered any problem where genetic algorithms seemed to me the right way to attack it. Further, I have never seen any computational results reported using genetic algorithms that have favorably impressed me. Stick to simulated annealing for your heuristic search voodoo needs.

—Steven Skiena^{[23]:267}

89.6 History

In 1950, Alan Turing proposed a “learning machine” which would parallel the principles of evolution.^[24] Computer simulation of evolution started as early as in 1954 with the work of Nils Aall Barricelli, who was using the computer at the Institute for Advanced Study in Princeton, New Jersey.^{[25][26]} His 1954 publication was not widely noticed. Starting in 1957,^[27] the Australian quantitative geneticist Alex Fraser published a series of papers on simulation of artificial selection of organisms with multiple loci controlling a measurable trait. From these beginnings, computer simulation of evolution by biologists became more common in the early 1960s, and the methods were described in books by Fraser and Burnell (1970)^[28] and Crosby (1973).^[29] Fraser’s simulations included all of the essential elements of modern genetic algorithms. In addition, Hans-Joachim Bremermann published a series of papers in the 1960s that also adopted a population of solution to optimization problems, undergoing recombination, mutation, and selection. Bremermann’s research also included the elements of modern genetic algorithms.^[30] Other noteworthy early pioneers include Richard Friedberg, George Friedman, and Michael Conrad. Many early papers are reprinted by Fogel (1998).^[31]

Although Barricelli, in work he reported in 1963, had simulated the evolution of ability to play a simple game,^[32] artificial evolution became a widely recognized optimization method as a result of the work of Ingo Rechenberg and Hans-Paul Schwefel in the 1960s and early 1970s – Rechenberg’s group was able to solve complex engineering problems through evolution strategies.^{[33][34][35][36]} Another approach was the evolutionary programming technique of Lawrence J. Fogel, which was proposed for generating artificial intelligence. Evolutionary programming originally used finite state machines for predicting environments, and used variation and selection to optimize the predictive logics. Genetic algorithms in particular became popular through the work of John Holland in the early 1970s, and particularly his book *Adaptation in Natural and Artificial Systems* (1975). His work originated with studies of cellular automata, conducted by Holland and his students at the University of Michigan. Holland introduced a formalized framework for predicting the quality of the next gener-

ation, known as Holland’s Schema Theorem. Research in GAs remained largely theoretical until the mid-1980s, when The First International Conference on Genetic Algorithms was held in Pittsburgh, Pennsylvania.

As academic interest grew, the dramatic increase in desktop computational power allowed for practical application of the new technique. In the late 1980s, General Electric started selling the world’s first genetic algorithm product, a mainframe-based toolkit designed for industrial processes. In 1989, Axicelis, Inc. released Evolver, the world’s first commercial GA product for desktop computers. The New York Times technology writer John Markoff wrote^[37] about Evolver in 1990, and it remained the only interactive commercial genetic algorithm until 1995.^[38] Evolver was sold to Palisade in 1997, translated into several languages, and is currently in its 6th version.^[39]

89.7 Related techniques

See also: List of genetic algorithm applications

89.7.1 Parent fields

Genetic algorithms are a sub-field of:

- Evolutionary algorithms
- Evolutionary computing
- Metaheuristics
- Stochastic optimization
- Optimization

89.7.2 Related fields

Evolutionary algorithms

Evolutionary algorithms is a sub-field of evolutionary computing.

- Evolution strategies (ES, see Rechenberg, 1994) evolve individuals by means of mutation and intermediate or discrete recombination. ES algorithms are designed particularly to solve problems in the real-value domain. They use self-adaptation to adjust control parameters of the search. De-randomization of self-adaptation has led to the contemporary Covariance Matrix Adaptation Evolution Strategy (CMA-ES).

- Evolutionary programming (EP) involves populations of solutions with primarily mutation and selection and arbitrary representations. They use self-adaptation to adjust parameters, and can include other variation operations such as combining information from multiple parents.
- Gene expression programming (GEP) also uses populations of computer programs. These complex computer programs are encoded in simpler linear chromosomes of fixed length, which are afterwards expressed as expression trees. Expression trees or computer programs evolve because the chromosomes undergo mutation and recombination in a manner similar to the canonical GA. But thanks to the special organization of GEP chromosomes, these genetic modifications always result in valid computer programs.^[40]
- Genetic programming (GP) is a related technique popularized by John Koza in which computer programs, rather than function parameters, are optimized. Genetic programming often uses tree-based internal data structures to represent the computer programs for adaptation instead of the list structures typical of genetic algorithms.
- Grouping genetic algorithm (GGA) is an evolution of the GA where the focus is shifted from individual items, like in classical GAs, to groups or subset of items.^[41] The idea behind this GA evolution proposed by Emanuel Falkenauer is that solving some complex problems, a.k.a. *clustering* or *partitioning* problems where a set of items must be split into disjoint group of items in an optimal way, would better be achieved by making characteristics of the groups of items equivalent to genes. These kind of problems include bin packing, line balancing, clustering with respect to a distance measure, equal piles, etc., on which classic GAs proved to perform poorly. Making genes equivalent to groups implies chromosomes that are in general of variable length, and special genetic operators that manipulate whole groups of items. For bin packing in particular, a GGA hybridized with the Dominance Criterion of Martello and Toth, is arguably the best technique to date.
- Interactive evolutionary algorithms are evolutionary algorithms that use human evaluation. They are usually applied to domains where it is hard to design a computational fitness function, for example, evolving images, music, artistic designs and forms to fit users' aesthetic preference.
- Ant colony optimization (**ACO**) uses many ants (or agents) to traverse the solution space and find locally productive areas. While usually inferior to genetic algorithms and other forms of local search, it is able to produce results in problems where no global or up-to-date perspective can be obtained, and thus the other methods cannot be applied.
- Particle swarm optimization (**PSO**) is a computational method for multi-parameter optimization which also uses population-based approach. A population (swarm) of candidate solutions (particles) moves in the search space, and the movement of the particles is influenced both by their own best known position and swarm's global best known position. Like genetic algorithms, the PSO method depends on information sharing among population members. In some problems the PSO is often more computationally efficient than the GAs, especially in unconstrained problems with continuous variables.^[42]
- Intelligent Water Drops or the IWD algorithm^[43] is a nature-inspired optimization algorithm inspired from natural water drops which change their environment to find the near optimal or optimal path to their destination. The memory is the river's bed and what is modified by the water drops is the amount of soil on the river's bed.

Other evolutionary computing algorithms

Evolutionary computation is a sub-field of the metaheuristic methods.

- Harmony search (HS) is an algorithm mimicking the behaviour of musicians in the process of improvisation.
- Memetic algorithm (MA), often called *hybrid genetic algorithm* among others, is a population-based method in which solutions are also subject to local improvement phases. The idea of memetic algorithms comes from **memes**, which unlike genes, can adapt themselves. In some problem areas they are shown to be more efficient than traditional evolutionary algorithms.
- Bacteriologic algorithms (BA) inspired by evolutionary ecology and, more particularly, bacteriologic adaptation. Evolutionary ecology is the study of living organisms in the context of their environment, with the aim of discovering how they adapt. Its basic concept is that in a heterogeneous environment, you can't find one individual that fits the whole environment. So, you need to reason at the population level. It is also believed BAs could be successfully applied to complex positioning

Swarm intelligence

Swarm intelligence is a sub-field of evolutionary computing.

problems (antennas for cell phones, urban planning, and so on) or data mining.^[44]

- **Cultural algorithm** (CA) consists of the population component almost identical to that of the genetic algorithm and, in addition, a knowledge component called the belief space.
- **Differential Search Algorithm** (DS) inspired by migration of superorganisms.^[45]
- **Gaussian adaptation** (normal or natural adaptation, abbreviated NA to avoid confusion with GA) is intended for the maximisation of manufacturing yield of signal processing systems. It may also be used for ordinary parametric optimisation. It relies on a certain theorem valid for all regions of acceptability and all Gaussian distributions. The efficiency of NA relies on information theory and a certain theorem of efficiency. Its efficiency is defined as information divided by the work needed to get the information.^[46] Because NA maximises mean fitness rather than the fitness of the individual, the landscape is smoothed such that valleys between peaks may disappear. Therefore it has a certain “ambition” to avoid local peaks in the fitness landscape. NA is also good at climbing sharp crests by adaptation of the moment matrix, because NA may maximise the disorder (*average information*) of the Gaussian simultaneously keeping the **mean fitness** constant.

Other metaheuristic methods

Metaheuristic methods broadly fall within **stochastic** optimisation methods.

- **Simulated annealing** (SA) is a related global optimization technique that traverses the search space by testing random mutations on an individual solution. A mutation that increases fitness is always accepted. A mutation that lowers fitness is accepted probabilistically based on the difference in fitness and a decreasing temperature parameter. In SA parlance, one speaks of seeking the lowest energy instead of the maximum fitness. SA can also be used within a standard GA algorithm by starting with a relatively high rate of mutation and decreasing it over time along a given schedule.
- **Tabu search** (TS) is similar to simulated annealing in that both traverse the solution space by testing mutations of an individual solution. While simulated annealing generates only one mutated solution, tabu search generates many mutated solutions and moves to the solution with the lowest energy of those generated. In order to prevent cycling and encourage

greater movement through the solution space, a tabu list is maintained of partial or complete solutions. It is forbidden to move to a solution that contains elements of the tabu list, which is updated as the solution traverses the solution space.

- **Extremal optimization** (EO) Unlike GAs, which work with a population of candidate solutions, EO evolves a single solution and makes **local** modifications to the worst components. This requires that a suitable representation be selected which permits individual solution components to be assigned a quality measure (“fitness”). The governing principle behind this algorithm is that of *emergent* improvement through selectively removing low-quality components and replacing them with a randomly selected component. This is decidedly at odds with a GA that selects good solutions in an attempt to make better solutions.

Other stochastic optimisation methods

- The **cross-entropy** (CE) method generates candidates solutions via a parameterized probability distribution. The parameters are updated via cross-entropy minimization, so as to generate better samples in the next iteration.
- **Reactive search optimization** (RSO) advocates the integration of sub-symbolic machine learning techniques into search heuristics for solving complex optimization problems. The word reactive hints at a ready response to events during the search through an internal online feedback loop for the self-tuning of critical parameters. Methodologies of interest for Reactive Search include machine learning and statistics, in particular reinforcement learning, active or query learning, neural networks, and metaheuristics.

89.8 See also

- List of genetic algorithm applications
- Propagation of schema
- Universal Darwinism
- Metaheuristics

89.9 References

- [1] Mitchell 1996, p. 2.
[2] Whitley 1994, p. 66.

- [3] Eiben, A. E. et al (1994). "Genetic algorithms with multi-parent recombination". PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: 78–87. ISBN 3-540-58484-6.
- [4] Ting, Chuan-Kang (2005). "On the Mean Convergence Time of Multi-parent Genetic Algorithms Without Selection". Advances in Artificial Life: 403–412. ISBN 978-3-540-28848-0.
- [5] Akbari, Ziarati (2010). "A multilevel evolutionary algorithm for optimizing numerical functions" IJIEC 2 (2011): 419–430
- [6] Goldberg 1989, p. 41.
- [7] Taherdangkoo, Mohammad; Paziresh, Mahsa; Yazdi, Mehran; Bagheri, Mohammad Hadi (19 November 2012). "An efficient algorithm for function optimization: modified stem cells algorithm". *Central European Journal of Engineering* 3 (1): 36–50. doi:10.2478/s13531-012-0047-8.
- [8] Wolpert, D.H., Macready, W.G., 1995. No Free Lunch Theorems for Optimisation. Santa Fe Institute, SFI-TR-05-010, Santa Fe.
- [9] Goldberg, David E. (1991). "The theory of virtual alphabets". *Parallel Problem Solving from Nature, Lecture Notes in Computer Science* 496: 13–22. doi:10.1007/BFb0029726. Retrieved 2 July 2013.
- [10] Janikow, C. Z.; Michalewicz, Z. (1991). "An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms". *Proceedings of the Fourth International Conference on Genetic Algorithms*: 31–36. Retrieved 2 July 2013.
- [11] Patrascu, M.; Stancu, A.F.; Pop, F. (2014). "HELGA: a heterogeneous encoding lifelike genetic algorithm for population evolution modeling and simulation". *Soft Computing* 18: 2565–2576.
- [12] Baluja, Shumeet; Caruana, Rich (1995). *Removing the genetics from the standard genetic algorithm*. ICML.
- [13] Srinivas. M and Patnaik. L, "Adaptive probabilities of crossover and mutation in genetic algorithms," IEEE Transactions on System, Man and Cybernetics, vol.24, no.4, pp.656–667, 1994.
- [14] ZHANG. J, Chung. H and Lo. W. L, "Clustering-Based Adaptive Crossover and Mutation Probabilities for Genetic Algorithms", IEEE Transactions on Evolutionary Computation vol.11, no.3, pp. 326–335, 2007.
- [15] Evolution-in-a-nutshell
- [16] D.E. Goldberg, B. Korb, and K. Deb. "Messy genetic algorithms: Motivation, analysis, and first results". Complex Systems, 5(3):493–530, October 1989.
- [17] Gene expression: The missing link in evolutionary computation
- [18] G. Harik. Learning linkage to efficiently solve problems of bounded difficulty using genetic algorithms. PhD thesis, Dept. Computer Science, University of Michigan, Ann Arbor, 1997
- [19] Tomoiagă B, Chindriş M, Sumper A, Sudria-Andreu A, Villafafila-Robles R. Pareto Optimal Reconfiguration of Power Distribution Systems Using a Genetic Algorithm Based on NSGA-II. Energies. 2013; 6(3):1439–1455.
- [20] Gross, Bill. "A solar energy system that tracks the sun". *TED*. Retrieved 20 November 2013.
- [21] Hornby, G. S.; Linden, D. S.; Lohn, J. D., *Automated Antenna Design with Evolutionary Algorithms*
- [22] <http://goatstream.com/research/papers/SA2013/index.html>
- [23] Skiena, Steven (2010). *The Algorithm Design Manual* (2nd ed.). Springer Science+Business Media. ISBN 1-849-96720-2.
- [24] Turing, Alan M. "Computing machinery and intelligence". *Mind* LIX (238): 433–460. doi:10.1093/mind/LIX.236.433.
- [25] Barricelli, Nils Aall (1954). "Esempi numerici di processi di evoluzione". *Methodos*: 45–68.
- [26] Barricelli, Nils Aall (1957). "Symbiogenetic evolution processes realized by artificial methods". *Methodos*: 143–182.
- [27] Fraser, Alex (1957). "Simulation of genetic systems by automatic digital computers. I. Introduction". *Aust. J. Biol. Sci.* 10: 484–491.
- [28] Fraser, Alex; Burnell, Donald (1970). *Computer Models in Genetics*. New York: McGraw-Hill. ISBN 0-07-021904-4.
- [29] Crosby, Jack L. (1973). *Computer Simulation in Genetics*. London: John Wiley & Sons. ISBN 0-471-18880-8.
- [30] 02.27.96 - UC Berkeley's Hans Bremermann, professor emeritus and pioneer in mathematical biology, has died at 69
- [31] Fogel, David B. (editor) (1998). *Evolutionary Computation: The Fossil Record*. New York: IEEE Press. ISBN 0-7803-3481-7.
- [32] Barricelli, Nils Aall (1963). "Numerical testing of evolution theories. Part II. Preliminary tests of performance, symbiogenesis and terrestrial life". *Acta Biotheoretica* (16): 99–126.
- [33] Rechenberg, Ingo (1973). *Evolutionsstrategie*. Stuttgart: Holzmann-Froboog. ISBN 3-7728-0373-3.
- [34] Schwefel, Hans-Paul (1974). *Numerische Optimierung von Computer-Modellen (PhD thesis)*.
- [35] Schwefel, Hans-Paul (1977). *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie : mit einer vergleichenden Einführung in die Hill-Climbing- und Zufallsstrategie*. Basel; Stuttgart: Birkhäuser. ISBN 3-7643-0876-1.

- [36] Schwefel, Hans-Paul (1981). *Numerical optimization of computer models (Translation of 1977 Numerische Optimierung von Computer-Modellen mittels der Evolutionstrategie)*. Chichester ; New York: Wiley. ISBN 0-471-09988-0.
- [37] Markoff, John (29 August 1990). "What's the Best Answer? It's Survival of the Fittest". New York Times. Retrieved 9 August 2009. Check date values in: |year= / |date= mismatch (help)
- [38] Ruggiero, Murray A.. (2009-08-01) Fifteen years and counting. Futuresmag.com. Retrieved on 2013-08-07.
- [39] Evolver: Sophisticated Optimization for Spreadsheets. Palisade. Retrieved on 2013-08-07.
- [40] Ferreira, C. "Gene Expression Programming: A New Adaptive Algorithm for Solving Problems". *Complex Systems*, Vol. 13, issue 2: 87-129.
- [41] Falkenauer, Emanuel (1997). *Genetic Algorithms and Grouping Problems*. Chichester, England: John Wiley & Sons Ltd. ISBN 978-0-471-97150-4.
- [42] Rania Hassan, Babak Cohanim, Olivier de Weck, Gerhard Venter (2005) A comparison of particle swarm optimization and the genetic algorithm
- [43] Hamed Shah-Hosseini, The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm, International Journal of Bio-Inspired Computation (IJBIC), vol. 1, no. ½, 2009,
- [44] Baudry, Benoit; Franck Fleurey, Jean-Marc Jézéquel, and Yves Le Traon (March–April 2005). "Automatic Test Case Optimization: A Bacteriologic Algorithm" (PDF). *IEEE Software* (IEEE Computer Society) **22** (2): 76–82. doi:10.1109/MS.2005.30. Retrieved 9 August 2009.
- [45] Civicioglu, P. (2012). "Transforming Geocentric Cartesian Coordinates to Geodetic Coordinates by Using Differential Search Algorithm". *Computers & Geosciences* **46**: 229–247. doi:10.1016/j.cageo.2011.12.011.
- [46] Kjellström, G. (December 1991). "On the Efficiency of Gaussian Adaptation". *Journal of Optimization Theory and Applications* **71** (3): 589–597. doi:10.1007/BF00941405.
- Cha, Sung-Hyuk; Tappert, Charles C. (2009). "A Genetic Algorithm for Constructing Compact Binary Decision Trees". *Journal of Pattern Recognition Research* **4** (1): 1–13. doi:10.13176/11.44.
- Fraser, Alex S. (1957). "Simulation of Genetic Systems by Automatic Digital Computers. I. Introduction". *Australian Journal of Biological Sciences* **10**: 484–491.
- Goldberg, David (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley Professional. ISBN 978-0201157673.
- Goldberg, David (2002). *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Norwell, MA: Kluwer Academic Publishers. ISBN 978-1402070983.
- Fogel, David. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (3rd ed.). Piscataway, NJ: IEEE Press. ISBN 978-0471669517.
- Holland, John (1992). *Adaptation in Natural and Artificial Systems*. Cambridge, MA: MIT Press. ISBN 978-0262581110.
- Koza, John (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press. ISBN 978-0262111706.
- Michalewicz, Zbigniew (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag. ISBN 978-3540606765.
- Mitchell, Melanie (1996). *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press. ISBN 9780585030944.
- Poli, R.; Langdon, W. B.; McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. Lulu.com, freely available from the internet. ISBN 978-1-4092-0073-4.
- Rechenberg, Ingo (1994): *Evolutionsstrategie '94*, Stuttgart: Fromman-Holzboog.
- Schmitt, Lothar M; Nehaniv, Christopher L; Fujii, Robert H (1998), *Linear analysis of genetic algorithms*, Theoretical Computer Science 208: 111–148
- Schmitt, Lothar M (2001), *Theory of Genetic Algorithms*, Theoretical Computer Science 259: 1–61
- Schmitt, Lothar M (2004), *Theory of Genetic Algorithms II: models for genetic operators over the string-tensor representation of populations and convergence to global optima for arbitrary fitness function under scaling*, Theoretical Computer Science 310: 181–231

89.10 Bibliography

- Banzhaf, Wolfgang; Nordin, Peter; Keller, Robert; Francone, Frank (1998). *Genetic Programming – An Introduction*. San Francisco, CA: Morgan Kaufmann. ISBN 978-1558605107.
- Bies, Robert R.; Muldoon, Matthew F.; Pollock, Bruce G.; Manuck, Steven; Smith, Gwenn; Sale, Mark E. (2006). "A Genetic Algorithm-Based, Hybrid Machine Learning Approach to Model Selection". *Journal of Pharmacokinetics and Pharmacodynamics* (Netherlands: Springer): 196–221.

- Schwefel, Hans-Paul (1974): Numerische Optimierung von Computer-Modellen (PhD thesis). Reprinted by Birkhäuser (1977).
- Vose, Michael (1999). *The Simple Genetic Algorithm: Foundations and Theory*. Cambridge, MA: MIT Press. ISBN 978-0262220583.
- Whitley, Darrell (1994). “A genetic algorithm tutorial”. *Statistics and Computing* 4 (2): 65–85. doi:10.1007/BF00175354.
- Hingston, Philip; Barone, Luigi; Michalewicz, Zbigniew (2008). *Design by Evolution: Advances in Evolutionary Design*. Springer. ISBN 978-3540741091.
- Eiben, Agoston; Smith, James (2003). *Introduction to Evolutionary Computing*. Springer. ISBN 978-3540401841.

89.11 External links

89.11.1 Resources

- Genetic Algorithms Index Provides a list of resources in the genetic algorithms field

89.11.2 Tutorials

- Genetic Algorithms Computer “evolve” in ways that resemble natural selection can solve complex problems even their creators do not fully understand An excellent introduction to GA by John Holland and with an application to the Prisoner’s Dilemma
- An online interactive GA tutorial for a reader to practise or learn how a GA works: Learn step by step or watch global convergence in batch, change the population size, crossover rates/bounds, mutation rates/bounds and selection mechanisms, and add constraints.
- A Genetic Algorithm Tutorial by Darrell Whitley Computer Science Department Colorado State University An excellent tutorial with lots of theory
- “Essentials of Metaheuristics”, 2009 (225 p). Free open text by Sean Luke.
- Global Optimization Algorithms – Theory and Application

Chapter 90

Genetic algorithms in economics

Main article: genetic algorithm

Genetic algorithms have increasingly been applied to economics since the pioneering work by John H. Miller in 1986. It has been used to characterize a variety of models including the cobweb model, the overlapping generations model, game theory, schedule optimization and asset pricing. Specifically, it has been used as a model to represent learning, rather than as a means for fitting a model.

90.1 Genetic algorithm in the cob-web model

The cobweb model is a simple supply and demand model for a good over t periods. Firms (agents) make a production quantity decision in a given period, however their output is not produced until the following period. Thus, the firms are going to have to use some sort of method to forecast what the future price will be. The GA is used as a sort of learning behaviour for the firms. Initially their quantity production decisions are random, however each period they learn a little more. The result is the agents converge within the area of the rational expectations (RATEX) equilibrium for the stable and unstable case. If the election operator is used, the GA converges exactly to the RATEX equilibrium.

There are two types of learning methods these agents can be deployed with: social learning and individual learning. In social learning, each firm is endowed with a single string which is used as its quantity production decision. It then compares this string against other firms' strings. In the individual learning case, agents are endowed with a pool of strings. These strings are then compared against other strings within the agent's population pool. This can be thought of as mutual competing ideas within a firm whereas in the social case, it can be thought of as a firm learning from more successful firms. Note that in the social case and in the individual learning case with identical cost functions, that this is a homogeneous solution, that is all agents' production decisions are identical. However, if the cost functions are not identical, this will result in

a heterogeneous solution, where firms produce different quantities (note that they are still locally homogeneous, that is within the firm's own pool all the strings are identical).

After all agents have made a quantity production decision, the quantities are aggregated and plugged into a demand function to get a price. Each firm's profit is then calculated. Fitness values are then calculated as a function of profits. After the offspring pool is generated, hypothetical fitness values are calculated. These hypothetical values are based on some sort of estimation of the price level, often just by taking the previous price level.

90.2 References

- J H Miller, 'A Genetic Model of Adaptive Economic Behavior', University of Michigan working paper, 1986.
- J Arifovic, 'Learning by Genetic Algorithm in Economic Environments', PhD Thesis, University of Chicago, 1991.
- J Arifovic, 'Genetic Algorithm Learning and the Cobweb Model ', Journal of Economic Dynamics and Control, vol. 18, Issue 1, (January 1994), 3–28.

90.3 External links

- Centre for Adaptive Behaviour in Economics
- Agent-Based Computational Economics and Artificial Life: A Brief Intro

Chapter 91

Glowworm swarm optimization

The **glowworm swarm optimization (GSO)** is a **swarm intelligence** optimization algorithm developed based on the behaviour of **glowworms** (also known as fireflies or lightning bugs). The behaviour pattern of glowworms which is used for this algorithm is the apparent capability of the glowworms to change the intensity of the luciferin emission and thus appear to glow at different intensities.

1. The GSO algorithm makes the agents glow at intensities approximately proportional to the function value being optimized. It is assumed that glowworms of brighter intensities attract glowworms that have lower intensity.

2. The second significant part of the algorithm incorporates a dynamic decision range by which the effect of distant glowworms are discounted when a glowworm has sufficient number of neighbours or the range goes beyond the range of perception of the glowworms.

The part 2 of the algorithm makes it different from **Firefly algorithm(FA)**. In the **Firefly algorithm**, fireflies can automatically subdivide into subgroups and thus can find multiple global solutions simultaneously, and thus FA is very suitable for multimodal problems. However, in GSO, there is no “sufficient number or neighbours” limit and there is no perception limit based on distance, but it can have still have “cognitive limits” which allows swarms of glowworms to split into sub-groups and converge to high function value points. This property of the algorithm allows it to be used to identify multiple peaks of a multimodal function and makes it part of **Evolutionary multi-modal optimization algorithms family**.

The GSO algorithm was developed and introduced by K.N. Krishnanand and D. Ghose in 2005 at the Guidance, Control, and Decision Systems Laboratory in the Department of Aerospace Engineering at the Indian Institute of Science, Bangalore, India. Subsequently, it has been used in various applications and several papers have appeared in the literature using the GSO algorithm.

- Evolutionary multi-modal optimization

91.2 References

- K.N. Krishnanand and D. Ghose (2005). Detection of multiple source locations using a glowworm metaphor with applications to collective robotics. *IEEE Swarm Intelligence Symposium*, Pasadena, California, USA, pp. 84- 91. doi:10.1109/SIS.2005.1501606
- K.N. Krishnanand and D. Ghose. (2006). Glowworm swarm based optimization algorithm for multimodal functions with collective robotics applications. *Multi-agent and Grid Systems*, **2**(3):209-222.
- K.N. Krishnanand and D. Ghose. (2009) Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. *Swarm Intelligence*, **3**(2):87- 124. doi:10.1007/s11721-008-0021-5
- K.N. Krishnanand and D. Ghose. (2008). Theoretical foundations for rendezvous of glowworm-inspired agent swarms at multiple locations. *Robotics and Autonomous Systems*, **56**(7):549- 569. doi:10.1016/j.robot.2007.11.003
- Prasad, Bhanu (2008). Studies in Fuzziness and Soft Computing, Soft Computing Applications in Industry, Volume 226/2008, 165- 87, doi:10.1007/978-3-540-77465-5

91.3 External links

- Glowworm Swarm Optimization algorithm in C++

91.1 See also

- Metaheuristic
- Firefly algorithm (FA)

Chapter 92

Gradient method

In optimization, **gradient method** is an algorithm to solve problems of the form

$$\min_{x \in \mathbb{R}^n} f(x)$$

with the search directions defined by the gradient of the function at the current point. Examples of gradient method are the **gradient descent** and the **conjugate gradient**.

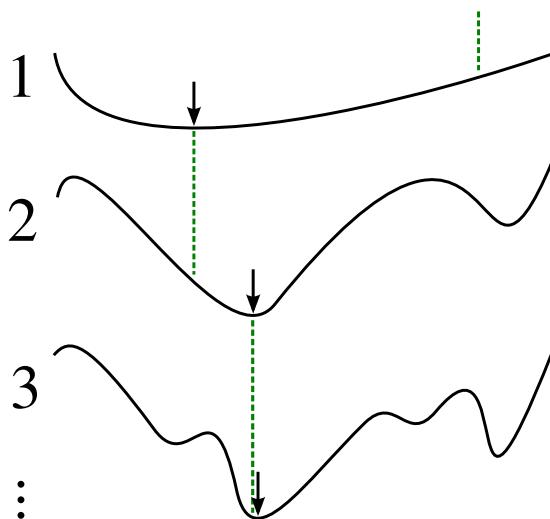
92.1 See also

Chapter 93

Graduated optimization

Graduated optimization is a **global optimization** technique that attempts to solve a difficult optimization problem by initially solving a greatly simplified problem, and progressively transforming that problem (while optimizing) until it is equivalent to the difficult optimization problem.^{[1][2][3]}

93.1 Technique description



An illustration of graduated optimization.

Graduated optimization is an improvement to hill climbing that enables a hill climber to avoid settling into local optima. It breaks a difficult optimization problem into a sequence of optimization problems, such that the first problem in the sequence is convex (or nearly convex), the solution to each problem gives a good starting point to the next problem in the sequence, and the last problem in the sequence is the difficult optimization problem that it ultimately seeks to solve. Often, graduated optimization gives better results than simple hill climbing. Further, when certain conditions exist, it can be shown to find an optimal solution to the final problem in the sequence. These conditions are:

- The first optimization problem in the sequence can

be solved given the initial starting point.

- The locally convex region around the global optimum of each problem in the sequence includes the point that corresponds to the global optimum of the previous problem in the sequence.

It can be shown inductively that if these conditions are met, then a hill climber will arrive at the global optimum for the difficult problem. Unfortunately, it can be difficult to find a sequence of optimization problems that meet these conditions. Often, graduated optimization yields good results even when the sequence of problems cannot be proven to strictly meet all of these conditions.

93.2 Some examples

Graduated optimization is commonly used in image processing for locating objects within a larger image. This problem can be made to be *more convex* by blurring the images. Thus, objects can be found by first searching the most-blurred image, then starting at that point and searching within a less-blurred image, and continuing in this manner until the object is located with precision in the original sharp image. The proper choice of the blurring operator depends on the geometric transformation relating the object in one image to the other.^[4]

Graduated optimization can be used in manifold learning. The Manifold Sculpting algorithm, for example, uses graduated optimization to seek a manifold embedding for non-linear dimensionality reduction.^[5] It gradually scales variance out of extra dimensions within a data set while optimizing in the remaining dimensions. It has also been used to calculate conditions for fractionation with tumors,^[6] for object tracking in computer vision,^[7] and other purposes.

A thorough review of the method and its applications can be found in.^[3]

93.3 Related optimization techniques

Simulated annealing is closely related to graduated optimization. Instead of smoothing the function over which it is optimizing, simulated annealing randomly perturbs the current solution by a decaying amount, which may have a similar effect. Because simulated annealing relies on random sampling to find improvements, however, its computation complexity is exponential in the number of dimensions being optimized. By contrast, graduated optimization smooths the function being optimized, so local optimization techniques that are efficient in high-dimensional space (such as gradient-based techniques, hill climbers, etc.) may still be used.

93.4 See also

- Numerical continuation

93.5 References

- [1] Thacker, Neil; Cootes, Tim (1996). “Graduated Non-Convexity and Multi-Resolution Optimization Methods”. *Vision Through Optimization*.
- [2] Blake, Andrew; Zisserman, Andrew (1987). *Visual Reconstruction*. MIT Press. ISBN 0-262-02271-0.
- [3] Hossein Mobahi, John W. Fisher III. On the Link Between Gaussian Homotopy Continuation and Convex Envelopes, In Lecture Notes in Computer Science (EMM-CVPR 2015), Springer, 2015.
- [4] Hossein Mobahi, C. Lawrence Zitnick, Yi Ma. Seeing through the Blur, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2012.
- [5] Gashler, M.; Ventura, D.; Martinez, T. (2008). “Iterative Non-linear Dimensionality Reduction with Manifold Sculpting”. In Platt, J. C.; Koller, D.; Singer, Y. et al. *Advances in Neural Information Processing Systems 20*. Cambridge, MA: MIT Press. pp. 513–20.
- [6] Afanasjev, BP; Akimov, AA; Kozlov, AP; Berkovic, AN (1989). “Graduated optimization of fractionation using a 2-component model”. *Radiobiologia, radiotherapia* **30** (2): 131–5. PMID 2748803.
- [7] Ming Ye; Haralick, R.M.; Shapiro, L.G. (2003). “Estimating piecewise-smooth optical flow with global matching and graduated optimization”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **25** (12): 1625–30. doi:10.1109/TPAMI.2003.1251156.

Chapter 94

Great Deluge algorithm

The **Great Deluge algorithm** (GD) is a generic algorithm applied to optimization problems. It is similar in many ways to the hill-climbing and simulated annealing algorithms.

The name comes from the analogy that in a great deluge a person climbing a hill will try to move in any direction that does not get his/her feet wet in the hope of finding a way up as the water level rises.

In a typical implementation of the GD, the algorithm starts with a poor approximation, S , of the optimum solution. A numerical value called the *badness* is computed based on S and it measures how undesirable the initial approximation is. The higher the value of *badness* the more undesirable is the approximate solution. Another numerical value called the *tolerance* is calculated based on a number of factors, often including the initial badness.

A new approximate solution S' , called a neighbour of S , is calculated based on S . The badness of S' , b' , is computed and compared with the tolerance. If b' is better than tolerance, then the algorithm is recursively restarted with $S := S'$, and $\text{tolerance} := \text{decay}(\text{tolerance})$ where decay is a function that lowers the tolerance (representing a rise in water levels). If b' is worse than tolerance, a different neighbour S^* of S is chosen and the process repeated. If all the neighbours of S produce approximate solutions beyond *tolerance*, then the algorithm is terminated and S is put forward as the best approximate solution obtained.

94.2 See also

- de:Gunter Dueck

94.1 References

- Gunter Dueck: “New Optimization Heuristics: The Great Deluge Algorithm and the Record-to-Record Travel”, Technical report, IBM Germany, Heidelberg Scientific Center, 1990.
- Gunter Dueck: “New Optimization Heuristics The Great Deluge Algorithm and the Record-to-Record Travel”, Journal of Computational Physics, Volume 104, Issue 1, p. 86-92, 1993

Chapter 95

Guided Local Search

Guided Local Search is a metaheuristic search method. A meta-heuristic method is a method that sits on top of a local search algorithm to change its behaviour.

Guided Local Search builds up penalties during a search. It uses penalties to help local search algorithms escape from local minima and plateaus. When the given local search algorithm settles in a local optimum, GLS modifies the objective function using a specific scheme (explained below). Then the local search will operate using an augmented objective function, which is designed to bring the search out of the local optimum. The key is in the way that the objective function is modified.

95.1.2 Selective penalty modifications

GLS computes the utility of penalising each feature. When the Local Search algorithm returns a local minimum x , GLS penalizes all those features (through increments to the penalty of the features) present in that solution which have maximum utility, $\text{util}(x, i)$, as defined below.

$$\text{util}(x, i) = I_i(x) \frac{c_i(x)}{1 + p_i}.$$

The idea is to penalise features that have high costs, although the utility of doing so decreases as the feature is penalised more and more often.

95.1 Overview

95.1.1 Solution features

To apply GLS, solution features must be defined for the given problem. Solution features are defined to distinguish between solutions with different characteristics, so that regions of similarity around local optima can be recognized and avoided. The choice of solution features depends on the type of problem, and also to a certain extent on the local search algorithm. For each feature f_i a cost function c_i is defined.

Each feature is also associated with a penalty p_i (initially set to 0) to record the number of occurrences of the feature in local minima.

The features and costs often come directly from the objective function. For example, in the traveling salesman problem, “whether the tour goes directly from city X to city Y” can be defined to be a feature. The distance between X and Y can be defined to be the cost. In the SAT and weighted MAX-SAT problems, the features can be “whether clause C satisfied by the current assignments”.

At the implementation level, we define for each feature i an Indicator Function I_i indicating whether the feature is present in the current solution or not. I_i is 1 when solution x exhibits property i , 0 otherwise.

95.1.3 Searching through an augmented cost function

GLS uses an augmented cost function (defined below), to allow it to guide the Local Search algorithm out of the local minimum, through penalising features present in that local minimum. The idea is to make the local minimum more costly than the surrounding search space, where these features are not present.

$$g(x) = f(x) + \lambda a \sum_{1 \leq i \leq m} I_i(x)p_i$$

The parameter λ may be used to alter the intensification of the search for solutions. A higher value for λ will result in a more diverse search, where plateaus and basins are searched more coarsely; a low value will result in a more intensive search for the solution, where the plateaus and basins in the search landscape are searched in finer detail. The coefficient a is used to make the penalty part of the objective function balanced relative to changes in the objective function and is problem specific. A simple heuristic for setting a is simply to record the average change in objective function up until the first local minimum, and then set a to this value divided by the number of GLS features in the problem instance.

95.1.4 Extensions of Guided Local Search

Mills (2002) has described an Extended Guided Local Search (EGLS) which utilises random moves and an aspiration criterion designed specifically for penalty based schemes. The resulting algorithm improved the robustness of GLS over a range of parameter settings, particularly in the case of the quadratic assignment problem. A general version of the GLS algorithm, using a min-conflicts based hill climber (Minton et al. 1992) and based partly on GENET for constraint satisfaction and optimisation, has also been implemented in the Computer Aided Constraint Programming project.

Alsheddy (2011) extended Guided Local Search to multi-objective optimization, and demonstrated its use in staff empowerment in scheduling .

95.2 Related work

GLS was built on GENET, which was developed by Chang Wang, Edward Tsang and Andrew Davenport.

The breakout method is very similar to GENET. It was designed for constraint satisfaction.

Tabu search is a class of search methods which can be instantiated to specific methods. GLS can be seen as a special case of Tabu search.

By sitting GLS on top of genetic algorithm, Tung-leng Lau introduced the Guided Genetic Programming (GGA) algorithm. It was successfully applied to the general assignment problem (in scheduling), processors configuration problem (in electronic design) and a set of radio-link frequency assignment problems (an abstracted military application).

Choi et al. cast GENET as a Lagrangian search.

95.3 Bibliography

- Alsheddy, A., Empowerment scheduling: a multi-objective optimization approach using Guided Local Search, PhD Thesis, School of Computer Science and Electronic Engineering, University of Essex, 2011
- Choi, K.M.F., Lee, J.H.M. & Stuckey, P.J., A Lagrangian Reconstruction of GENET, Artificial Intelligence, 2000, 123(1-2), 1-39
- Davenport A., Tsang E.P.K., Kangmin Zhu & C J Wang, GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement, Proc., AAAI, 1994, p.325-330
- Lau, T.L. & Tsang, E.P.K., Solving the processor configuration problem with a mutation-based genetic algorithm, International Journal on Artificial Intelligence Tools (IJAIT), World Scientific, Vol.6, No.4, December 1997, 567-585
- Lau, T.L. & Tsang, E.P.K., Guided genetic algorithm and its application to radio link frequency assignment problems, Constraints, Vol.6, No.4, 2001, 373-398
- Lau, T.L. & Tsang, E.P.K., The guided genetic algorithm and its application to the general assignment problems, IEEE 10th International Conference on Tools with Artificial Intelligence (ICTAI'98), Taiwan, November 1998
- Mills, P. & Tsang, E.P.K., Guided local search for solving SAT and weighted MAX-SAT problems, Journal of Automated Reasoning, Special Issue on Satisfiability Problems, Kluwer, Vol.24, 2000, 205-223
- Mills, P. & Tsang, E.P.K. & Ford, J., Applying an Extended Guided Local Search on the Quadratic Assignment Problem, Annals of Operations Research, Kluwer Academic Publishers, Vol.118, 2003, 121-135
- Minton, S., Johnston, M., Philips, A.B. & Laird, P., Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, Artificial Intelligence (Special Volume on Constraint Based Reasoning), Vol.58, Nos.1-3 1992, 161-205
- Tsang, E.P.K. & Voudouris, C., Fast local search and guided local search and their application to British Telecom's workforce scheduling problem, Operations Research Letters, Elsevier Science Publishers, Amsterdam, Vol.20, No.3, March 1997, 119-127
- Voudouris, C, Guided local search for combinatorial optimisation problems, PhD Thesis, Department of Computer Science, University of Essex, Colchester, UK, July, 1997
- Voudouris, C., Guided Local Search—An illustrative example in function optimisation, BT Technology Journal, Vol.16, No.3, July 1998, 46-50
- Voudouris, C. & Tsang, E.P.K., Guided Local Search and its application to the Travelling Salesman Problem, European Journal of Operational Research, Anbar Publishing, Vol.113, Issue 2, March 1999, 469-499
- Voudouris, C. & Tsang, E.P.K., Guided local search joins the elite in discrete optimisation, DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 57, 2001, 29-39
- Voudouris, C. & Tsang, E.P.K., Guided local search, in F. Glover (ed.), Handbook of metaheuristics, Kluwer, 2003, 185-218

- Voudouris, C., Tsang, E.P.K. & Alshedy, A., Guided local search, Chapter 11, in M. Gendreau & J-Y Potvin (ed.), *Handbook of Metaheuristics*, Springer, 2010, 321-361

95.4 External links

- [Guided Local Search Home Page](#)

Chapter 96

Harmony search

In computer science and operations research, **harmony search** (HS) is a phenomenon-mimicking algorithm (also known as **metaheuristic algorithm**, **soft computing algorithm** or **evolutionary algorithm**) inspired by the improvisation process of musicians proposed by Zong Woo Geem in 2001. In the HS algorithm, each musician (= decision variable) plays (= generates) a note (= a value) for finding a best harmony (= global optimum) all together. Proposents claim the following merits:

- HS does not require differential gradients, thus it can consider discontinuous functions as well as continuous functions.
- HS can handle discrete variables as well as continuous variables.
- HS does not require initial value setting for the variables.
- HS is free from divergence.
- HS may escape local optima.
- HS may overcome the drawback of GA's **building block theory** which works well only if the relationship among variables in a chromosome is carefully considered. If neighbor variables in a chromosome have weaker relationship than remote variables, building block theory may not work well because of crossover operation. However, HS explicitly considers the relationship using **ensemble** operation.
- HS has a novel **stochastic derivative** applied to discrete variables, which uses musician's experiences as a searching direction.
- Certain HS variants do not require algorithm parameters such as HMCR and PAR, thus novice users can easily use the algorithm.

96.1 Basic harmony search algorithm

Harmony search tries to find a vector \mathbf{x} which optimizes (minimizes or maximizes) a certain objective function.

The algorithm has the following steps:

Step 1: Generate random vectors ($\mathbf{x}^1, \dots, \mathbf{x}^{hms}$) as many as hms (harmony memory size), then store them in harmony memory (HM).

$$\text{HM} = \left[\begin{array}{ccc|c} x_1^1 & \cdots & x_n^1 & | & f(\mathbf{x}^1) \\ \vdots & \ddots & \vdots & | & \vdots \\ x_1^{hms} & \cdots & x_n^{hms} & | & f(\mathbf{x}^{hms}) \end{array} \right].$$

Step 2: Generate a new vector \mathbf{x}' . For each component x'_i ,

- with probability $hmcr$ (harmony memory considering rate; $0 \leq hmcr \leq 1$), pick the stored value from HM: $x'_i \leftarrow x_i^{int(u(0,1)*hms)+1}$
- with probability $1 - hmcr$, pick a random value within the allowed range.

Step 3: Perform additional work if the value in Step 2 came from HM.

- with probability par (pitch adjusting rate; $0 \leq par \leq 1$), change x'_i by a small amount: $x'_i \leftarrow x'_i + \delta$ or $x'_i \leftarrow x'_i - \delta$ for discrete variable; or $x'_i \leftarrow x'_i + fw \cdot u(-1, 1)$ for continuous variable.
- with probability $1 - par$, do nothing.

Step 4: If \mathbf{x}' is better than the worst vector \mathbf{x}^{Worst} in HM, replace \mathbf{x}^{Worst} with \mathbf{x}' .

Step 5: Repeat from Step 2 to Step 4 until termination criterion (e.g. maximum iterations) is satisfied.

The parameters of the algorithm are

- hms = the size of the harmony memory. It generally varies from 1 to 100. (typical value = 30)

- $hmcr$ = the rate of choosing a value from the harmony memory. It generally varies from 0.7 to 0.99. (typical value = 0.9)
- par = the rate of choosing a neighboring value. It generally varies from 0.1 to 0.5. (typical value = 0.3)
- δ = the amount between two neighboring values in discrete candidate set.
- fw (fret width, formerly bandwidth) = the amount of maximum change in pitch adjustment. This can be $(0.01 \times \text{allowed range})$ to $(0.001 \times \text{allowed range})$.

It is possible to vary the parameter values as the search progresses, which gives an effect similar to simulated annealing.

Parameter-setting-free researches have been also performed. In the researches, algorithm users do not need tedious parameter setting process.

96.2 Other related algorithms

Harmony search lies in the fields of:

- Evolutionary computing
 - Metaheuristics
 - Stochastic optimization
 - Optimization

Other evolutionary computing methods include:

- Evolutionary algorithms, including:
 - Genetic algorithms
 - Genetic programming
- Swarm algorithms, including:
 - Ant colony optimization
 - Particle swarm optimization
 - Intelligent Water Drops

Other metaheuristic methods include:

- Simulated annealing
- Tabu search

Other stochastic methods include:

- Cross-entropy method

96.3 Criticism

In 2010, Dennis Weyland, a PhD student at the Dalle Molle Institute for Artificial Intelligence Research in Switzerland published an article titled “A Rigorous Analysis of the Harmony Search Algorithm: How the Research Community can be Misled by a “Novel” Methodology” in the *International Journal of Applied Metaheuristic Computing* (IJAMC),^[1] stating that:

It turns out that *Harmony Search* is a special case of *Evolution Strategies*. We give compelling evidence for the thesis that research in Harmony Search, although undoubtedly conducted with the best of intentions, is fundamentally misguided, marred by a preoccupation with retracing paths already well traveled, and we conclude that future research effort could better be devoted to more promising areas.

A rebuttal was published by Geem in a later issue of the same journal,^[2] (updated manuscript) but Kenneth Sørensen, professor of operations research at Antwerp University, called it “less than fully convincing”.^[3]

Independent of the work of Weyland, Miriam Padberg has shown in 2011 that for binary optimization problems the Harmony Search algorithm is equivalent to a certain evolutionary algorithm.^[4] In fact, the reasoning is similar to that used in the work of Weyland, but this time explicitly stated in a rigorous mathematical way.

96.4 Notes

[1] Weyland, Dennis (2010). “A Rigorous Analysis of the Harmony Search Algorithm: How the Research Community can be Misled by a “Novel” Methodology”. *International Journal of Applied Metaheuristic Computing* 1 (2): 50–60. doi:10.4018/jamc.2010040104.

[2] Geem, Zong Woo (2010). “Research Commentary: Survival of the Fittest Algorithm or the Novelest Algorithm?”. *International Journal of Applied Metaheuristic Computing* 1 (4): 75–79. doi:10.4018/jamc.2010100105.

[3] Sørensen, Kenneth. “Metaheuristics—the metaphor exposed”. *International Transactions in Operational Research*. doi:10.1111/itor.12001.

[4] Padberg, Miriam (2012). “Harmony Search Algorithms for binary optimization problems”. *Operations Research Proceedings 2011*: 343–348.

96.5 References

96.5.1 General information

- Algorithm Website: Harmony Search Algorithm
- Book 1 Music-Inspired Harmony Search Algorithm, Springer 2009
- Book 2 Recent Advances in Harmony Search Algorithm, Springer 2010
- Book 3 Harmony Search Algorithms for Structural Design Optimization, Springer 2009
- Book 4 Optimal Design of Water Distribution Networks Using Harmony Search, LAP 2009
- Book 5 Engineering Optimization: An Introduction with Metaheuristic Applications, Wiley 2010
- Book 6 Clever Algorithms: Nature-Inspired Programming Recipes, Lulu.com 2011

96.5.2 Theory of harmony search

- Original Harmony Search: Geem ZW, Kim JH, and Loganathan GV, **A New Heuristic Optimization Algorithm: Harmony Search**, Simulation, 2001.
- Stochastic Partial Derivative: Geem ZW, Novel Derivative of Harmony Search Algorithm for Discrete Design Variables, Applied Mathematics and Computation, 2008.
- Ensembled Harmony Search: Geem ZW, Improved Harmony Search from Ensemble of Music Players, Lecture Notes in Artificial Intelligence, 2006.
- Continuous Harmony Search: Lee KS and Geem ZW, **A New Meta-Heuristic Algorithm for Continuous Engineering Optimization: Harmony Search Theory and Practice**, Computer Methods in Applied Mechanics and Engineering, 2005.
- Exploratory Power of Harmony Search: Das S, Mukhopadhyay A, Roy A, Abraham A, Panigrahi BK, **Exploratory Power of the Harmony Search Algorithm: Analysis and Improvements for Global Numerical Optimization**, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 41(1), 2011.
- Improved Harmony Search: Mahdavi M, Fesanghary M, and Damangir E, **An Improved Harmony Search Algorithm for Solving Optimization Problems**, Applied Mathematics and Computation, 2007.
- Particle-Swarm Harmony Search: Omran MGH and Mahdavi M, **Global-Best Harmony Search**, Applied Mathematics and Computation, 2008.

• Hybrid Harmony Search: Fesanghary M, Mahdavi M, Minary-Jolandan M, and Alizadeh Y, **Hybridizing Harmony Search Algorithm with Sequential Quadratic Programming for Engineering Optimization Problems**, Computer Methods in Applied Mechanics and Engineering, 2008.

• Parameter-Setting-Free Harmony Search: Geem ZW and Sim K-B, **Parameter-Setting-Free Harmony Search Algorithm**, Applied Mathematics and Computation, 2010.

• Multiobjective Harmony Search Algorithm Proposals: Juan Ricart, Germán Hüttemann, Joaquín Lima, Benjamín Barán. **Multiobjective Harmony Search Algorithm Proposals**, Electronic Notes in Theoretical Computer Science, 2011.

• Hybrid Harmony Search: HS-BFGS algorithm : Karahan H, Gurarslan G and Geem ZW, [doi:[http://dx.doi.org/10.1061/\(ASCE\)HE.1943-5584.0000608](http://dx.doi.org/10.1061/(ASCE)HE.1943-5584.0000608)] “Parameter Estimation of the nonlinear Muskingum flood routing model using a hybrid harmony search algorithm”, Journal of Hydrologic Engineering, doi:[10.1061/\(ASCE\)HE.1943-5584.0000608](http://dx.doi.org/10.1061/(ASCE)HE.1943-5584.0000608), 2012.

• Generalised Adaptive Harmony Search: Jaco Fourie, Richard Green, and Zong Woo Geem, **Generalised Adaptive Harmony Search: A Comparative Analysis of Modern Harmony Search**, Journal of Applied Mathematics, vol. 2013, Article ID 380985, 13 pages, 2013. doi:[10.1155/2013/380985](http://dx.doi.org/10.1155/2013/380985)

96.5.3 Applications in computer science

- Music Composition: Geem, Z. W. and Choi, J. Y. **Music Composition Using Harmony Search Algorithm**, Lecture Notes in Computer Science, 2007.
- Tetris Agent Optimization: Romero, V., Tomes, L., Yusong, J., **Harmonetris: Tetris Agent Optimization Using Harmony Search Algorithm**, International Journal of Computer Science Issues, 2011.
- Sudoku Puzzle: Geem, Z. W. **Harmony Search Algorithm for Solving Sudoku**, Lecture Notes in Artificial Intelligence, 2007.
- Tour Planning: Geem, Z. W., Tseng, C. -L., and Park, Y. **Harmony Search for Generalized Orienteering Problem: Best Touring in China**, Lecture Notes in Computer Science, 2005.
- Visual Tracking: J. Fourie, S. Mills and R. Green **Visual tracking using the harmony search algorithm**, Image and Vision Computing New Zealand, 2008. 23rd International Conference

- Visual Tracking: Jaco Fourie, Steven Mills, Richard Green, Harmony Filter: A Robust Visual Tracking System Using the Improved Harmony Search Algorithm, *Image and Vision Computing* (2010), doi:10.1016/j.imavis.2010.05.006
 - Visual Correspondence: J. Fourie, S. Mills and R. Green Directed correspondence search: Finding feature correspondences in images using the Harmony Search algorithm, *Image and Vision Computing New Zealand*, 23-25 Nov. 2009. 24th International Conference
 - Image Deconvolution: J. Fourie, S. Mills and R. Green Counterpoint Harmony Search: An accurate algorithm for the blind deconvolution of binary images, *Audio Language and Image Processing (ICALIP)*, 2010 International Conference on, Shanghai, China
 - Capacitated clustering 1: I. Landa-Torres, S. Gil-Lopez, S. Salcedo-Sanz, J. Del Ser, J. A. Portilla-Figueras, A Novel Grouping Harmony Search Algorithm for the Multiple-Type Access Node Location Problem, *Expert Systems with Applications*, vol. 39, no. 5, pp. 5262–5270, April 2012.
 - Capacitated clustering 2: I. Landa-Torres, J. Del Ser, S. Salcedo-Sanz, S. Gil-Lopez, J.A. Portilla-Figueras, O. Alonso-Garrido, A comparative study of two hybrid grouping evolutionary techniques for the capacitated P-median problem, *Computers and Operations Research*, vol. 39, no. 9, pp. 2214–2222, September 2012.
 - Design of radar codes: S. Gil-Lopez, J. Del Ser, S. Salcedo-Sanz, A. M. Perez-Bellido, J. M. Cabero and J. A. Portilla-Figueras, A Hybrid Harmony Search Algorithm for the Spread Spectrum Radar Polyphase Codes Design Problem, *Expert Systems with Applications*, Volume 39, Issue 12, pp. 11089–11093, September 2012.
 - Dynamic Spectrum Allocation: J. Del Ser, M. Matinmikko, S. Gil-Lopez and M. Mustonen, Centralized and Distributed Spectrum Channel Assignment in Cognitive Wireless Networks: A Harmony Search Approach, *Applied Soft Computing*, vol. 12, no. 2, pp. 921–930, February 2012.
 - Power and subcarrier allocation in OFDMA systems: J. Del Ser, M. N. Bilbao, S. Gil-Lopez, M. Matinmikko, S. Salcedo-Sanz, Iterative Power and Subcarrier Allocation in Rate-Constrained OFDMA Downlink Systems based on Harmony Search Heuristics, *Elsevier Engineering Applications of Artificial Intelligence*, Vol. 24, N. 5, pp. 748–756, August 2011.
 - Efficient design of open Wifi networks: I. Landa-Torres, S. Gil-Lopez, J. Del Ser, S. Salcedo-Sanz, D. Manjarres, J. A. Portilla-Figueras, Efficient City-wide Planning of Open WiFi Access Networks using Novel Grouping Harmony Search Heuristics, accepted for its publication in *Engineering Applications of Artificial Intelligence*, May 2012.
 - Single-objective localization: D. Manjarres, J. Del Ser, S. Gil-Lopez, M. Vecchio, I. Landa-Torres, R. Lopez-Valcarce, A Novel Heuristic Approach for Distance- and Connectivity-based Multi-hop Node Localization in Wireless Sensor Networks, *Springer Soft Computing*, accepted, June 2012.
 - Bi-objective localization: D. Manjarres, J. Del Ser, S. Gil-Lopez, M. Vecchio, I. Landa-Torres, S. Salcedo-Sanz, R. Lopez-Valcarce, On the Design of a Novel Two-Objective Harmony Search Approach for Distance- and Connectivity-based Node Localization in Wireless Sensor Networks, *Engineering Applications of Artificial Intelligence*, in press, June 2012.
- #### 96.5.4 Applications in engineering
- Fuzzy Data Clustering: Malaki, M.,Pourbaghery, JA, A Abolhassani, H. A combinatory approach to fuzzy clustering with harmony search and its applications to space shuttle data, *Proceedings of the SCIS & ISIS*,17–21,2008.
 - Structural Design: Lee, K. S. and Geem, Z. W. A New Structural Optimization Method Based on the Harmony Search Algorithm, *Computers & Structures*, 2004.
 - Structural Design: Saka, M. P. Optimum Geometry Design of Geodesic Domes Using Harmony Search Algorithm, *Advances in Structural Engineering*, 2007.
 - Water Network Design: Geem, Z. W. Optimal Cost Design of Water Distribution Networks using Harmony Search, *Engineering Optimization*, 2006.
 - Vehicle Routing: Geem, Z. W., Lee, K. S., and Park, Y. Application of Harmony Search to Vehicle Routing, *American Journal of Applied Sciences*, 2005.
 - Ground Water Modeling: Ayvaz, M. T. Simultaneous Determination of Aquifer Parameters and Zone Structures with Fuzzy C-Means Clustering and Meta-Heuristic Harmony Search Algorithm, *Advances in Water Resources*, 2007.
 - Soil Stability Analysis: Cheng, Y. M., Li, L., Lansivaara, T., Chi, S. C. and Sun, Y. J. An Improved Harmony Search Minimization Algorithm Using Different Slip Surface Generation Methods for Slope Stability Analysis, *Engineering Optimization*, 2008.

- Energy System Dispatch: Vasebi, A., Fesanghary, M., and Bathaeaa, S.M.T. Combined Heat and Power Economic Dispatch by Harmony Search Algorithm, International Journal of Electrical Power & Energy Systems, 2007.
- Offshore Structure Mooring: Ryu, S., Duggal, A.S., Heyl, C. N., and Geem, Z. W. Mooring Cost Optimization via Harmony Search, Proceedings of the 26th International Conference on Offshore Mechanics and Arctic Engineering (OMAE 2007), ASME, San Diego, CA, USA, June 10–15, 2007.
- Hydrologic Parameter Calibration: Kim, J. H., Geem, Z. W., and Kim, E. S. Parameter Estimation of the Nonlinear Muskingum Model using Harmony Search, Journal of the American Water Resources Association, 2001.
- Hydrologic Parameter Calibration: Karahan, H, Gurarslan, G. and Geem, Z.W.[doi:[http://dx.doi.org/10.1061/\(ASCE\)HE.1943-5584.0000608](http://dx.doi.org/10.1061/(ASCE)HE.1943-5584.0000608)] “Parameter Estimation of the nonlinear Muskingum flood routing model using a hybrid harmony search algorithm”, Journal of Hydrologic Engineering, doi:[10.1061/\(ASCE\)HE.1943-5584.0000608](http://dx.doi.org/10.1061/(ASCE)HE.1943-5584.0000608), 2012.
- Satellite Heat Pipe Design: Geem, Z. W. and Hwangbo, H. Application of Harmony Search to Multi-Objective Optimization for Satellite Heat Pipe Design, Proceedings of US-Korea Conference on Science, Technology, & Entrepreneurship (UKC 2006), CD-ROM, Teaneck, NJ, USA, Aug. 10-13 2006.
- Dam Scheduling: Geem, Z. W. Optimal Scheduling of Multiple Dam System Using Harmony Search Algorithm, Lecture Notes in Computer Science, 2007.
- Ecological Conservation: Geem, Z. W. and Williams, J. C. Ecological Optimization Using Harmony Search, Proceedings of American Conference on Applied Mathematics, Harvard University, Cambridge, MA, USA, March 24–26, 2008.
- Heat exchanger design: Fesanghary, M., Damangir, E. and Soleimani, I. Design optimization of shell and tube heat exchangers using global sensitivity analysis and harmony search, Applied Thermal Engineering, In press.
- Heat exchanger design: Doodman, A., Fesanghary, M. and Hosseini, R. A robust stochastic approach for design optimization of air cooled heat exchangers, Applied Energy, In press.
- Heat exchanger network design: Khorasani, R.M., Fesanghary, M. A novel approach for synthesis of cost-optimal heat exchanger networks, Computers and Chemical Engineering, In press.
- Face milling: Zarei, O., Fesanghary, M., Farshi, B., Jalili Saffar, R. and Razfar, M.R. Optimization of multi-pass face-milling via harmony search algorithm, Journal of Materials Processing Technology, In press.
- Document Clustering:, Mahdavi., M., Chehreganinia, H., Abolhassania, H., Forsati, R. Novel metaheuristic algorithms for document clustering, AMC Journal
- Multicast Routing: Forsat, R., Haghigat, M., Mahdavi, M., Harmony search based algorithms for bandwidth-delay-constrained least-cost multicast routing, Computer Communications, Elsevier
- AYVAZ, M.T. and GENÇ, Ö., Optimal estimation of Manning’s roughness in open channel flows using a linked simulation-optimization model, BALWOIS 2012, International Conference on Water, Climate and Environment, May 28 - June 2, 2012, Ohrid, Macedonia.
- Poursalehi, N., Zolfaghari,A., Minuchehr, A., PWR loading pattern optimization using Harmony Search algorithm, Ann. Nucl. Energy, 2013, Vol. 53, pp. 288-298.

96.5.5 Applications in economics

- I. Landa-Torres, E. G. Ortiz-Garcia, S. Salcedo-Sanz, M. J. Segovia, S. Gil-Lopez, M. Miranda, J. M. Leiva-Murillo, J. Del Ser, Evaluating the Internationalization Success of Companies using a Hybrid Grouping Harmony Search - Extreme Learning Machine Approach, IEEE Journal on Selected Topics in Signal Processing, Vol. PP., N. 99 (early access), May 2012.

96.6 Source codes

- Improved Harmony Search (MATLAB)
- Hybrid HS-SQP (Visual C++)
- Multiobjective Harmony Search (C#)
- Other HS Variants
- Multiobjective Harmony Search Algorithm Proposals (C++)
- pyHarmonySearch (Python)

Chapter 97

Imperialist competitive algorithm

In computer science, **Imperialist Competitive Algorithm (ICA)**^[1] is a computational method that is used to solve optimization problems of different types. Like most of the methods in the area of evolutionary computation, ICA does not need the gradient of the function in its optimization process.

From a specific point of view, ICA can be thought of as the social counterpart of genetic algorithms (GAs). ICA is the mathematical model and the computer simulation of human social evolution, while GAs are based on the biological evolution of species.

97.1 Algorithm

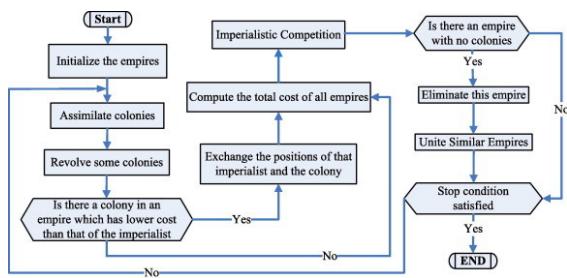


Figure 1: Flowchart of Imperialist Competitive Algorithm (ICA)

Figure 1 shows the flowchart of the Imperialist Competitive Algorithm. This algorithm starts by generating a set of candidate random solutions in the search space of the optimization problem. The generated random points are called the initial *Countries*. Countries in this algorithm are the counterpart of *Chromosomes* in GAs and *Particles* in Particle Swarm Optimization (PSO) and it is an array of values of a candidate solution of optimization problem. The cost function of the optimization problem determines the power of each country. Based on their power, some of the best initial countries (the countries with the least cost function value), become *Imperialists* and start taking control of other countries (called *colonies*) and form the initial *Empires*.^[1]

Two main operators of this algorithm are *Assimilation* and *Revolution*. Assimilation makes the colonies of each empire get closer to the imperialist state in

the space of socio-political characteristics (optimization search space). Revolution brings about sudden random changes in the position of some of the countries in the search space. During assimilation and revolution a colony might reach a better position and has the chance to take the control of the entire empire and replace the current imperialist state of the empire.^[2]

Imperialistic Competition is another part of this algorithm. All the empires try to win this game and take possession of colonies of other empires. In each step of the algorithm, based on their power, all the empires have a chance to take control of one or more of the colonies of the weakest empire.^[1]

Algorithm continues with the mentioned steps (Assimilation, Revolution, Competition) until a stop condition is satisfied.

97.2 Pseudocode

The above steps can be summarized as the below pseudocode.^[2]

- 0) Define objective function: $f(\mathbf{x})$, $\mathbf{x} = (x_1, x_2, \dots, x_d)$;
- 1) Initialization of the algorithm. Generate some random solution in the search space and create initial empires.
- 2) Assimilation: Colonies move towards imperialist states in different directions.
- 3) Revolution: Random changes occur in the characteristics of some countries.
- 4) Position exchange between a colony and Imperialist. A colony with a better position than the imperialist, has the chance to take the control of empire by replacing the existing imperialist.
- 5) Imperialistic competition: All imperialists compete to take possession of colonies of each other.
- 6) Eliminate the powerless empires. Weak empires lose their power gradually and they will finally be eliminated.
- 7) If the stop condition is satisfied, stop, if not go to 2.
- 8) End

97.3 Variants

Like for PSO, the first version of ICA was proposed for solving continuous optimization problems. Then in other

works different variants of ICA were proposed for solving both discrete and continuous problems. For example Chaotic ICA is proposed by Duan, et.al.^[3] and also a version of this algorithm for handling constrained optimization problems is proposed by Zhang, et.al.^[4]

97.4 Applications

ICA is used to solve different optimization problems in various areas of engineering and science. The following are some of the applications of this algorithm.

- Designing controller for industrial systems^{[5][6][7][8][9]}
- Designing Intelligent Recommender Systems^[10]
- Solving optimization problems in communication systems.^{[11][12][13]}
- Solving scheduling and production management problems^{[2][14][15][16][17][18][19]}
- Training and analysis of Artificial Neural Networks^{[20][21]}
- Nash Equilibrium Point Achievement^[8]
- Design and thermodynamic optimization of plate-fin heat exchangers^[22]
- Feature selection^{[23][24]}
- and so on^{[25][26][27][28]}

97.5 References

- [1] Atashpaz-Gargari, E.; Lucas, C (2007). “Imperialist Competitive Algorithm: An algorithm for optimization inspired by imperialistic competition”. *IEEE Congress on Evolutionary Computation* **7**. pp. 4661–4666.
- [2] Nazari-Shirkouhi, S.; Eivazy, H.; Ghodsi, R.; Rezaie, K.; Atashpaz-Gargari, E. (2010). “Solving the Integrated Product Mix-Outsourcing Problem by a Novel Meta-Heuristic Algorithm: Imperialist Competitive Algorithm”. *Expert Systems with Applications* **37** (12): 7615–7626. doi:10.1016/j.eswa.2010.04.081.
- [3] Duan, H.; Xu, C.; Liu, S.; Shao, S. (2009). “Template matching using chaotic imperialist competitive algorithm”. *Pattern Recognition Letters* **39** (6): 1362–1381.
- [4] Zhang, Yang; Wang, Yong; Peng, Cheng (2009). “Improved Imperialist Competitive Algorithm for Constrained Optimization”. *Computer Science-Technology and Applications, IFCSTA*.
- [5] Rajabioun, R.; Hashemzadeh, F.; Atashpaz-Gargari, E.; Mesgari, B.; Rajaei Salmasi, F. (2008). “Identification of a MIMO evaporator and its decentralized PID controller tuning using Colonial Competitive Algorithm”. In the proceeding of *IFAC World Congress, Seoul, Korea*. pp. 9952–9957.
- [6] Atashpaz-Gargari, E.; Hashemzadeh, F.; Rajabioun, R.; Lucas, C. (2008). “Colonial competitive algorithm: A novel approach for PID controller design in MIMO distillation column process”. *International Journal of Intelligent Computing and Cybernetics* **1** (3): 337–355. doi:10.1108/17563780810893446.
- [7] Atashpaz-Gargari, E.; Lucas, C. (2007). “Designing an optimal PID controller using Colonial Competitive Algorithm”. *Proceeding of First Iranian Joint Congress on Intelligent and Fuzzy Systems*.
- [8] Rajabioun, R.; Atashpaz-Gargari, E.; Lucas, C. (2008). “Colonial competitive algorithm as a tool for Nash equilibrium point achievement”. *Computational Science and Its Applications–ICCSA*. pp. 680–695.
- [9] Atashpaz-Gargari, E.; Rajabioun, R.; Hashemzadeh, F.; R. Salmasi, F. (2009). “A Decentralized PID controller based on Optimal Shrinkage of Gershgorin Bands and PID tuning Using Colonial Competitive Algorithm”. *International Journal of Innovative Computing, Information and Control* **5** (10(A)).
- [10] Sepehri Rad, H.; Lucas, C. (2008). “Application of Imperialistic Competition Algorithm in recommender systems”. In *13th Int'l CSI Computer Conference (CSICC'08)*.
- [11] Khabbazi, Arash; Atashpaz-Gargari, Esmaeil; Lucas, Caro (2009). “Imperialist competitive algorithm for minimum bit error rate beamforming”. *International Journal of Bio-Inspired Computation* **1** (1-2): 125–133. doi:10.1504/ijbic.2009.022781.
- [12] Alikhani koupaei, Javad; Abdechiri, Marjan (2010). “An Optimization Problem for Evaluation of Image Segmentation Methods”. *International Journal of Computer and Network Security* **2** (6).
- [13] H. Sayadnavard, Monireh; T. Haghighat, Abolfazl; Abdechiri, Marjan (2010). “Wireless sensor network localization using Imperialist Competitive Algorithm”. *3rd IEEE International Conference on Computer Science and Information Technology*.
- [14] Jolai, F.; Sangari, M.; Babaie, M. (2010). “Pareto simulated annealing and colonial competitive algorithm to solve an offline scheduling problem with rejection”. *Journal of Engineering Manufacture* **224** (7): 1119–1131. doi:10.1243/09544054jem1746.
- [15] Shokrollahpour, E.; Zandieh, M.; Dorri, Behrouz (2010). “A novel imperialist competitive algorithm for bi-criteria scheduling of the assembly flowshop problem”. *International Journal of Production Research*.
- [16] Forouharfarid, S.; Zandieh, M. (2010). “An imperialist competitive algorithm to schedule of receiving and shipping trucks in cross-docking systems”. *International Journal of Advanced Manufacturing Technology*.

- [17] Karimi, N.; Zandieh, M. (2010). "Group scheduling in flexible flow shops: A hybridised approach of imperialist competitive algorithm and electromagnetic-like mechanism". *International Journal of Production Research*. doi:10.1080/00207543.2010.481644.
- [18] Bagher, M.; Zandieh, M.; Farsijani, H. (2010). "Balancing of stochastic U-type assembly lines: an imperialist competitive algorithm". *International Journal of Advanced Manufacturing Technology*.
- [19] Sarayloo, Fatemeh; Tavakkoli-Moghaddam, Reza (2010). "Imperialistic Competitive Algorithm for Solving a Dynamic Cell Formation Problem with Production Planning". *Advanced Intelligent Computing Theories and Applications, Lecture Notes in Computer Science* **6215**: 266–276. doi:10.1007/978-3-642-14922-1_34.
- [20] Biabangard-Oskouyi, A.; Atashpaz-Gargari, E.; Soltani, N.; Lucas, C. (2009). "Application of Imperialist Competitive Algorithm for Material Properties Characterization from Sharp Indentation Test". *Int J Eng Simul* **10** (1).
- [21] T., Maryam; F., Nafiseh; L., Caro; T., Fattaneh (2009). "Artificial Neural Network Weights Optimization based on Imperialist Competitive Algorithm". *Seventh International Conference on Computer Science and Information Technologies*.
- [22] Yousefi, Moslem; Mohammadi, Hossein (2011). "Second law based optimization of a plate fin heat exchanger using Imperialist Competitive Algorithm". *International Journal of the Physical Sciences* **6**. pp. 4749–4759.
- [23] Seyed Jalaleddin Mousavi Rad, Fardin Akhlaghian Tab, Kaveh Mollazade, " Application of Imperialist Competition Algorithm for Feature Selection: A Case Study On Rice Classification", International Journal of Computer Application, Vol. 40, No.16, 2012
- [24] S.J. Mousavirad, H. Ebrahimpour, Feature selection using modified Imperilaist Competitive Algorithm, International Conference on Computer and Knowledge Engineering(ICKKE 2013), October 31 & November 1, 2013, Ferdowsi University of Mashhad, Mashhad, Iran
- [25] Lucas, C.; Nasiri-Gheidari, Z.; Tootoonchian, F. (2010). "Application of an imperialist competitive algorithm to the design of a linear induction motor". *Energy Conversion and Management* **51** (7): 1407–1411. doi:10.1016/j.enconman.2010.01.014.
- [26] Movahhedi, Omid; R. Salmasi, Farzad (2009). "Optimal Design of Propulsion System with Adaptive Fuzzy Controller for a PHEV Based on Non-dominated Sorting Genetic and Colonial Competitive Algorithms". *International Review of Automatic Control*.
- [27] Niknam, Taher; Taherian Fard, Elahe; Pourjafarian, Narges; Rousta, Alireza (2010). "An efficient hybrid algorithm based on modified imperialist competitive algorithm and K-means for data clustering". *Engineering Applications of Artificial Intelligence*.
- [28] Mozafari, Hamid; Abdi, Behzad; Ayob, Amran (2010). "Optimization of Transmission Conditions for Thin Interphase Layer Based on Imperialist Competitive Algorithm". *International Journal on Computer Science and Engineering* **2** (7): 2486–2490.

Chapter 98

Intelligent Water Drops algorithm

Intelligent Water Drops algorithm, or the IWD algorithm,^[1] is a swarm-based nature-inspired optimization algorithm. This algorithm contains a few essential elements of natural water drops and actions and reactions that occur between river's bed and the water drops that flow within. The IWD algorithm may fall into the category of **Swarm intelligence** and **Metaheuristic**. Intrinsically, the IWD algorithm can be used for **Combinatorial optimization**. However, it may be adapted for continuous optimization too. The IWD was first introduced for the **traveling salesman problem** in 2007.^[2] Since then, multitude of researchers have focused on improving the algorithm for different problems.

98.1 Introduction

Almost every IWD algorithm is composed of two parts: a graph that plays the role of distributed memory on which soils of different edges are preserved, and the moving part of the IWD algorithm, which is a few number of Intelligent water drops. These Intelligent Water Drops (IWDs) both compete and cooperate to find better solutions and by changing soils of the graph, the paths to better solutions become more reachable. It is mentioned that the IWD-based algorithms need at least two IWDs to work.

98.2 Pseudo-code

The IWD algorithm has two types of parameters: Static and Dynamic parameters. Static parameters are constant during the process of the IWD algorithm. Dynamic parameters are reinitialized after each iteration of the IWD algorithm. The pseudo-code of an IWD-based algorithm may be specified in eight steps:

- 1) Static parameter initialization
 - a) *Problem representation in the form of a graph*
 - b) *Setting values for static parameters*

- 2) Dynamic parameter initialization: soil and velocity of IWDs
- 3) Distribution of IWDs on the problem's graph
- 4) Solution construction by IWDs along with soil and velocity updating
 - a) *Local soil updating on the graph*
 - b) *Soil and velocity updating on the IWDs*
- 5) Local search over each IWD's solution (optional)
- 6) Global soil updating
- 7) Total-best solution updating
- 8) Go to step 2 unless termination condition is satisfied

98.3 Applications

Some of the researches performed with the IWD-based algorithms for different applications are given below:

- Multidimensional **Knapsack problem** (MKP)^[3]
- Air Robot Path Planning^[4]
- **Vehicle routing problem**^[5]
- MANET Routing algorithm^[6]
- Economic Load Dispatch^[7]
- Travelling salesman problem (TSP)^[8]
- texture feature selection^[9]
- Automatic multilevel thresholding using a modified Otsu's criterion^[10]
- Continuous optimization^[11]
- Job shop scheduling^[12]
- Steiner tree problem^[13]

- Maximum Clique problem [14]
- Optimal data aggregation tree in wireless sensor networks [15]
- Test data generation based on test path discovery [16]
- Code coverage [17]
- Optimization of manufacturing process models [18]
- Optimizing routing protocol [19]
- Rough set feature selection [20]

98.4 See also

- Swarm Intelligence

98.5 References

- [1] Shah-Hosseini, H. (2009). "The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm". *International Journal of Bio-Inspired Computation* **1** (1/2): 71–79. doi:10.1504/ijbic.2009.022775.
- [2] problem solving by intelligent water drops
- [3] Shah-Hosseini, H. (2008). "Intelligent water drops algorithm: a new optimization method for solving the multiple knapsack problem". *Int. Journal of Intelligent Computing and Cybernetics* **1** (2): 193–212. doi:10.1108/17563780810874717.
- [4] Duan et al. (2009). "Novel intelligent water drops optimization approach to single UCAV smooth path planning". *Aerospace Science and Technology* **13** (8): 442–449. doi:10.1016/j.ast.2009.07.002.
- [5] Kamkar et al. (2010). "Intelligent water drops a new optimization algorithm for solving the Vehicle Routing Problem". *IEEE International Conference on Systems, Man and Cybernetics*: 4142–4146.
- [6] Fan et al. (2010). "The Intelligent-Water-Drop Based Routing algorithm for MANET". *Int. Conf. on Future Information Technology*: 253–255.
- [7] Rayapudi, S. R. (2011). "An intelligent water drop algorithm for economic load dispatch". *International Journal of Electrical and Electronics Engineering* **5** (1): 43–49.
- [8] Msallam et al. (2011). "Improved intelligent water drops algorithm using adaptive schema". *International Journal of Bio-Inspired Computation* **3** (2): 103–111. doi:10.1504/ijbic.2011.039909.
- [9] Hendrawan et al. (2011). "Neural-Intelligent Water Drops algorithm to select relevant textural features for developing precision irrigation system using machine vision". *Computers and Electronics in Agriculture* **77** (2): 214–228. doi:10.1016/j.compag.2011.05.005.
- [10] Shah-Hosseini, H. (2012). "Intelligent Water Drops algorithm for automatic multilevel thresholding of gray-level images using a modified Otsu's criterion". *Int. J. of Modelling, Identification and Control* **15** (4): 241–249. doi:10.1504/ijmic.2012.046402.
- [11] Shah-Hosseini, H. (2012). "An approach to continuous optimization by the Intelligent Water Drops algorithm". *Procedia - Social and Behavioral Sciences* **32**: 224–229. doi:10.1016/j.sbspro.2012.01.033.
- [12] Niu et al. (2012). "An improved Intelligent Water Drops algorithm for achieving optimal job-shop scheduling solutions". *International Journal of Production Research* **50** (15): 4195–4205. doi:10.1080/00207543.2011.600346.
- [13] Noferesti et al. (2012). "A Hybrid Algorithm for Solving Steiner Tree Problem". *International Journal of Computer Applications* **41** (5): 14–20. doi:10.5120/5536-7584.
- [14] al-Taani et al. (2012). "SOLVING THE MAXIMUM CLIQUE PROBLEM USING INTELLIGENT WATER DROPS ALGORITHM". *The International Conference on Computing, Networking and Digital Technologies (IC-NDT2012)*: 142–151.
- [15] Hoang et al. (2012). "Optimal data aggregation tree in wireless sensor networks based on intelligent water drops algorithm". *IET Wireless Sensor Systems* **2** (3): 282–292. doi:10.1049/iet-wss.2011.0146.
- [16] Srivastava et al. (2012). "Test Data Generation Based on Test Path Discovery Using Intelligent Water Drop". *International journal of applied metaheuristic computing* **3** (2).
- [17] agarwal et al. (2012). "Code coverage using intelligent water drop (IWD)". *International Journal of Bio-Inspired Computation* **4** (6): 392–402. doi:10.1504/ijbic.2012.051396.
- [18] Luangpaiboon, P. (2012). "Optimisation of Manufacturing Process Models via Intelligent Water Drop Algorithm". *Applied Mechanics and Materials*. 217–219: 1501–1505. doi:10.4028/www.scientific.net/amm.217-219.1501.
- [19] Khaleel et al. (2013). "Using intelligent water drops algorithm for optimisation routing protocol in mobile ad-hoc networks". *International Journal of Reasoning-based Intelligent Systems* **4** (4): 227–234. doi:10.1504/ijris.2012.051724.
- [20] Alijla et al. (2013). "Intelligent Water Drops Algorithm for Rough Set Feature Selection". *Intelligent Information and Database systems* **7803**: 356–365. doi:10.1007/978-3-642-36543-0_37.

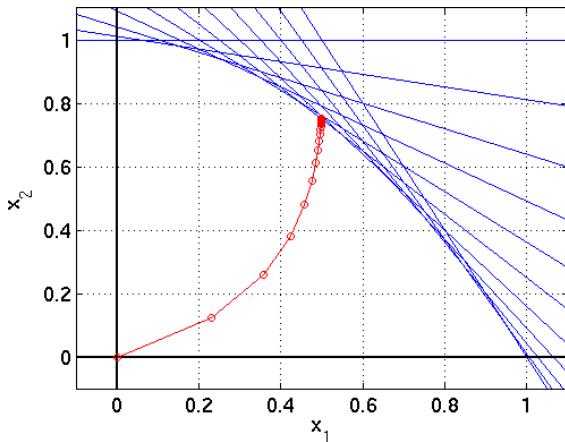
98.6 External links

- Source code of the IWD algorithm for the TSP using C# language
- A community page for IWD algorithm discussion

Chapter 99

Interior point method

Interior point methods (also referred to as **barrier methods**) are a certain class of algorithms to solve linear and nonlinear **convex optimization** problems.



Example solution

John von Neumann^[1] suggested an interior point method of linear programming which was neither a polynomial time method nor an efficient method in practice. In fact, it turned out to be slower in practice compared to simplex method which is not a polynomial time method. In 1984, Narendra Karmarkar developed a method for linear programming called **Karmarkar's algorithm** which runs in provably polynomial time and is also very efficient in practice. It enabled solutions of linear programming problems which were beyond the capabilities of **simplex method**. Contrary to the simplex method, it reaches a best solution by traversing the interior of the **feasible region**. The method can be generalized to convex programming based on a **self-concordant barrier function** used to encode the **convex set**.

Any convex optimization problem can be transformed into minimizing (or maximizing) a linear function over a convex set by converting to the **epigraph form**.^[2] The idea of encoding the feasible set using a barrier and designing barrier methods was studied by Anthony V. Fiacco, Garth P. McCormick, and others in the early 1960s. These ideas were mainly developed for general **nonlinear programming**, but they were later abandoned due to the presence of more competitive methods for this class of

problems (e.g. sequential quadratic programming).

Yurii Nesterov and Arkadi Nemirovski came up with a special class of such barriers that can be used to encode any convex set. They guarantee that the number of iterations of the algorithm is bounded by a polynomial in the dimension and accuracy of the solution.^[3]

Karmarkar's breakthrough revitalized the study of interior point methods and barrier problems, showing that it was possible to create an algorithm for linear programming characterized by **polynomial complexity** and, moreover, that was competitive with the simplex method. Already Khachiyan's ellipsoid method was a polynomial time algorithm; however, it was too slow to be of practical interest.

The class of primal-dual path-following interior point methods is considered the most successful. Mehrotra's predictor-corrector algorithm provides the basis for most implementations of this class of methods.^[4]

99.1 Primal-dual interior point method for nonlinear optimization

The primal-dual method's idea is easy to demonstrate for constrained **nonlinear optimization**. For simplicity consider the all-inequality version of a nonlinear optimization problem:

$$\begin{aligned} & \text{minimize } f(x) \text{ subject to } c_i(x) \geq 0 \text{ for } i = \\ & 1, \dots, m, \quad x \in \mathbb{R}^n, \text{ where } f : \mathbb{R}^n \rightarrow \mathbb{R}, c_i : \\ & \mathbb{R}^n \rightarrow \mathbb{R} \end{aligned} \quad (1)$$

The logarithmic **barrier function** associated with (1) is

$$B(x, \mu) = f(x) - \mu \sum_{i=1}^m \ln(c_i(x)) \quad (2)$$

Here μ is a small positive scalar, sometimes called the “barrier parameter”. As μ converges to zero the minimum of $B(x, \mu)$ should converge to a solution of (1).

The barrier function gradient is

$$g_b = g - \mu \sum_{i=1}^m \frac{1}{c_i(x)} \nabla c_i(x) \quad (3)$$

where g is the gradient of the original function $f(x)$ and ∇c_i is the gradient of c_i .

In addition to the original (“primal”) variable x we introduce a **Lagrange multiplier** inspired **dual** variable $\lambda \in \mathbb{R}^m$

$$\forall_{i=1}^m c_i(x) \lambda_i = \mu \quad (4)$$

(4) is sometimes called the “perturbed complementarity” condition, for its resemblance to “complementary slackness” in KKT conditions.

We try to find those (x_μ, λ_μ) for which the gradient of the barrier function is zero.

Applying (4) to (3) we get an equation for the gradient:

$$g - A^T \lambda = 0 \quad (5)$$

where the matrix A is the constraint $c(x)$ Jacobian.

The intuition behind (5) is that the gradient of $f(x)$ should lie in the subspace spanned by the constraints’ gradients. The “perturbed complementarity” with small μ (4) can be understood as the condition that the solution should either lie near the boundary $c_i(x) = 0$ or that the projection of the gradient g on the constraint component $c_i(x)$ normal should be almost zero.

Applying **Newton’s method** to (4) and (5) we get an equation for (x, λ) update (p_x, p_λ) :

$$\begin{pmatrix} W & -A^T \\ \Lambda A & C \end{pmatrix} \begin{pmatrix} p_x \\ p_\lambda \end{pmatrix} = \begin{pmatrix} -g + A^T \lambda \\ \mu 1 - C \lambda \end{pmatrix}$$

where W is the **Hessian matrix** of $f(x) - \lambda_1 c_1 - \dots - \lambda_m c_m$ and Λ is a diagonal matrix of λ and C is a diagonal matrix where C_{ii} is $c_i(x)$.

Because of (1), (4) the condition

$$\lambda \geq 0$$

should be enforced at each step. This can be done by choosing appropriate α :

$$(x, \lambda) \rightarrow (x + \alpha p_x, \lambda + \alpha p_\lambda)$$

99.2 See also

- Augmented Lagrangian method
- Penalty method
- Karush–Kuhn–Tucker conditions

99.3 References

- [1] Dantzig, George B.; Thapa, Mukund N. (2003). *Linear Programming 2: Theory and Extensions*. Springer-Verlag.
- [2] Boyd, Stephen; Vandenberghe, Lieven (2004). *Convex Optimization*. Cambridge: Cambridge University Press. p. 143. ISBN 0-521-83378-7. MR 2061575.
- [3] Wright, Margaret H. (2004). “The interior-point revolution in optimization: History, recent developments, and lasting consequences”. *Bulletin of the American Mathematical Society* **42**: 39. doi:10.1090/S0273-0979-04-01040-7. MR 2115066.
- [4] Potra, Florian A.; Stephen J. Wright (2000). “Interior-point methods”. *Journal of Computational and Applied Mathematics* **124** (1–2): 281–302. doi:10.1016/S0377-0427(00)00433-7.

99.4 Bibliography

- Bonnans, J. Frédéric; Gilbert, J. Charles; Lemaréchal, Claude; Sagastizábal, Claudia A. (2006). *Numerical optimization: Theoretical and practical aspects*. Universitext (Second revised ed. of translation of 1997 French ed.). Berlin: Springer-Verlag. pp. xiv+490. doi:10.1007/978-3-540-35447-5. ISBN 3-540-35445-X. MR 2265882.
- Karmarkar, N. (1984). “Proceedings of the sixteenth annual ACM symposium on Theory of computing - STOC ‘84”. p. 302. doi:10.1145/800057.808695. ISBN 0-89791-133-4. lchapter= ignored (help)
- Mehrotra, Sanjay (1992). “On the Implementation of a Primal-Dual Interior Point Method”. *SIAM Journal on Optimization* **2** (4): 575. doi:10.1137/0802028.
- Nocedal, Jorge; Stephen Wright (1999). *Numerical Optimization*. New York, NY: Springer. ISBN 0-387-98793-2.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). “Section 10.11. Linear Programming: Interior-Point Methods”. *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.

- Wright, Stephen (1997). *Primal-Dual Interior-Point Methods*. Philadelphia, PA: SIAM. ISBN 0-89871-382-X.
- Boyd, Stephen; Vandenberghe, Lieven (2004). *Convex Optimization*. Cambridge University Press.

Chapter 100

Interval contractor

In mathematics, an **interval contractor** (or **contractor** for short) [1] associated to a set X is an operator C which associates to a box $[x]$ in \mathbf{R}^n another box $C([x])$ of \mathbf{R}^n such that the two following properties are always satisfied

- $C([x]) \subset [x]$ (contractance property)
- $C([x]) \cap X = [x] \cap X$ (completeness property)

A *contractor associated to a constraint* (such as an equation or an inequality) is a contractor associated to the set X of all x which satisfy the constraint. Contractors make it possible to improve the efficiency of branch-and-bound algorithms classically used in *interval analysis*.

100.1 Properties of contractors

A contractor C is *monotonic* if we have $[x] \subset [y] \Rightarrow C([x]) \subset C([y])$

It is *minimal* if for all boxes $[x]$, we have $C([x]) = [[x] \cap X]$, where $[A]$ is the *interval hull* of the set A , i.e., the smallest box enclosing A .

The contractor C is *thin* if for all points x , $C(\{x\}) = \{x\} \cap X$ where $\{x\}$ denotes the degenerated box enclosing x as a single point.

The contractor C is *idempotent* if for all boxes $[x]$, we have $C \circ C([x]) = C([x])$.

The contractor C is *convergent* if for all sequences $[x](k)$ of boxes containing x , we have $[x](k) \rightarrow x \Rightarrow C([x](k)) \rightarrow \{x\} \cap X$.

100.2 Illustration

Figure 1 represents the set X painted grey and some boxes. Some of them are degenerated, i.e., they correspond to singletons. Figure 2 represents these boxes after contraction. Note that no point of X has been removed by the contractor. The contractor is minimal for the cyan box but is pessimistic for the green one. All degenerated blue boxes are contracted to the empty box. The magenta box and the red box cannot be contracted.

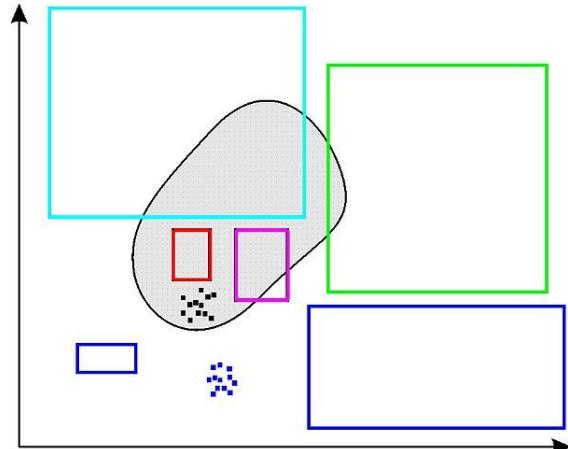


Figure 1: Boxes before contraction

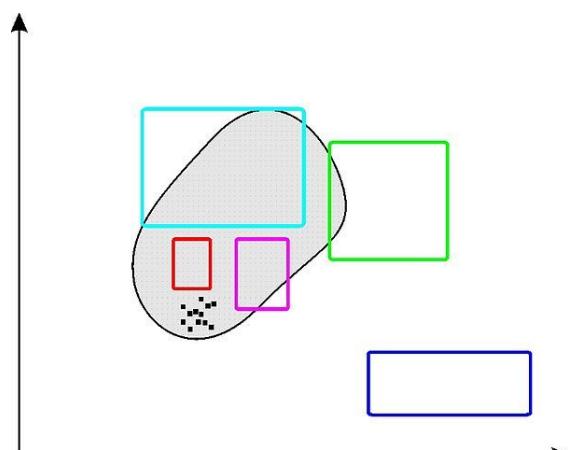


Figure 2: Boxes after contraction

100.3 Contractor algebra

Some operations can be performed on contractors to build more complex contractors. [2] The intersection, the union, the composition and the repetition are defined as follows.

$$(C_1 \cap C_2)([x]) = C_1([x]) \cap C_2([x])$$

$$(C_1 \cup C_2)([x]) = [C_1([x]) \cup C_2([x])]$$

$$(C_1 \circ C_2)([x]) = C_1(C_2([x]))$$

$$C^\infty([x]) = C \circ C \circ C \circ \dots \circ C([x])$$

100.4 Building contractors

There exist different ways to build contractors associated to equations and inequalities, say, $f(x)$ in $[y]$. Most of them are based on interval arithmetic. One of the most efficient and most simple is the *forward/backward contractor* (also called as HC4-revise). [3] [4]

The principle is to evaluate $f(x)$ using interval arithmetic (this is the forward step). The resulting interval is intersected with $[y]$. A backward evaluation of $f(x)$ is then performed in order to contract the intervals for the x_i (this is the backward step). We now illustrate the principle on a simple example.

Consider the constraint $(x_1 + x_2) \cdot x_3 \in [1, 2]$. We can evaluate the function $f(x)$ by introducing the two intermediate variables a and b , as follows

$$a = x_1 + x_2$$

$$b = a \cdot x_3$$

The two previous constraints are called *forward constraints*. We get the *backward constraints* by taking each forward constraint in the reverse order and isolating each variable on the right hand side. We get

$$x_3 = \frac{b}{a}$$

$$a = \frac{b}{x_3}$$

$$x_1 = a - x_2$$

$$x_2 = a - x_1$$

The resulting forward/backward contractor $C([x_1], [x_2], [x_3])$ is obtained by evaluating the forward and the backward constraints using interval analysis.

$$[a] = [x_1] + [x_2]$$

$$[b] = [a] \cdot [x_3]$$

$$[b] = [b] \cap [1, 2]$$

$$[x_3] = [x_3] \cap \frac{[b]}{[a]}$$

$$[a] = [a] \cap \frac{[b]}{[x_3]}$$

$$[x_1] = [x_1] \cap [a] - [x_2]$$

$$[x_2] = [x_2] \cap [a] - [x_1]$$

100.5 References

- [1] Jaulin, Luc; Kieffer, Michel; Didrit, Olivier; Walter, Eric (2001). *Applied Interval Analysis*. Berlin: Springer. ISBN 1-85233-219-0.
- [2] Chabert, G.; Jaulin, L. (2009). “Contractor programming”. *Artificial Intelligence* **173**.
- [3] Messine, F. (1997). *Méthode d'optimisation globale basée sur l'analyse d'intervalles pour la résolution de problèmes avec contraintes*. Thèse de doctorat, Institut National Polytechnique de Toulouse.
- [4] Benhamou, F.; Goualard, F.; Granvilliers, L.; Puget, J.F. (1999). *Revising hull and box consistency*. In Proceedings of the 1999 international conference on Logic programming.

Chapter 101

IOSO

IOSO (Indirect Optimization on the basis of Self-Organization) is a multiobjective, multidimensional nonlinear optimization technology.

101.1 IOSO approach

IOSO Technology is based on the response surface methodology approach. At each IOSO iteration the internally constructed response surface model for the objective is being optimized within the current search region. This step is followed by a direct call to the actual mathematical model of the system for the candidate optimal point obtained from optimizing internal response surface model. During IOSO operation, the information about the system behavior is stored for the points in the neighborhood of the extremum, so that the response surface model becomes more accurate for this search area. The following steps are internally taken while proceeding from one IOSO iteration to another:

- the modification of the experiment plan;
- the adaptive adjustment of the current search area;
- the function type choice (global or middle-range) for the response surface model;
- the adjustment of the response surface model;
- the modification of both parameters and structure of the optimization algorithms; if necessary, the selection of the new promising points within the search area.

101.2 History

IOSO is based on the technology being developed for more than 20 years by **Sigma Technology** which grew out of IOSO Technology Center in 2001. Sigma Technology is headed by prof . Egorov I. N., CEO.

101.3 Products

IOSO is the name of the group of multidisciplinary design optimization software that runs on Microsoft Windows as well as on Unix/Linux OS and was developed by **Sigma Technology**. It is used to improve the performance of complex systems and technological processes and to develop new materials based on a search for their optimal parameters. IOSO is easily integrated with almost any computer aided engineering (CAE) tool.

IOSO group of software consists of:

- IOSO NM: Multi-objective optimization;
- IOSO PM: Parallel multi-objective optimization;
- IOSO LM: Multilevel multi-objective optimization with adaptive change of the object model fidelity (low-, middle-, high fidelity models);
- IOSO RM: Robust design optimization and robust optimal control software;

101.4 Purpose

101.4.1 Performance improvement and design optimisation

IOSO NM is used to maximise or minimise system or object characteristics which can include the performance or cost of or loads on the object in question. The search for optimal values for object or system characteristics is carried out by means of optimal change to design, geometrical or other parameters of the object.

101.4.2 Search for optimal system management laws

It is often necessary to select or co-ordinate management parameters for the system while it is in operation in order to achieve a certain effect during the operation of the system or to reduce the impact of some factors on the system.

101.4.3 Identification of mathematical models

When the design process involves the use of any mathematical models of real-life objects, whether commercial or corporate, there is the problem of co-ordinating the experiment findings and model computation results. All models imply a set of unknown factors or constants. Searching for the optimal values thereof makes it possible to co-ordinate the experiment findings and model computation results.

101.5 Robust design optimization and robust optimal control

101.5.1 Introduction

Practical application of the numerical optimization results is difficult because any complex technical system is a stochastic system and the characteristics of this system have probabilistic nature. We would like to emphasize that, speaking about the stochastic properties of a technical system within the frame of optimization tasks, we imply that the important parameters of any system are stochastically spread. Normally it occurs during the production stage despite of the up-to-date level of modern technology. Random deviations of the system parameters lead to a random change in system efficiency.

An efficiency extreme value, obtained during the optimization problem while solving in traditional (deterministic) approach, is simply a maximum attainable value and can be considered as just conventional optimum from the point of view of its practical realization. Thus, one can consider two different types of optimization criteria. One of them is an ideal efficiency which can be achieved under the conditions of absolutely precise practical replication of the system parameters under consideration. Other optimization criteria are of probabilistic nature. For example: mathematical expectation of the efficiency; the total probability of assuring preset constraints; variance of the efficiency and so on. It is evident that the extreme of the one of these criteria doesn't guarantee the assurance of the high level of another one. Even more, these criteria may contradict to each other. Thus, in this case we have a multiobjective optimization problem.

101.5.2 IOSO robust design optimization concept

IOSO concept of robust design optimization and robust optimal control allows to determine the optimal practical solution that could be implemented with the high probability for the given technology level of the production plants. Many modern probabilistic approaches either employ the estimation of probabilistic efficiency criteria

only at the stage of the analysis of obtaining deterministic solution, or use significantly simplified assessments of probabilistic criteria during optimization process. The distinctive feature of our approach is that during robust design optimization we solve the optimization problem involving direct stochastic formulation, where the estimation of probabilistic criteria is accomplished at each iteration. This procedure reliably produces fully robust optimal solution. High efficiency of the robust design optimization is provided by the capabilities of IOSO algorithms to solve stochastic optimization problems with large level of noise.

101.6 References

- I.N. Egorov. Indirect Optimization Method on the Basis of Self-Organization. ICOTA'98, Perth, Australia, July 1...3, 1998 Conference Proceedings, vol.2, pp. 683–690
- Brian H. Dennis, Igor N. Egorov, Helmut Sobieczky, George S. Dulikravich, Shinobu Yoshimura. PARALLEL THERMOELASTICITY OPTIMIZATION OF 3-D SERPENTINE COOLING PASSAGES IN TURBINE BLADES. GT2003-38180, Proceedings of Turbo Expo 2003; Power for Land, Sea, and Air; June 16–19, 2003, Atlanta, Georgia, USA
- Brian H. Dennis, Igor N. Egorov, George S. Dulikravich, Shinobu Yoshimura. OPTIMIZATION OF A LARGE NUMBER OF COOLANT PASSAGES LOCATED CLOSE TO THE SURFACE OF A TURBINE BLADE. GT2003-38051, Proceedings of Turbo Expo 2003; 2003 ASME Turbo Expo; Atlanta, Georgia, June 16–19, 2003
- Egorov, I.N., Kretinin, G.V. and Leshchenko, I.A. "Robust Design Optimization Strategy of IOSO Technology". WCCM V, Fifth World Congress on Computational Mechanics, July 7–12, 2002, Vienna, Austria
- Egorov, I.N., Kretinin, G.V. and Leshchenko, I.A. "How to Execute Robust Design Optimization" (.pdf, 395Kb), 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, 04–06 September 2002, Atlanta, Georgia

101.7 External links

- IOSO technology website

Application examples

- Optimization of the Gas Turbine Engine Parts Using Methods of Numerical Simulation (pdf, 1500Kb)

- Sam146 Fan Stress Characteristics Optimization by IOSO (pdf, 120Kb)
- Parallel Thermoelasticity Optimization of 3-D Serpentine Cooling Passages in Turbine Blades (pdf, 260Kb)
- Optimization of Turbine Disk aimed to Mass and Stress Reduction (pdf, 680Kb)
- Calibration of Microprocessor Control Systems (pdf, 480Kb)
- Optimization of concentrations of alloying elements in steel (pdf, 370 Kb)
- Application of IOSO NM and ABAQUS at Civil Structures of NPP (pdf, 550Kb)

Chapter 102

IPOPT

IPOPT, short for "Interior Point OPTimizer, pronounced I-P-Opt", is a software library for large scale nonlinear optimization of continuous systems. It is written in Fortran and C and is released under the EPL (formerly CPL). IPOPT implements a primal-dual interior point method, and uses line searches based on Filter methods (Fletcher and Leyffer). IPOPT can be called from various modeling environments and C.

IPOPT is part of the COIN-OR project.

IPOPT is designed to exploit 1st and 2nd derivative (Hessians) information if provided (usually via automatic differentiation routines in modeling environments such as AMPL). If no Hessians are provided, IPOPT will approximate them using a quasi-Newton methods, specifically a BFGS update.

It was originally developed by Andreas Wächter, a former Ph.D. student in department of chemical engineering at Carnegie Mellon University, under the supervision of Lorenz T. Biegler.

Arvind Raghunathan later created an extension to IPOPT for Mathematical programming with equilibrium constraints (MPEC) . This version of IPOPT is generally known as IPOPT-C (with the 'C' standing for 'complementarity'). While in theory any mixed-integer program can be recast as an MPEC, it may or may not be solvable with IPOPT-C. Solution of MINLPs (Mixed-Integer Nonlinear Programs) using IPOPT is still being explored

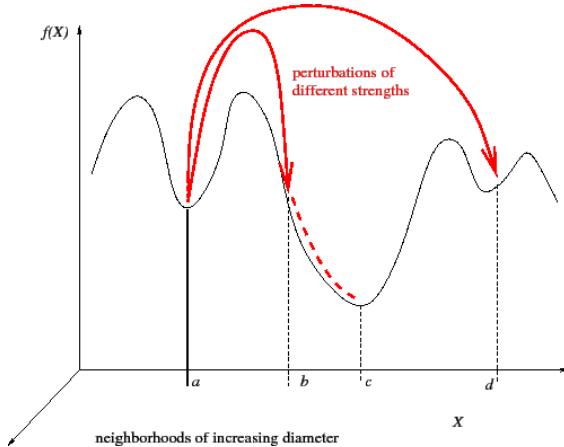
Carl Laird and Andreas Wächter are the developers of IPOPT 3.0, which is a re-implementation of IPOPT in C++.

102.1 External links

- IPOPT home page
- IPOPT is supported in AIMMS, AMPL, APMonitor, GAMS, MATLAB and OpenOpt

Chapter 103

Iterated local search



Iterated local search kicks a solution out from a local optimum

Iterated Local Search^{[1][2]} (ILS) is a term in applied mathematics and computer science defining a modification of local search or hill climbing methods for solving discrete optimization problems.

Local search methods can get stuck in a local minimum, where no improving neighbors are available.

A simple modification consists of *iterating* calls to the local search routine, each time starting from a different initial configuration. This is called *repeated local search*, and implies that the knowledge obtained during the previous local search phases is not used. Learning implies that the previous history, for example the memory about the previously found local minima, is mined to produce better and better starting points for local search.

The implicit assumption is that of a *clustered distribution of local minima*: when minimizing a function, determining good local minima is easier when starting from a **local minimum** with a low value than when starting from a random point. The only caveat is to avoid confinement in a given attraction basin, so that the *kick* to transform a local minimizer into the starting point for the next run has to be appropriately strong, but not too strong to avoid reverting to memory-less random restarts.

Iterated Local Search is based on building a sequence of locally optimal solutions by:

1. perturbing the current local minimum;

2. applying local search after starting from the modified solution.

The perturbation strength has to be sufficient to lead the trajectory to a different attraction basin leading to a different local optimum.

The method has been applied to several Combinatorial Optimization Problems including the Job-Shop Scheduling Problems,^{[3][4]} Flow-Shop Problems,^[5] Vehicle Routing Problems^[6] as well as many others.

103.1 References

- [1] Lourenço, H.R.; Martin O. and Stützle T. (2010). “Iterated Local Search: Framework and Applications”. *Handbook of Metaheuristics*, 2nd. Edition. Kluwer Academic Publishers, International Series in Operations Research & Management Science **146**: 363–397. doi:10.1007/978-1-4419-1665-5_12.
- [2] Lourenço, H.R.; Martin O. and Stützle T. (2003). “Iterated Local Search”. *Handbook of Metaheuristics*. Kluwer Academic Publishers, International Series in Operations Research & Management Science **57**: 321–353.
- [3] Lourenço, H.R.; Zwijnenburg M. (1996). “Combining the large-step optimization with tabu-search: application to the job-shop scheduling problem”. *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers: 219–236.
- [4] Lourenço, H.R. (1995). “Job-Shop Scheduling: computational study of local search and large-step optimization methods”. *European Journal of Operational Research* **83** (2): 347–364. doi:10.1016/0377-2217(95)00012-F.
- [5] Juan, A.A.; Lourenço, H.; Mateo, M.; Luo, R.; Castella, Q. (in press). “Using Iterated Local Search for solving the Flow-Shop Problem: parametrization, randomization and parallelization issues”. *International Transactions in Operational Research*. Check date values in: ldate=(help)
- [6] Penna, Puca H.V.; Satori Ochi L. Subramanian A. (2013). “An Iterated Local Search heuristic for the Heterogeneous Fleet Vehicle Routing Problem”. *Journal of Heuristics* **19** (2): 201–232. doi:10.1007/s10732-011-9186-y.

Chapter 104

Job shop scheduling

For other uses, see [Scheduling](#).

Job shop scheduling (or job-shop problem) is an optimization problem in [computer science](#) and [operations research](#) in which ideal jobs are assigned to resources at particular times. The most basic version is as follows:

We are given n jobs J_1, J_2, \dots, J_n of varying sizes, which need to be scheduled on m identical machines, while trying to minimize the makespan. The makespan is the total length of the schedule (that is, when all the jobs have finished processing). Nowadays, the problem is presented as an [online problem](#) (dynamic scheduling), that is, each job is presented, and the [online algorithm](#) needs to make a decision about that job before the next job is presented.

This problem is one of the best known online problems, and was the first problem for which [competitive analysis](#) was presented, by Graham in 1966.^[1] Best problem instances for basic model with makespan objective are due to Taillard.^[2]

104.1 Problem variations

Many variations of the problem exist, including the following:

- Machines can be related, independent, equal
- Machines can require a certain gap between jobs or no idle-time
- Machines can have sequence-dependent setups
- Objective function can be to minimize the makespan, the [L_p](#) norm, tardiness, maximum lateness etc. It can also be multi-objective optimization problem
- Jobs may have constraints, for example a job i needs to finish before job j can be started (see [workflow](#)). Also, the objective function can be multi-criteria.^[3]
- Jobs and machines have mutual constraints, for example, certain jobs can be scheduled on some machines only

- Set of jobs can relate to different set of machines
- Deterministic (fixed) processing times or probabilistic processing times
- There may also be some other side constraints

104.2 NP-hardness

If one already knows that the [travelling salesman problem](#) is NP-hard (as it is), then the job-shop problem with sequence-dependent setup is clearly also NP-hard, since the TSP is special case of the JSP with $m = 1$ (the salesman is the machine and the cities are the jobs).

104.3 Problem representation

The [disjunctive graph](#)^[4] is one of the popular models used for describing the job shop scheduling problem instances.^[5]

A mathematical statement of the problem can be made as follows:

Let $M = \{M_1, M_2, \dots, M_m\}$ and $J = \{J_1, J_2, \dots, J_n\}$ be two finite sets. On account of the industrial origins of the problem, the M_i are called **machines** and the J_j are called **jobs**.

Let \mathcal{X} denote the set of all sequential assignments of jobs to machines, such that every job is done by every machine exactly once; elements $x \in \mathcal{X}$ may be written as $n \times m$ matrices, in which column i lists the jobs that machine M_i will do, in order. For example, the matrix

$$x = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 1 \end{pmatrix}$$

means that machine M_1 will do the three jobs J_1, J_2, J_3 in the order J_1, J_2, J_3 , while machine M_2 will do the jobs in the order J_2, J_3, J_1 .

Suppose also that there is some **cost function** $C : \mathcal{X} \rightarrow [0, +\infty]$. The cost function may be interpreted as a “total

processing time”, and may have some expression in terms of times $C_{ij} : M \times J \rightarrow [0, +\infty]$, the cost/time for machine M_i to do job J_j .

The **job-shop problem** is to find an assignment of jobs $x \in \mathcal{X}$ such that $C(x)$ is a minimum, that is, there is no $y \in \mathcal{X}$ such that $C(x) > C(y)$.

104.4 The problem of infinite cost

One of the first problems that must be dealt with in the JSP is that many proposed solutions have infinite cost: i.e., there exists $x_\infty \in \mathcal{X}$ such that $C(x_\infty) = +\infty$. In fact, it is quite simple to concoct examples of such x_∞ by ensuring that two machines will deadlock, so that each waits for the output of the other's next step.

104.5 Major results

Graham had already provided the List scheduling algorithm in 1966, which is $(2 - 1/m)$ -competitive, where m is the number of machines.^[1] Also, it was proved that List scheduling is optimum online algorithm for 2 and 3 machines. The **Coffman–Graham algorithm** (1972) for uniform-length jobs is also optimum for two machines, and is $(2 - 2/m)$ -competitive.^{[6][7]} In 1992, Bar-tal, Fiat, Karloff and Vohra presented an algorithm that is 1.986 competitive.^[8] A 1.945-competitive algorithm was presented by Karger, Philips and Torn in 1994.^[9] In 1992, Albers provided a different algorithm that is 1.923-competitive.^[10] Currently, the best known result is an algorithm given by Fleischer and Wahl, which achieves a competitive ratio of 1.9201.^[11]

A lower bound of 1.852 was presented by Albers.^[12] Tail-lard instances has an important role in developing job shop scheduling with makespan objective.

In 1976 Garey provided a proof^[13] that this problem is NP-complete for $m > 2$, that is, no optimal solution can be computed in polynomial time for three or more machines (unless $P=NP$).

104.6 Offline makespan minimization

104.6.1 Atomic jobs

See also: Multiprocessor scheduling

The simplest form of the offline makespan minimisation problem deals with atomic jobs, that is, jobs that are not subdivided into multiple operations. It is equivalent to packing a number of items of various different sizes into

a fixed number of bins, such that the maximum bin size needed is as small as possible. (If instead the number of bins is to be minimised, and the bin size is fixed, the problem becomes a different problem, known as the bin packing problem.)

Hochbaum and Shmoys presented a polynomial-time approximation scheme in 1987 that finds an approximate solution to the offline makespan minimisation problem with atomic jobs to any desired degree of accuracy.^[14]

104.6.2 Jobs consisting of multiple operations

The basic form of the problem of scheduling jobs with multiple (M) operations, over M machines, such that all of the first operations must be done on the first machine, all of the second operations on the second, etc., and a single job cannot be performed in parallel, is known as the **open shop scheduling** problem. Various algorithms exist, including genetic algorithms.^[15]

Johnson's algorithm

See also: Johnson's rule

A heuristic algorithm by S. M. Johnson can be used to solve the case of a 2 machine N job problem when all jobs are to be processed in the same order.^[16] The steps of algorithm are as follows:

Job P_i has two operations, of duration P_{i1}, P_{i2} , to be done on Machine M_1, M_2 in that sequence.

- *Step 1.* List $A = \{ 1, 2, \dots, N \}$, List $L1 = \{ \}$, List $L2 = \{ \}$.
- *Step 2.* From all available operation durations, pick the minimum.

If the minimum belongs to P_{k1} ,

Remove K from list A ; Add K to end of List $L1$.

If minimum belongs to P_{k2} ,

Remove K from list A ; Add K to beginning of List $L2$.

- *Step 3.* Repeat Step 2 until List A is empty.
- *Step 4.* Join List $L1$, List $L2$. This is the optimum sequence.

Johnson's method only works optimally for two machines. However, since it is optimal, and easy to compute, some researchers have tried to adopt it for M machines, ($M > 2$.)

The idea is as follows: Imagine that each job requires m operations in sequence, on $M_1, M_2 \dots M_m$. We combine the first $m/2$ machines into an (imaginary) Machining center, MC1, and the remaining Machines into a Machining Center MC2. Then the total processing time for a Job P on MC1 = sum(operation times on first $m/2$ machines), and processing time for Job P on MC2 = sum(operation times on last $m/2$ machines).

By doing so, we have reduced the m -Machine problem into a Two Machining center scheduling problem. We can solve this using Johnson's method.

104.7 Example

Here is an example of a job shop scheduling problem formulated in **AMPL** as a mixed-integer programming problem with indicator constraints:

```
param N_JOBS; param N_MACHINES; set JOBS ordered = 1..N_JOBS; set MACHINES ordered = 1..N_MACHINES; param ProcessingTime{JOBS, MACHINES} > 0; param CumulativeTime{i in JOBS, j in MACHINES} = sum {jj in MACHINES: ord(jj) <= ord(j)} ProcessingTime[i,jj]; param TimeOffset{i1 in JOBS, i2 in JOBS: i1 <> i2} = max {j in MACHINES} (CumulativeTime[i1,j] - CumulativeTime[i2,j] + ProcessingTime[i2,j]); var end >= 0; var start{JOBS} >= 0; var precedes{i1 in JOBS, i2 in JOBS: ord(i1) < ord(i2)} binary; minimize makespan: end; subj to makespan_def{i in JOBS}: end >= start[i] + sum{j in MACHINES} ProcessingTime[i,j]; subj to no12_conflict{i1 in JOBS, i2 in JOBS: ord(i1) < ord(i2)}: precedes[i1,i2] ==> start[i2] >= start[i1] + TimeOffset[i1,i2]; subj to no21_conflict{i1 in JOBS, i2 in JOBS: ord(i1) < ord(i2)}: !precedes[i1,i2] ==> start[i1] >= start[i2] + TimeOffset[i2,i1]; data; param N_JOBS := 4; param N_MACHINES := 3; param ProcessingTime: 1 2 3 := 1 4 2 1 2 3 6 2 3 7 2 3 4 1 5 8;
```

104.8 See also

- Disjunctive graph
- Dynamic programming
- Flow shop scheduling
- Genetic algorithm scheduling
- List of NP-complete problems
- Open shop scheduling
- Optimal control
- Scheduling (production processes)

104.9 References

- [1] Graham, R. (1966). "Bounds for certain multiprocessor anomalies" (PDF). *Bell System Technical Journal* **45**: 1563–1581. doi:10.1002/j.1538-7305.1966.tb01709.x.
- [2] "Taillard Instances".
- [3] Malakooti, B (2013). *Operations and Production Systems with Multiple Objectives*. John Wiley & Sons. ISBN 978-1-118-58537-5.
- [4] B. Roy, B. Sussmann, Les problèmes d'ordonnancement avec contraintes disjonctives, SEMA, Note D.S., No. 9, Paris, 1964.
- [5] Jacek Błażewicz, Erwin Pesch, Małgorzata Sterna, The disjunctive graph machine representation of the job shop scheduling problem, European Journal of Operational Research, Volume 127, Issue 2, 1 December 2000, Pages 317-331, ISSN 0377-2217, 10.1016/S0377-2217(99)00486-5.
- [6] Coffman, E. G., Jr.; Graham, R. L. (1972), "Optimal scheduling for two-processor systems", *Acta Informatica* **1**: 200–213, doi:10.1007/bf00288685, MR 0334913.
- [7] Lam, Shui; Sethi, Ravi (1977), "Worst case analysis of two scheduling algorithms", *SIAM Journal on Computing* **6** (3): 518–536, doi:10.1137/0206037, MR 0496614.
- [8] Bartal, Y.; A. Fiat; H. Karloff; R. Vohra (1992). "New Algorithms for an Ancient Scheduling Problem". *Proc. 24th ACM Symp. Theory of Computing*. pp. 51–58. doi:10.1145/129712.129718.
- [9] Karger, D.; S. Phillips; E. Torng (1994). "A Better Algorithm for an Ancient Scheduling Problem". *Proc. Fifth ACM Symp. Discrete Algorithms*.
- [10] Albers, Susanne; Torben Hagerup (1992). "Improved parallel integer sorting without concurrent writing". *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*. Symposium on Discrete Algorithms archive. pp. 463–472.
- [11] Fleischer, Rudolf (2000). *Algorithms - ESA 2000*. Berlin / Heidelberg: Springer. pp. 202–210. ISBN 978-3-540-41004-1.
- [12] Albers, Susanne (1999). "Better bounds for online scheduling". *SIAM Journal on Computing* **29** (2): 459–473. doi:10.1137/S0097539797324874.
- [13] Garey, M.R. (1976). "The Complexity of Flowshop and Jobshop Scheduling". *Mathematics of Operations Research* **1** (2): 117–129. doi:10.1287/moor.1.2.117. JSTOR 3689278.
- [14] Hochbaum, Dorit; Shmoys, David (1987). "Using dual approximation algorithms for scheduling problems: theoretical and practical results" (PDF). *Journal of the ACM* **34** (1): 144–162. doi:10.1145/7531.7535.
- [15] Khuri, Sami; Miryala, Sowmya Rao (1999). "Genetic Algorithms for Solving Open Shop Scheduling Problems". *Proceedings of the 9th Portuguese Conference on Artificial Intelligence: Progress in Artificial Intelligence*. London: Springer Verlag. CiteSeerX: 10.1.1.29.4699.

- [16] S.M. Johnson, Optimal two- and three-stage production schedules with setup times included, Naval Res. Log. Quart. I(1954)61-68.

104.10 External links

- University of Vienna Directory of methodologies, systems and software for dynamic optimization.
- Taillard instances

Chapter 105

Kantorovich theorem

The **Kantorovich theorem** is a mathematical statement on the convergence of Newton's method. It was first stated by Leonid Kantorovich in 1940.

Newton's method constructs a sequence of points—that—with good luck—will converge to a solution x of an equation $f(x) = 0$ or a vector solution of a system of equation $F(x) = 0$. The Kantorovich theorem gives conditions on the initial point of this sequence. If those conditions are satisfied then a solution exists close to the initial point and the sequence converges to that point.

105.1 Assumptions

Let $X \subset \mathbb{R}^n$ be an open subset and $F : \mathbb{R}^n \supset X \rightarrow \mathbb{R}^n$ a differentiable function with a Jacobian $F'(x)$ that is locally Lipschitz continuous (for instance if it is twice differentiable). That is, it is assumed that for any open subset $U \subset X$ there exists a constant $L > 0$ such that for any $\mathbf{x}, \mathbf{y} \in U$

$$\|F'(\mathbf{x}) - F'(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\|$$

holds. The norm on the left is some operator norm that is compatible with the vector norm on the right. This inequality can be rewritten to only use the vector norm. Then for any vector $v \in \mathbb{R}^n$ the inequality

$$\|F'(\mathbf{x})(v) - F'(\mathbf{y})(v)\| \leq L \|\mathbf{x} - \mathbf{y}\| \|v\|$$

must hold.

Now choose any initial point $\mathbf{x}_0 \in X$. Assume that $F'(\mathbf{x}_0)$ is invertible and construct the Newton step $\mathbf{h}_0 = -F'(\mathbf{x}_0)^{-1}F(\mathbf{x}_0)$.

The next assumption is that not only the next point $\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{h}_0$ but the entire ball $B(\mathbf{x}_1, \|\mathbf{h}_0\|)$ is contained inside the set X . Let $M \leq L$ be the Lipschitz constant for the Jacobian over this ball.

As a last preparation, construct recursively, as long as it is possible, the sequences $(\mathbf{x}_k)_k$, $(\mathbf{h}_k)_k$, $(\alpha_k)_k$ according to

$$\mathbf{h}_k = -F'(\mathbf{x}_k)^{-1}F(\mathbf{x}_k)$$

$$\alpha_k = M \|F'(\mathbf{x}_k)^{-1}\| \|\mathbf{h}_k\|$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{h}_k.$$

105.2 Statement

Now if $\alpha_0 \leq \frac{1}{2}$ then

1. a solution \mathbf{x}^* of $F(\mathbf{x}^*) = 0$ exists inside the closed ball $\bar{B}(\mathbf{x}_1, \|\mathbf{h}_0\|)$ and
2. the Newton iteration starting in \mathbf{x}_0 converges to \mathbf{x}^* with at least linear order of convergence.

A statement that is more precise but slightly more difficult to prove uses the roots $t^* \leq t^{**}$ of the quadratic polynomial

$$p(t) = \left(\frac{1}{2}L \|F'(\mathbf{x}_0)^{-1}\|^{-1} \right) t^2 - t + \|\mathbf{h}_0\|$$

$$t^{*/**} = \frac{2\|\mathbf{h}_0\|}{1 \pm \sqrt{1 - 2\alpha}}$$

and their ratio

$$\theta = \frac{t^*}{t^{**}} = \frac{1 - \sqrt{1 - 2\alpha}}{1 + \sqrt{1 - 2\alpha}}.$$

Then

1. a solution \mathbf{x}^* exists inside the closed ball $\bar{B}(\mathbf{x}_1, \theta \|\mathbf{h}_0\|) \subset \bar{B}(\mathbf{x}_0, t^*)$
2. it is unique inside the bigger ball $B(\mathbf{x}_0, t^{**})$
3. and the convergence to the solution of F is dominated by the convergence of the Newton iteration of the quadratic polynomial $p(t)$ towards its smallest root t^* ,^[1] if $t_0 = 0$, $t_{k+1} = t_k - \frac{p(t_k)}{p'(t_k)}$, then

$$\|\mathbf{x}_{k+p} - \mathbf{x}_k\| \leq t_{k+p} - t_k.$$

4. The quadratic convergence is obtained from the error estimate^[2]

$$\|\mathbf{x}_{n+1} - \mathbf{x}^*\| \leq \theta^{2^n} \|\mathbf{x}_{n+1} - \mathbf{x}_n\| \leq \frac{\theta^{2^n}}{2^n} \|\mathbf{h}_0\|.$$

105.3 Notes

- [1] Ortega, J. M. (1968). “The Newton-Kantorovich Theorem”. *Amer. Math. Monthly* **75** (6): 658–660. JSTOR 2313800.
- [2] Gragg, W. B.; Tapia, R. A. (1974). “Optimal Error Bounds for the Newton-Kantorovich Theorem”. *SIAM Journal on Numerical Analysis* **11** (1): 10–13. doi:10.1137/0711002. JSTOR 2156425.

105.4 References

- John H. Hubbard and Barbara Burke Hubbard: *Vector Calculus, Linear Algebra, and Differential Forms: A Unified Approach*, Matrix Editions, ISBN 978-0-9715766-3-6 (preview of 3. edition and sample material including Kant.-thm.)

105.5 Literature

- Kantorowitsch, L. (1948): *Functional analysis and applied mathematics* (russ.). UMN3, 6 (28), 89–185.
- Kantorowitsch, L. W.; Akilow, G. P. (1964): *Functional analysis in normed spaces*.
- P. Deuflhard: *Newton Methods for Nonlinear Problems. Affine Invariance and Adaptive Algorithms.*, Springer, Berlin 2004, ISBN 3-540-21099-7 (Springer Series in Computational Mathematics, Vol. 35)

Chapter 106

Killer heuristic

In competitive two-player games, the **killer heuristic** is a technique for improving the efficiency of alpha-beta pruning, which in turn improves the efficiency of the minimax algorithm. This algorithm has an exponential search time to find the optimal next move, so general methods for speeding it up are very useful.

Alpha-beta pruning works best when the best moves are considered first. This is because the best moves are the ones most likely to produce a *cutoff*, a condition where the game playing program knows that the position it is considering could not possibly have resulted from best play by both sides and so need not be considered further. I.e. the game playing program will always make its best available move for each position. It only needs to consider the other player's possible responses to that best move, and can skip evaluation of responses to (worse) moves it will not make.

The killer heuristic attempts to produce a cutoff by assuming that a move that produced a cutoff in another branch of the **game tree** at the same depth is likely to produce a cutoff in the present position, that is to say that a move that was a very good move from a different (but possibly similar) position might also be a good move in the present position. By trying the *killer move* before other moves, a game playing program can often produce an early cutoff, saving itself the effort of considering or even generating all legal moves from a position.

In practical implementation, game playing programs frequently keep track of two killer moves for each depth of the game tree (greater than depth of 1) and see if either of these moves, if legal, produces a cutoff before the program generates and considers the rest of the possible moves. If a non-killer move produces a cutoff, it replaces one of the two killer moves at its depth. This idea can be generalized into a set of refutation tables.

A generalization of the killer heuristic is the *history heuristic*. The history heuristic can be implemented as a table that is indexed by some characteristic of the move, for example “from” and “to” squares or piece moving and the “to” square. When there is a cutoff, the appropriate entry in the table is incremented, such as by adding d^2 or 2^d where d is the current search depth. This information is used when ordering moves.

106.1 External links

- Informed Search in Complex Games by Mark Winands, https://dke.maastrichtuniversity.nl/m.winands/documents/informed_search.pdf

Chapter 107

LINCOA

LINCOA (LInearly Constrained Optimization Algorithm) is a numerical optimization algorithm by Michael J. D. Powell. It is also the name of Powell's Fortran 77 implementation of the algorithm.

LINCOA solves linearly constrained optimization problems without using derivatives of the objective function, which makes it a derivative-free algorithm. The algorithm solves the problem using a trust region method that forms quadratic models by interpolation. One new point is computed on each iteration, usually by solving a trust region subproblem subject to the linear constraints, or alternatively, by choosing a point to replace an interpolation point that may be too far away for reliability. In the second case, the new point may not satisfy the linear constraints.

The same as NEWUOA, LINCOA constructs the quadratic models by the least Frobenius norm updating [1] technique. A model function is determined by interpolating the objective function at m (an integer between $n + 2$ and $(n + 1)(n + 2)/2$) points; the remaining freedom, if any, is taken up by minimizing the Frobenius norm of the change to the model's Hessian (with respect to the last iteration).

LINCOA software was released on December 6, 2013.^[2] In the comment of the source code,^[3] it is said that LINCOA is not suitable for very large numbers of variables (which is typically true for algorithms not using derivatives), but “a few calculations with 1000 variables, however, have been run successfully overnight, and the performance of LINCOA is satisfactory usually for small numbers of variables.”^[3] It is also pointed out that the author's typical choices of m are $n + 6$ and $2n + 1$, the latter “being recommended for a start”, and “larger values tend to be highly inefficient when the number of variables is substantial, due to the amount of work and extra difficulty of adjusting more points.”^[3]

The trust region subproblem is solved by the truncated conjugate gradient method described in Powell's report,^[4] but Powell has not written a report on the other details of LINCOA (as of August 29, 2014).

107.1 See also

- TOLMIN
- COBYLA
- UOBYQA
- NEWUOA
- BOBYQA

107.2 References

- [1] Powell, M. J. D. (2004). “Least Frobenius norm updating of quadratic models that satisfy interpolation conditions”. *Mathematical Programming* (Springer) **100**: 183–215. doi:10.1007/s10107-003-0490-7.
- [2] “A repository of Powell's software”. Retrieved 2014-01-18.
- [3] “Source code of LINCOA software”. Retrieved 2014-01-18.
- [4] Powell, M. J. D. (August 2014). On fast trust region methods for quadratic models with linear constraints (Report). Department of Applied Mathematics and Theoretical Physics, Cambridge University. DAMTP 2014/NA02. Retrieved 2014-08-29.

107.3 External links

- Source code of LINCOA software

Chapter 108

Local convergence

In numerical analysis, an iterative method is called **locally convergent** if the successive approximations produced by the method are guaranteed to converge to a solution when the initial approximation is already close enough to the solution. Iterative methods for nonlinear equations and their systems, such as Newton's method are usually only locally convergent.

An iterative method that converges for an arbitrary initial approximation is called **globally convergent**. Iterative methods for systems of linear equations are usually globally convergent.

Chapter 109

Luus–Jaakola

In computational engineering, **Luus–Jaakola (LJ)** denotes a heuristic for global optimization of a real-valued function.^[1] In engineering use, LJ is not an algorithm that terminates with an optimal solution; nor is it an iterative method that generates a sequence of points that converges to an optimal solution (when one exists). However, when applied to a twice continuously differentiable function, the LJ heuristic is a proper iterative method, that generates a sequence that has a convergent subsequence; for this class of problems, Newton’s method is recommended and enjoys a quadratic rate of convergence, while no convergence rate analysis has been given for the LJ heuristic.^[1] In practice, the LJ heuristic has been recommended for functions that need be neither convex nor differentiable nor locally Lipschitz: The LJ heuristic does not use a gradient or subgradient when one be available, which allows its application to non-differentiable and non-convex problems.

Proposed by Luus and Jaakola,^[2] LJ generates a sequence of iterates. The next iterate is selected from a sample from a neighborhood of the current position using a uniform distribution. With each iteration, the neighborhood decreases, which forces a subsequence of iterates to converge to a cluster point.^[1]

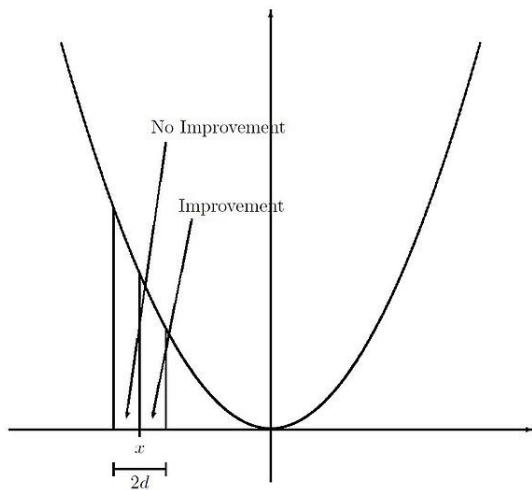
Luus has applied LJ in optimal control,^[3] transformer design,^[4] metallurgical processes,^[5] and chemical engineering.^[6]

109.1 Motivation

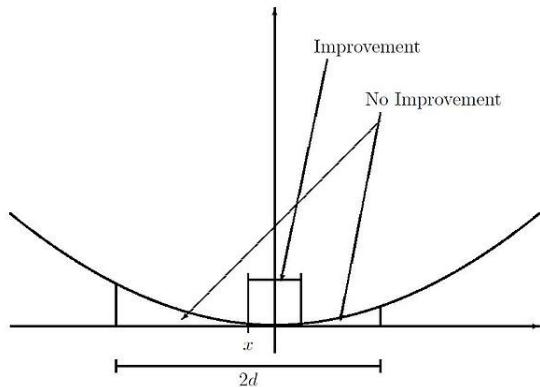
At each step, the LJ heuristic maintains a box from which it samples points randomly, using a uniform distribution on the box. For a unimodal function, the probability of reducing the objective function decreases as the box approach a minimum. The picture displays a one-dimensional example.

109.2 Heuristic

Let $f: \mathbb{R}^n \rightarrow \mathbb{R}$ be the fitness or cost function which must be minimized. Let $\mathbf{x} \in \mathbb{R}^n$ designate a position or candi-



When the current position x is far from the optimum the probability is 1/2 for finding an improvement through uniform random sampling.



As we approach the optimum the probability of finding further improvements through uniform sampling decreases towards zero if the sampling-range d is kept fixed.

date solution in the search-space. The LJ heuristic iterates the following steps:

- Initialize $\mathbf{x} \sim U(\mathbf{b}_{lo}, \mathbf{b}_{up})$ with a random uniform position in the search-space, where \mathbf{b}_{lo} and \mathbf{b}_{up} are the lower and upper boundaries, respectively.
- Set the initial sampling range to cover the entire

search-space (or a part of it): $\mathbf{d} = \mathbf{b}_{\text{up}} - \mathbf{b}_{\text{lo}}$

- Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat the following:
 - Pick a random vector $\mathbf{a} \sim U(-\mathbf{d}, \mathbf{d})$
 - Add this to the current position \mathbf{x} to create the new potential position $\mathbf{y} = \mathbf{x} + \mathbf{a}$
 - If ($f(\mathbf{y}) < f(\mathbf{x})$) then move to the new position by setting $\mathbf{x} = \mathbf{y}$, otherwise decrease the sampling-range: $\mathbf{d} = 0.95 \mathbf{d}$
- Now \mathbf{x} holds the best-found position.

109.3 Convergence

Nair proved a convergence analysis. For twice continuously differentiable functions, the LJ heuristic generates a sequence of iterates having a convergent subsequence.^[1] For this class of problems, Newton's method is the usual optimization method, and it has quadratic convergence (*regardless of the dimension* of the space, which can be a Banach space, according to Kantorovich's analysis).

The worst-case complexity of minimization on the class of unimodal functions grows exponentially in the dimension of the problem, according to the analysis of Yudin and Nemirovsky, however. The Yudin-Nemirovsky analysis implies that no method can be fast on high-dimensional problems that lack convexity:

“The catastrophic growth [in the number of iterations needed to reach an approximate solution of a given accuracy] as [the number of dimensions increases to infinity] shows that it is meaningless to pose the question of constructing universal methods of solving ... problems of any appreciable dimensionality 'generally'. It is interesting to note that the same [conclusion] holds for ... problems generated by uni-extremal [that is, unimodal] (but not convex) functions.”^[7]

When applied to twice continuously differentiable problems, the LJ heuristic's rate of convergence decreases as the number of dimensions increases.^[8]

109.4 See also

- **Random optimization** is a related family of optimization methods that sample from general distributions, for example the uniform distribution.
- **Random search** is a related family of optimization methods that sample from general distributions, for example, a uniform distribution on the unit sphere.

- Pattern search are used on noisy observations, especially in response surface methodology in chemical engineering. They do not require users to program gradients or hessians.

109.5 References

- [1] Nair, G. Gopalakrishnan (1979). “On the convergence of the LJ search method”. *Journal of Optimization Theory and Applications* **28** (3): 429–434. doi:10.1007/BF00933384. MR 543384.
- [2] Luus, R.; Jaakola, T.H.I. (1973). “Optimization by direct search and systematic reduction of the size of search region”. *American Institute of Chemical Engineers Journal (AIChE)* **19** (4): 760–766. doi:10.1002/aic.690190413.
- [3] Bojkov, R.; Hansel, B.; Luus, R. (1993). “Application of direct search optimization to optimal control problems”. *Hungarian Journal of Industrial Chemistry* **21**: 177–185.
- [4] Spaans, R.; Luus, R. (1992). “Importance of search-domain reduction in random optimization”. *Journal of Optimization Theory and Applications* **75**: 635–638. doi:10.1007/BF00940497. MR 1194836.
- [5] Papangelakis, V.G.; Luus, R. (1993). “Reactor optimization in the pressure oxidization process”. *Proc. Int. Symp. on Modelling, Simulation and Control of Metallurgical Processes*. pp. 159–171.
- [6] Lee, Y.P.; Rangaiah, G.P.; Luus, R. (1999). “Phase and chemical equilibrium calculations by direct search optimization”. *Computers & Chemical Engineering* **23** (9): 1183–1191.
- [7] Nemirovsky & Yudin (1983, p. 7)
Page 7 summarizes the later discussion of Nemirovsky & Yudin (1983, pp. 36–39): Nemirovsky, A. S.; Yudin, D. B. (1983). *Problem complexity and method efficiency in optimization*. Wiley-Interscience Series in Discrete Mathematics (Translated by E. R. Dawson from the (1979) Russian (Moscow: Nauka) ed.). New York: John Wiley & Sons, Inc. pp. xv+388. ISBN 0-471-10345-4. MR 702836.
- [8] Nair (1979, p. 433)

Nemirovsky, A. S.; Yudin, D. B. (1983). *Problem complexity and method efficiency in optimization*. Wiley-Interscience Series in Discrete Mathematics (Translated by E. R. Dawson from the (1979) Russian (Moscow: Nauka) ed.). New York: John Wiley & Sons, Inc. pp. xv+388. ISBN 0-471-10345-4. MR 702836.

Chapter 110

Matrix chain multiplication

Matrix chain multiplication (or Matrix Chain Ordering Problem, MCOP) is an optimization problem that can be solved using dynamic programming. Given a sequence of matrices, the goal is to find the most efficient way to multiply these matrices. The problem is not actually to *perform* the multiplications, but merely to decide the sequence of the matrix multiplications involved.

We have many options because matrix multiplication is associative. In other words, no matter how we parenthesize the product, the result obtained will remain the same. For example, if we had four matrices A , B , C , and D , we would have:

$$((AB)C)D = ((A(BC))D) = (AB)(CD) = A((BC)D) = A(B(CD)).$$

However, the order in which we parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the *efficiency*. For example, suppose A is a 10×30 matrix, B is a 30×5 matrix, and C is a 5×60 matrix. Then,

$$(AB)C = (10 \times 30 \times 5) + (10 \times 5 \times 60) = 1500 + 3000 = 4500 \text{ operations}$$

$$A(BC) = (30 \times 5 \times 60) + (10 \times 30 \times 60) = 9000 + 18000 = 27000 \text{ operations.}$$

Clearly the first method is more efficient. With this information, the problem statement can be refined, how do we determine the optimal parenthesization of a product of n matrices? We could go through each possible parenthesization (**brute force**), requiring a *run-time* that is exponential in the number of matrices, which is very slow and impractical for large n . A quicker solution to this problem can be achieved by breaking up the problem into a set of related subproblems. By solving subproblems one time and reusing these solutions, we can drastically reduce the run-time required. This concept is known as dynamic programming.

110.1 A Dynamic Programming Algorithm

To begin, let us assume that all we really want to know is the minimum cost, or minimum number of arithmetic operations, needed to multiply out the matrices. If we are only multiplying two matrices, there is only one way to multiply them, so the minimum cost is the cost of doing this. In general, we can find the minimum cost using the following recursive algorithm:

- Take the sequence of matrices and separate it into two subsequences.
- Find the minimum cost of multiplying out each subsequence.
- Add these costs together, and add in the cost of multiplying the two result matrices.
- Do this for each possible position at which the sequence of matrices can be split, and take the minimum over all of them.

For example, if we have four matrices $ABCD$, we compute the cost required to find each of $(A)(BCD)$, $(AB)(CD)$, and $(ABC)(D)$, making recursive calls to find the minimum cost to compute ABC , AB , CD , and BCD . We then choose the best one. Better still, this yields not only the minimum cost, but also demonstrates the best way of doing the multiplication: group it the way that yields the lowest total cost, and do the same for each factor.

Unfortunately, if we implement this algorithm we discover that it is just as slow as the naive way of trying all permutations! What went wrong? The answer is that we're doing a lot of redundant work. For example, above we made a recursive call to find the best cost for computing both ABC and AB . But finding the best cost for computing ABC also requires finding the best cost for AB . As the recursion grows deeper, more and more of this type of unnecessary repetition occurs.

One simple solution is called memoization: each time we compute the minimum cost needed to multiply out a spe-

cific subsequence, we save it. If we are ever asked to compute it again, we simply give the saved answer, and do not recompute it. Since there are about $n^2/2$ different subsequences, where n is the number of matrices, the space required to do this is reasonable. It can be shown that this simple trick brings the runtime down to $O(n^3)$ from $O(2^n)$, which is more than efficient enough for real applications. This is **top-down** dynamic programming.

From [1] Pseudocode:

```
// Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
MatrixChainOrder(int p[]) { // length[p] = n + 1 n =
p.length - 1; // m[i,j] = Minimum number of scalar
multiplications (i.e., cost) // needed to compute the
matrix A[i]A[i+1]...A[j] = A[i..j] // cost is zero when
multiplying one matrix for (i = 1; i <= n; i++) m[i,i] =
0; for (L=2; L<=n; L++) { // L is chain length for (i=1;
i<=n-L+1; i++) { j = i+L-1; m[i,j] = MAXINT; for
(k=i; k<=j-1; k++) { // q = cost/scalar multiplications q
= m[i,k] + m[k+1,j] + p[i-1]*p[k]*p[j]; if (q < m[i,j]) {
m[i,j] = q; s[i,j]=k; // s[i,j] = Second auxiliary table that
stores k // k = Index that achieved optimal cost } } } }
```

- Note : The first index for p is 0 and the first index for m and s is 1

Another solution is to anticipate which costs we will need and precompute them. It works like this:

- For each k from 2 to n , the number of matrices:
 - Compute the minimum costs of each subsequence of length k , using the costs already computed.

The code in java using zero based array indexes along with a convenience method for printing the solved order of operations:

```
public class MatrixOrderOptimization { protected
int[][]m; protected int[][]s; public void matrixChain-
Order(int[] p) { int n = p.length - 1; m = new int[n][n];
s = new int[n][n]; for (int ii = 1; ii < n; ii++) { for
(int i = 0; i < n - ii; i++) { int j = i + ii; m[i][j] =
Integer.MAX_VALUE; for (int k = i; k < j; k++) { int q =
m[i][k] + m[k+1][j] + p[i]*p[k+1]*p[j+1]; if (q < m[i][j]) { m[i][j] = q; s[i][j] = k; } } } }
public void printOptimalParenthesizations() { boolean[]
inAResult = new boolean[s.length]; printOptimal-
Parenthesizations(s, 0, s.length - 1, inAResult); } void
printOptimalParenthesizations(int[][]s, int i, int j, /* for
pretty printing: */ boolean[] inAResult) { if (i != j) {
printOptimalParenthesizations(s, i, s[i][j], inAResult);
printOptimalParenthesizations(s, s[i][j] + 1, j, inARes-
ult); String istr = inAResult[i] ? "_result" : " ";
String jstr = inAResult[j] ? "_result" : " ";
System.out.println("A_" + i + istr + "* A_" + j + jstr);
inAResult[i] = true; inAResult[j] = true; } } }
```

At the end of this program, we have the minimum cost for the full sequence. Although, this algorithm requires $O(n^3)$ time, this approach has practical advantages that it requires no recursion, no testing if a value has already been computed, and we can save space by throwing away some of the subresults that are no longer required. This is bottom-up dynamic programming: a second way by which this problem can be solved.

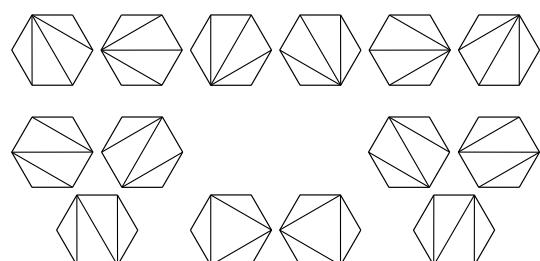
110.2 More Efficient Algorithms

There are algorithms that are more efficient than the $O(n^3)$ dynamic programming algorithm, though they are more complex.

110.2.1 Hu & Shing (1981)

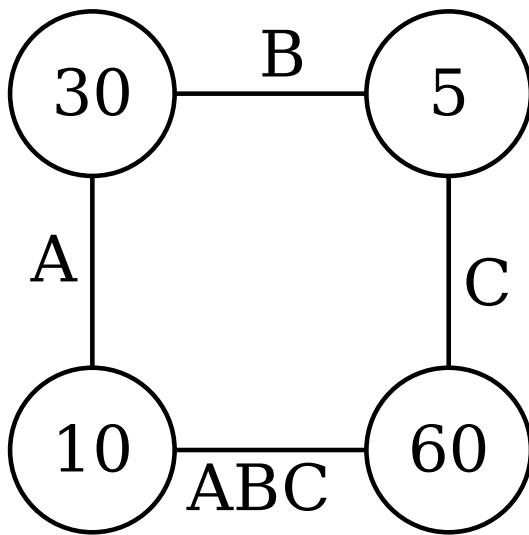
An algorithm published in 1981 by Hu and Shing achieves $O(n \log n)$ complexity.^{[2][3][4]} They showed how the matrix chain multiplication problem can be transformed (or reduced) into the problem of **triangulation** of a regular polygon. The polygon is oriented such that there is a horizontal bottom side, called the base, which represents the final result. The other n sides of the polygon, in the clockwise direction, represent the matrices. The vertices on each end of a side are the dimensions of the matrix represented by that side. With n matrices in the multiplication chain there are $n-1$ **binary operations** and C_{n-1} ways of placing parentheses, where C_{n-1} is the $(n-1)$ -th **Catalan number**. The algorithm exploits that there are also C_{n-1} possible triangulations of a polygon with $n+1$ sides.

This image illustrates possible triangulations of a regular **hexagon**. These correspond to the different ways that parentheses can be placed to order the multiplications for a product of 5 matrices.



For the example below, there are four sides: A, B, C and the final result ABC. A is a 10×30 matrix, B is a 30×5 matrix, C is a 5×60 matrix, and the final result is a 10×60 matrix. The regular polygon for this example is a 4-gon, i.e. a square:

The matrix product AB is a 10×5 matrix and BC is a 30×60 matrix. The two possible triangulations in this example are:



- Polygon representation of $(AB)C$
- Polygon representation of $A(BC)$

The cost of a single triangle in terms of the number of multiplications needed is the product of its vertices. The total cost of a particular triangulation of the polygon is the sum of the costs of all its triangles:

$$(AB)C: (10 \times 30 \times 5) + (10 \times 5 \times 60) = 1500 + 3000 = 4500 \text{ multiplications}$$

$$A(BC): (30 \times 5 \times 60) + (10 \times 30 \times 60) = 9000 + 18000 = 27000 \text{ multiplications}$$

Hu & Shing developed an algorithm that finds an optimum solution for the minimum cost partition problem in $O(n \log n)$ time.

110.3 Generalizations

The matrix chain multiplication problem generalizes to solving a more abstract problem: given a linear sequence of objects, an associative binary operation on those objects, and a way to compute the cost of performing that operation on any two given objects (as well as all partial results), compute the minimum cost way to group the objects to apply the operation over the sequence.^[5] One somewhat contrived special case of this is **string concatenation** of a list of strings. In C, for example, the cost of concatenating two strings of length m and n using `strcat` is $O(m + n)$, since we need $O(m)$ time to find the end of the first string and $O(n)$ time to copy the second string onto the end of it. Using this cost function, we can write a dynamic programming algorithm to find the fastest way to concatenate a sequence of strings. However, this optimization is rather useless because we can straightforwardly concatenate the strings in time proportional to the

sum of their lengths. A similar problem exists for singly linked lists.

Another generalization is to solve the problem when parallel processors are available. In this case, instead of adding the costs of computing each factor of a matrix product, we take the maximum because we can do them simultaneously. This can drastically affect both the minimum cost and the final optimal grouping; more “balanced” groupings that keep all the processors busy are favored. There are even more sophisticated approaches.^[6]

110.4 References

- [1] Cormen, Thomas H; Leiserson, Charles E; Rivest, Ronald L; Stein, Clifford (2001). “15.2: Matrix-chain multiplication”. *Introduction to Algorithms*. Second Edition. MIT Press and McGraw-Hill. pp. 331–338. ISBN 0-262-03293-7.
- [2] Hu, TC; Shing, MT (1981). *Computation of Matrix Chain Products, Part I, Part II* (PDF) (Technical report). Stanford University, Department of Computer Science. STAN-CS-TR-81-875.
- [3] Hu, TC; Shing, MT (1982). “Computation of Matrix Chain Products, Part I” (PDF). *SIAM Journal on Computing* (Society for Industrial and Applied Mathematics) **11** (2): 362–373. doi:10.1137/0211028. ISSN 0097-5397.
- [4] Hu, TC; Shing, MT (1984). “Computation of Matrix Chain Products, Part II” (PDF). *SIAM Journal on Computing* (Society for Industrial and Applied Mathematics) **13** (2): 228–251. doi:10.1137/0213017. ISSN 0097-5397.
- [5] G. Baumgartner, D. Bernholdt, D. Cociorva, R. Harrison, M. Nooijen, J. Ramanujam and P. Sadayappan. A Performance Optimization Framework for Compilation of Tensor Contraction Expressions into Parallel Programs. 7th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS '02). Fort Lauderdale, Florida. 2002 available at <http://citeseer.ist.psu.edu/610463.html> and at <http://www.csc.lsu.edu/~{}gb/TCE/Publications/OptFramework-HIPS02.pdf>
- [6] Heejo Lee, Jong Kim, Sungje Hong, and Sunggu Lee. Processor Allocation and Task Scheduling of Matrix Chain Products on Parallel Systems. *IEEE Trans. on Parallel and Distributed Systems*, Vol. 14, No. 4, pp. 394–407, Apr. 2003

Chapter 111

Maximum subarray problem

In computer science, the **maximum subarray problem** is the task of finding the contiguous subarray within a one-dimensional **array** of numbers (containing at least one positive number) which has the largest sum. For example, for the sequence of values $-2, 1, -3, 4, -1, 2, 1, -5, 4$; the contiguous subarray with the largest sum is $4, -1, 2, 1$, with sum 6.

The problem was first posed by Ulf Grenander of Brown University in 1977, as a simplified model for maximum likelihood estimation of patterns in digitized images. A linear time algorithm was found soon afterwards by Jay Kadane of Carnegie-Mellon University (Bentley 1984).

111.1 Kadane's algorithm

Kadane's algorithm consists of a scan through the array values, computing at each position the maximum (positive sum) subarray ending at that position. This subarray is either empty (in which case its sum is zero) or consists of one more element than the maximum subarray ending at the previous position. Thus, the problem can be solved with the following code, expressed here in Python:

```
def max_subarray(A): max_endng_here = max_so_far = 0 for x in A: max_endng_here = max(0, max_endng_here + x) max_so_far = max(max_so_far, max_endng_here) return max_so_far
```

A variation of the problem that does not allow zero-length subarrays to be returned, in the case that the entire array consists of negative numbers, can be solved with the following code:

```
def max_subarray(A): max_endng_here = max_so_far = A[0] for x in A[1:]: max_endng_here = max(x, max_endng_here + x) max_so_far = max(max_so_far, max_endng_here) return max_so_far
```

The algorithm can also be easily modified to keep track of the starting and ending indices of the maximum subarray.

Because of the way this algorithm uses optimal substructures (the maximum subarray ending at each position is calculated in a simple way from a related but smaller and

overlapping subproblem: the maximum subarray ending at the previous position) this algorithm can be viewed as a simple example of **dynamic programming**.

The runtime complexity of Kadane's algorithm is $O(n)$.

111.2 Generalizations

Similar problems may be posed for higher-dimensional arrays, but their solution is more complicated; see, e.g., Takaoka (2002). Brodal & Jørgensen (2007) showed how to find the k largest subarray sums in a one-dimensional array, in the optimal time bound $O(n + k)$.

111.3 See also

- Subset sum problem

111.4 References

- Bentley, Jon (1984), “Programming pearls: algorithm design techniques”, *Communications of the ACM* **27** (9): 865–873, doi:10.1145/358234.381162.
- Brodal, Gerth Stølting; Jørgensen, Allan Grønlund (2007), “A linear time algorithm for the k maximal sums problem”, *Mathematical Foundations of Computer Science 2007*, Lecture Notes in Computer Science **4708**, Springer-Verlag, pp. 442–453, doi:10.1007/978-3-540-74456-6_40.
- Takaoka, T. (2002), “Efficient algorithms for the maximum subarray problem by distance matrix multiplication”, *Electronic Notes in Theoretical Computer Science* **61**.

111.5 External links

- www.algorithmist.com

- alexeigor.wikidot.com
- Algorithm Design Techniques

Chapter 112

MCS algorithm

Multilevel Coordinate Search (MCS) is an algorithm for bound constrained global optimization using function values only.

To do so, the n-dimensional search space is represented by a set of non-intersecting hypercubes (boxes). The boxes are then iteratively split along an axis plane according to the value of the function at a representative point of the box and the box's size. These two splitting criteria combine to form a global search by splitting large boxes and a local search by splitting areas for which the function value is good.

Additionally a local search combining a quadratic interpolant of the function and line searches can be used to augment performance of the algorithm.

112.1 External links

- Homepage of the algorithm
- Performance of the algorithm relative to others

Chapter 113

Mehrotra predictor–corrector method

Mehrotra’s predictor–corrector method in optimization is an implementation of interior point methods. It was proposed in 1989 by Sanjay Mehrotra.^[1]

The method is based on the fact that at each iteration of an interior point algorithm it is necessary to compute the Cholesky decomposition (factorization) of a large matrix to find the search direction. The factorization step is the most computationally expensive step in the algorithm. Therefore it makes sense to use the same decomposition more than once before recomputing it.

At each iteration of the algorithm, Mehrotra’s predictor–corrector method uses the same Cholesky decomposition to find two different directions: a predictor and a corrector.

The idea is to first compute an optimizing search direction based on a first order term (predictor). The step size that can be taken in this direction is used to evaluate how much centrality correction is needed. Then, a corrector term is computed: this contains both a centrality term and a second order term.

The complete search direction is the sum of the predictor direction and the corrector direction.

Although there is no theoretical complexity bound on it yet, Mehrotra’s predictor–corrector method is widely used in practice.^[2] Its corrector step uses the same Cholesky decomposition found during the predictor step in an effective way, and thus it is only marginally more expensive than a standard interior point algorithm. However, the additional overhead per iteration is usually paid off by a reduction in the number of iterations needed to reach an optimal solution. It also appears to converge very fast when close to the optimum.

rent software; his work appeared in 1992.”

Potra, Florian A.; Stephen J. Wright (2000). “Interior-point methods”. *Journal of Computational and Applied Mathematics* **124** (1–2): 281–302. doi:10.1016/S0377-0427(00)00433-7.

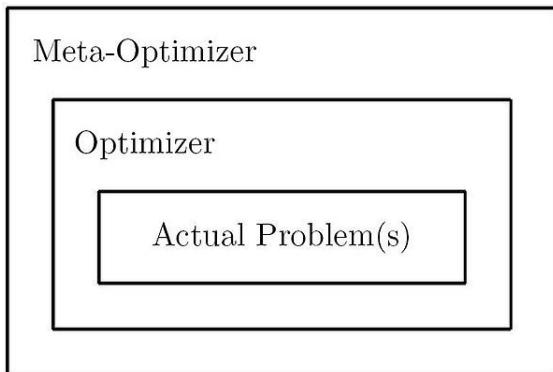
113.1 References

[1] Mehrotra, S. (1992). “On the implementation of a primal–dual interior point method”. *SIAM Journal on Optimization* **2** (4): 575–601. doi:10.1137/0802028.

[2] “In 1989, Mehrotra described a practical algorithm for linear programming that remains the basis of most cur-

Chapter 114

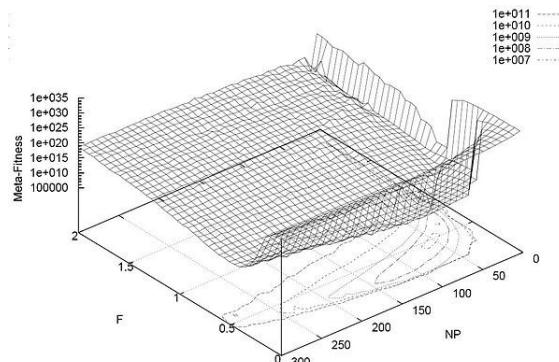
Meta-optimization



Meta-optimization concept.

In numerical optimization, **meta-optimization** is the use of one optimization method to tune another optimization method. Meta-optimization is reported to have been used as early as in the late 1970s by Mercer and Sampson^[1] for finding optimal parameter settings of a genetic algorithm. Meta-optimization is also known in the literature as meta-evolution, super-optimization, automated parameter calibration, hyper-heuristics, etc.

114.1 Motivation



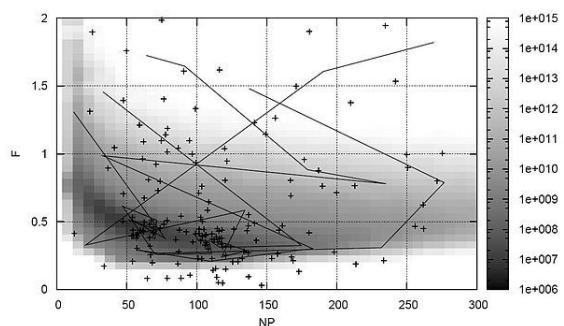
Performance landscape for differential evolution.

Optimization methods such as genetic algorithm and differential evolution have several parameters that govern

their behaviour and efficiency in optimizing a given problem and these parameters must be chosen by the practitioner to achieve satisfactory results. Selecting the behavioural parameters by hand is a laborious task that is susceptible to human misconceptions of what makes the optimizer perform well.

The behavioural parameters of an optimizer can be varied and the optimization performance plotted as a landscape. This is computationally feasible for optimizers with few behavioural parameters and optimization problems that are fast to compute, but when the number of behavioural parameters increases the time usage for computing such a performance landscape increases exponentially. This is the **curse of dimensionality** for the search-space consisting of an optimizer's behavioural parameters. An efficient method is therefore needed to search the space of behavioural parameters.

114.2 Methods



Meta-optimization of differential evolution.

A simple way of finding good behavioural parameters for an optimizer is to employ another overlaying optimizer, called the **meta-optimizer**. There are different ways of doing this depending on whether the behavioural parameters to be tuned are **real-valued** or **discrete-valued**, and depending on what performance measure is being used, etc.

Meta-optimizing the parameters of a genetic algorithm was done by Grefenstette^[2] and Keane,^[3] amongst oth-

ers, and experiments with meta-optimizing both the parameters and the genetic operators were reported by Bäck.^[4] Meta-optimization of the COMPLEX-RF algorithm was done by Krus and Andersson,^[5] and^[6] where performance index of optimization based on information theory was introduced and further developed. Meta-optimization of particle swarm optimization was done by Meissner et al.^[7] as well as by Pedersen and Chipperfield,^[8] who also meta-optimized differential evolution.^[9] Birattari et al.^{[10][11]} meta-optimized ant colony optimization. Statistical models have also been used to reveal more about the relationship between choices of behavioural parameters and optimization performance, see for example Francois and Lavergne,^[12] and Nannen and Eiben.^[13] A comparison of various meta-optimization techniques was done by Smit and Eiben.^[14]

114.3 See also

- Hyper-heuristics

114.4 References

- [1] Mercer, R.E.; Sampson, J.R. (1978). “Adaptive search using a reproductive metaplan”. *Kybernetes (The International Journal of Systems and Cybernetics)* **7** (3): 215–228. doi:10.1108/eb005486.
- [2] Grefenstette, J.J. (1986). “Optimization of control parameters for genetic algorithms”. *IEEE Transactions Systems, Man, and Cybernetics* **16** (1): 122–128. doi:10.1109/TSMC.1986.289288.
- [3] Keane, A.J. (1995). “Genetic algorithm optimization in multi-peak problems: studies in convergence and robustness”. *Artificial Intelligence in Engineering* **9** (2): 75–83. doi:10.1016/0954-1810(95)95751-Q.
- [4] Bäck, T. (1994). “Parallel optimization of evolutionary algorithms”. *Proceedings of the International Conference on Evolutionary Computation*. pp. 418–427.
- [5] Krus, P.K.; Andersson (Ölvander), J. (2003). “Optimizing optimization for design optimization”. *Proceedings of DETC'03 2003 ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference Chicago, Illinois, USA*.
- [6] Krus, P.K.; Ölvander(Andersson), J. (2013). “Performance index and meta-optimization of a direct search optimization method”. *Engineering Optimization* **45** (10): 1167–1185. doi:10.1080/0305215X.2012.725052.
- [7] Meissner, M.; Schmuker, M.; Schneider, G. (2006). “Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural network training”. *BMC Bioinformatics* **7** (1): 125. doi:10.1186/1471-2105-7-125. PMC 1464136. PMID 16529661.
- [8] Pedersen, M.E.H.; Chipperfield, A.J. (2010). “Simplifying particle swarm optimization”. *Applied Soft Computing* **10** (2): 618–628. doi:10.1016/j.asoc.2009.08.029.
- [9] Pedersen, M.E.H. (2010). *Tuning & Simplifying Heuristic Optimization* (PhD thesis). University of Southampton, School of Engineering Sciences, Computational Engineering and Design Group.
- [10] Birattari, M.; Stützle, T.; Paquete, L.; Varrentrapp, K. (2002). “A racing algorithm for configuring metaheuristics”. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. pp. 11–18.
- [11] Birattari, M. (2004). *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective* (PhD thesis). Université Libre de Bruxelles.
- [12] Francois, O.; Lavergne, C. (2001). “Design of evolutionary algorithms - a statistical perspective”. *IEEE Transactions on Evolutionary Computation* **5** (2): 129–148. doi:10.1109/4235.918434.
- [13] Nannen, V.; Eiben, A.E. (2006). “A method for parameter calibration and relevance estimation in evolutionary algorithms”. *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. pp. 183–190.
- [14] Smit, S.K.; Eiben, A.E. (2009). “Comparing parameter tuning methods for evolutionary algorithms”. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*. pp. 399–406.

Chapter 115

Minimax

This article is about the decision theory concept. For other uses, see [Minimax \(disambiguation\)](#).

Minimax (sometimes **MinMax** or **MM**^[1]) is a decision rule used in [decision theory](#), [game theory](#), [statistics](#) and [philosophy](#) for *minimizing* the possible [loss](#) for a worst case ([maximum loss](#)) scenario. Originally formulated for two-player [zero-sum game theory](#), covering both the cases where players take alternate moves and those where they make simultaneous moves, it has also been extended to more complex games and to general decision making in the presence of uncertainty.

This theorem was first published in 1928 by [John von Neumann](#),^[3] who is quoted as saying “As far as I can see, there could be no theory of games ... without that theorem ... I thought there was nothing worth publishing until the *Minimax Theorem* was proved”.^[4]

See Sion's minimax theorem and Parthasarathy's theorem for generalizations; see also example of a game without a value.

115.1 Game theory

In the theory of simultaneous games, a minimax strategy is a [mixed strategy](#) that is part of the solution to a zero-sum game. In zero-sum games, the minimax solution is the same as the [Nash equilibrium](#).

115.1.1 Minimax theorem

The minimax theorem states:^[2]

For every two-person, [zero-sum](#) game with finitely many strategies, there exists a value V and a mixed strategy for each player, such that

- (a) Given player 2's strategy, the best payoff possible for player 1 is V , and
- (b) Given player 1's strategy, the best payoff possible for player 2 is $-V$.

Equivalently, Player 1's strategy guarantees him a payoff of V regardless of Player 2's strategy, and similarly Player 2 can guarantee himself a payoff of $-V$. The name minimax arises because each player minimizes the maximum payoff possible for the other—since the game is zero-sum, he also minimizes his own maximum loss (i.e. maximize his minimum payoff).

115.1.2 Example

The following example of a zero-sum game, where **A** and **B** make simultaneous moves, illustrates *minimax* solutions. Suppose each player has three choices and consider the [payoff matrix](#) for **A** displayed at right. Assume the payoff matrix for **B** is the same matrix with the signs reversed (i.e. if the choices are A1 and B1 then **B** pays 3 to **A**). Then, the minimax choice for **A** is A2 since the worst possible result is then having to pay 1, while the simple minimax choice for **B** is B2 since the worst possible result is then no payment. However, this solution is not stable, since if **B** believes **A** will choose A2 then **B** will choose B1 to gain 1; then if **A** believes **B** will choose B1 then **A** will choose A1 to gain 3; and then **B** will choose B2; and eventually both players will realize the difficulty of making a choice. So a more stable strategy is needed.

Some choices are *dominated* by others and can be eliminated: **A** will not choose A3 since either A1 or A2 will produce a better result, no matter what **B** chooses; **B** will not choose B3 since some mixtures of B1 and B2 will produce a better result, no matter what **A** chooses.

A can avoid having to make an expected payment of more than $1/3$ by choosing A1 with probability $1/6$ and A2 with probability $5/6$: The expected payoff for **A** would be $3 \times (1/6) - 1 \times (5/6) = -1/3$ in case **B** chose B1 and $-2 \times (1/6) + 0 \times (5/6) = -1/3$ in case **B** chose B2. Similarly, **B** can ensure an expected gain of at least $1/3$, no matter what **A** chooses, by using a randomized strategy of choosing B1 with probability $1/3$ and B2 with probability $2/3$. These mixed minimax strategies are now stable and cannot be improved.

115.1.3 Maximin

Frequently, in game theory, **maximin** is distinct from minimax. Minimax is used in zero-sum games to denote minimizing the opponent's maximum payoff. In a zero-sum game, this is identical to minimizing one's own maximum loss, and to maximizing one's own minimum gain.

"Maximin" is a term commonly used for non-zero-sum games to describe the strategy which maximizes one's own minimum payoff. In non-zero-sum games, this is not generally the same as minimizing the opponent's maximum gain, nor the same as the **Nash equilibrium** strategy.

115.2 Combinatorial game theory

In combinatorial game theory, there is a minimax algorithm for game solutions.

A **simple** version of the minimax *algorithm*, stated below, deals with games such as **tic-tac-toe**, where each player can win, lose, or draw. If player A *can* win in one move, his best move is that winning move. If player B knows that one move will lead to the situation where player A *can* win in one move, while another move will lead to the situation where player A can, at best, draw, then player B's best move is the one leading to a draw. Late in the game, it's easy to see what the "best" move is. The Minimax algorithm helps find the best move, by working backwards from the end of the game. At each step it assumes that player A is trying to **maximize** the chances of A winning, while on the next turn player B is trying to **minimize** the chances of A winning (i.e., to maximize B's own chances of winning).

115.2.1 Minimax algorithm with alternate moves

A **minimax algorithm**^[5] is a recursive algorithm for choosing the next move in an n-player game, usually a two-player game. A value is associated with each position or state of the game. This value is computed by means of a **position evaluation function** and it indicates how good it would be for a player to reach that position. The player then makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves. If it is **A**'s turn to move, **A** gives a value to each of his legal moves.

A possible allocation method consists in assigning a certain win for **A** as +1 and for **B** as -1. This leads to **combinatorial game theory** as developed by John Horton Conway. An alternative is using a rule that if the result of a move is an immediate win for **A** it is assigned positive infinity and, if it is an immediate win for **B**, negative infinity. The value to **A** of any other move is the minimum

of the values resulting from each of **B**'s possible replies. For this reason, **A** is called the *maximizing player* and **B** is called the *minimizing player*, hence the name *minimax algorithm*. The above algorithm will assign a value of positive or negative infinity to any position since the value of every position will be the value of some final winning or losing position. Often this is generally only possible at the very end of complicated games such as **chess** or **go**, since it is not computationally feasible to look ahead as far as the completion of the game, except towards the end, and instead positions are given finite values as estimates of the degree of belief that they will lead to a win for one player or another.

This can be extended if we can supply a heuristic evaluation function which gives values to non-final game states without considering all possible following complete sequences. We can then limit the minimax algorithm to look only at a certain number of moves ahead. This number is called the "look-ahead", measured in "plies". For example, the chess computer **Deep Blue** (that beat Garry Kasparov) looked ahead at least 12 plies, then applied a heuristic evaluation function.^[6]

The algorithm can be thought of as exploring the **nodes** of a *game tree*. The **effective branching factor** of the tree is the average number of children of each node (i.e., the average number of legal moves in a position). The number of nodes to be explored usually **increases exponentially** with the number of plies (it is less than exponential if evaluating forced moves or repeated positions). The number of nodes to be explored for the analysis of a game is therefore approximately the branching factor raised to the power of the number of plies. It is therefore **impractical** to completely analyze games such as chess using the minimax algorithm.

The performance of the naïve minimax algorithm may be improved dramatically, without affecting the result, by the use of **alpha-beta pruning**. Other heuristic pruning methods can also be used, but not all of them are guaranteed to give the same result as the un-pruned search.

A naïve minimax algorithm may be trivially modified to additionally return an entire **Principal Variation** along with a minimax score.

115.2.2 Pseudocode

The pseudocode for the depth limited minimax algorithm is given below.

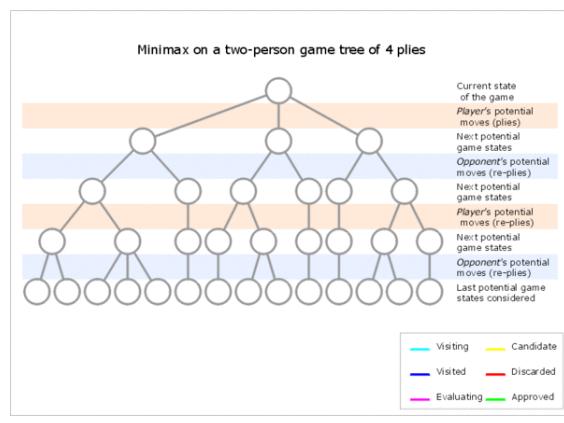
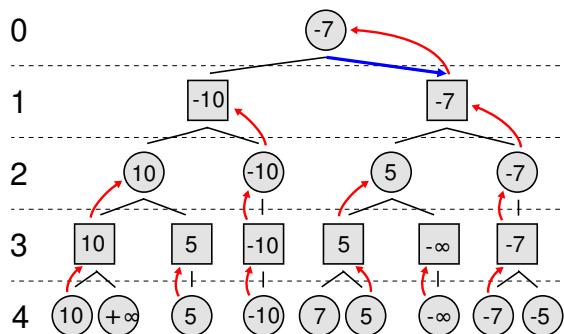
```
function minimax(node, depth, maximizingPlayer) if
depth = 0 or node is a terminal node return the heuristic
value of node if maximizingPlayer bestValue := -∞ for each child of node val := minimax(child, depth - 1, FALSE) bestValue := max(bestValue, val) return
bestValue else bestValue := +∞ for each child of node val := minimax(child, depth - 1, TRUE) bestValue := min(bestValue, val) return bestValue (* Initial call for
```

maximizing player *) minimax(origin, depth, TRUE)

The minimax function returns a heuristic value for **leaf nodes** (terminal nodes and nodes at the maximum search depth). Non leaf nodes inherit their value, *bestValue*, from a descendant leaf node. The heuristic value is a score measuring the favorability of the node for the maximizing player. Hence nodes resulting in a favorable outcome (such as a win) for the maximizing player have higher scores than nodes more favorable for the minimizing player. For non terminal leaf nodes at the maximum search depth, an evaluation function estimates a heuristic value for the node. The quality of this estimate and the search depth determine the quality and accuracy of the final minimax result.

Minimax treats the two players (the maximizing player and the minimizing player) separately in its code. Based on the observation that $\max(a, b) = -\min(-a, -b)$, minimax may often be simplified into the negamax algorithm.

115.2.3 Example



An animated pedagogical example that attempts to be human-friendly by substituting initial infinite (or arbitrarily large) values for emptiness and by avoiding using the negamax coding simplifications.

Suppose the game being played only has a maximum of two possible moves per player each turn. The algorithm generates the **tree** on the right, where the circles repre-

sent the moves of the player running the algorithm (*maximizing player*), and squares represent the moves of the opponent (*minimizing player*). Because of the limitation of computation resources, as explained above, the tree is limited to a *look-ahead* of 4 moves.

The algorithm evaluates each **leaf node** using a heuristic evaluation function, obtaining the values shown. The moves where the *maximizing player* wins are assigned with positive infinity, while the moves that lead to a win of the *minimizing player* are assigned with negative infinity. At level 3, the algorithm will choose, for each node, the **smallest** of the *child node* values, and assign it to that same node (e.g. the node on the left will choose the minimum between "10" and "+∞", therefore assigning the value "10" to itself). The next step, in level 2, consists of choosing for each node the **largest** of the *child node* values. Once again, the values are assigned to each *parent node*. The algorithm continues evaluating the maximum and minimum values of the child nodes alternately until it reaches the *root node*, where it chooses the move with the largest value (represented in the figure with a blue arrow). This is the move that the player should make in order to *minimize the maximum possible loss*.

115.3 Minimax for individual decisions

115.3.1 Minimax in the face of uncertainty

Minimax theory has been extended to decisions where there is no other player, but where the consequences of decisions depend on unknown facts. For example, deciding to prospect for minerals entails a cost which will be wasted if the minerals are not present, but will bring major rewards if they are. One approach is to treat this as a game against *nature* (see move by nature), and using a similar mindset as Murphy's law or resistentialism, take an approach which minimizes the maximum expected loss, using the same techniques as in the two-person zero-sum games.

In addition, **expectiminimax trees** have been developed, for two-player games in which chance (for example, dice) is a factor.

115.3.2 Minimax criterion in statistical decision theory

Main article: **Minimax estimator**

In classical statistical decision theory, we have an **estimator** δ that is used to estimate a **parameter** $\theta \in \Theta$. We also assume a **risk function** $R(\theta, \delta)$, usually specified as the integral of a **loss function**. In this framework, $\tilde{\delta}$ is called **minimax** if it satisfies

$$\sup_{\theta} R(\theta, \tilde{\delta}) = \inf_{\delta} \sup_{\theta} R(\theta, \delta).$$

An alternative criterion in the decision theoretic framework is the **Bayes estimator** in the presence of a prior distribution Π . An estimator is Bayes if it minimizes the *average* risk

$$\int_{\Theta} R(\theta, \delta) d\Pi(\theta).$$

115.3.3 Non-probabilistic decision theory

A key feature of minimax decision making is being non-probabilistic: in contrast to decisions using **expected value** or **expected utility**, it makes no assumptions about the probabilities of various outcomes, just scenario analysis of what the possible outcomes are. It is thus robust to changes in the assumptions, as these other decision techniques are not. Various extensions of this non-probabilistic approach exist, notably **minimax regret** and **Info-gap decision theory**.

Further, minimax only requires ordinal measurement (that outcomes be compared and ranked), not *interval* measurements (that outcomes include “how much better or worse”), and returns ordinal data, using only the modeled outcomes: the conclusion of a minimax analysis is: “this strategy is minimax, as the worst case is (outcome), which is less bad than any other strategy”. Compare to expected value analysis, whose conclusion is of the form: “this strategy yields $E(X)=n$.” Minimax thus can be used on ordinal data, and can be more transparent.

115.4 Maximin in philosophy

In philosophy, the term “maximin” is often used in the context of **John Rawls's A Theory of Justice**, where he refers to it (Rawls (1971, p. 152)) in the context of The Difference Principle. Rawls defined this principle as the rule which states that social and economic inequalities should be arranged so that “they are to be of the greatest benefit to the least-advantaged members of society”.^{[7][8]}

115.5 See also

- Alpha-beta pruning
- Claude Elwood Shannon
- Computer chess
- Expectiminimax
- Horizon effect

- Minimax Condorcet
- Monte Carlo tree search
- Minimax regret
- Negamax
- Negascout
- Sion's minimax theorem
- Transposition table
- Wald's maximin model

115.6 Notes

- [1] Provincial Healthcare Index 2013 (Bacchus Barua, Fraser Institute, January 2013 -see page 25-)
- [2] Osborne, Martin J., and Ariel Rubinstein. *A Course in Game Theory*. Cambridge, MA: MIT, 1994. Print.
- [3] Von Neumann, J: *Zur Theorie der Gesellschaftsspiele* Math. Annalen. **100** (1928) 295-320
- [4] John L Casti (1996). *Five golden rules: great theories of 20th-century mathematics – and why they matter*. New York: Wiley-Interscience. p. 19. ISBN 0-471-00261-5.
- [5] Russell, Stuart J.; Norvig, Peter (2003), *Artificial Intelligence: A Modern Approach* (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, pp. 163–171, ISBN 0-13-790395-2
- [6] Hsu, Feng-hsiung (1999), “IBM’s Deep Blue Chess Grandmaster Chips”, *IEEE Micro* (Los Alamitos, CA, USA: IEEE Computer Society) **19** (2): 70–81, doi:10.1109/40.755469, During the 1997 match, the software search extended the search to about 40 plies along the forcing lines, even though the nonextended search reached only about 12 plies.
- [7] Arrow, “Some Ordinalist-Utilitarian Notes on Rawls’s Theory of Justice, Journal of Philosophy 70, 9 (May 1973), pp. 245–263.
- [8] Harsanyi, “Can the Maximin Principle Serve as a Basis for Morality? a Critique of John Rawls’s Theory, American Political Science Review 69, 2 (June 1975), pp. 594–606.

115.7 External links

- Hazewinkel, Michiel, ed. (2001), “Minimax principle”, *Encyclopedia of Mathematics*, Springer, ISBN 978-1-55608-010-4
- A visualization applet
- “Maximin principle” from A Dictionary of Philosophical Terms and Names.

- Play a betting-and-bluffing game against a mixed minimax strategy
- The Dictionary of Algorithms and Data Structures entry for minimax.
- Minimax (with or without alpha-beta pruning) algorithm visualization — game tree solving (Java Applet), for balance or off-balance trees.
- CLISP minimax game. (in Spanish)
- Minimax Tutorial with a Numerical Solution Platform
- Java implementation used in a Checkers Game

Chapter 116

MM algorithm

The **MM algorithm** is an iterative optimization method which exploits the convexity of a function in order to find their maxima or minima. The **MM** stands for “Majorize-Minimization” or “Minorize-Maximization”, depending on whether you’re doing maximization or minimization. **MM** itself is not an algorithm, but a description of how to construct an optimization algorithm.

The **EM** algorithm can be treated as a special case of the **MM** algorithm.^[1] However, in the **EM** algorithm complex conditional expectation and extensive analytical skills are usually involved, while in the **MM** algorithm convexity and inequalities are our major focus, and it is relatively easier to understand and apply in most of the cases.

116.1 History

The original idea of the **MM algorithm** can be dated back at least to 1970 when Ortega and Rheinboldt were doing their studies related to line search methods.^[2] The same idea kept reappearing under different guises in different areas until 2000 when Hunter and Lange put forth “MM” as general frame work.^[3] Recently studies have shown that it can be used in a wide range of context, like mathematics, statistics, machine learning, engineering, etc.

116.2 How it works

MM algorithm works by finding a surrogate function that minorizes or majorizes the objective function. Optimizing the surrogate functions will drive the objective function upward or downward until a local optimum is reached.

Take the **minorize-maximization** version for example.

Let $f(\theta)$ be the objective concave function we want to maximize. At the m step of the algorithm, $m = 0, 1\dots$, the constructed function $g(\theta|\theta_m)$ will be called the minorized version of the objective function (the surrogate function) at θ_m if

$$g(\theta|\theta_m) \leq f(\theta) \text{ for all } \theta \quad g(\theta_m|\theta_m) = f(\theta_m)$$

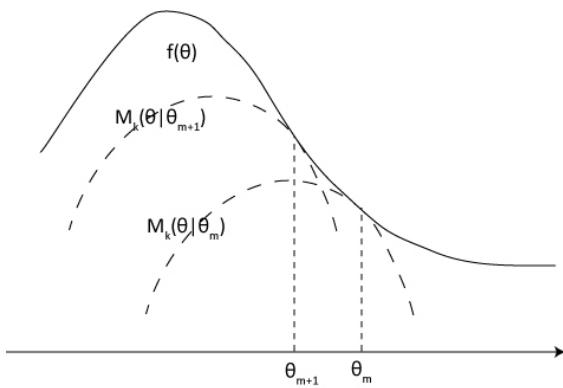
Then we maximize $g(\theta|\theta_m)$ instead of $f(\theta)$, and let

$$\theta_{m+1} = \arg \max_{\theta} g(\theta|\theta_m)$$

The above iterative method will guarantee that $f(\theta_m)$ will converge to a local optimum or a saddle point as m goes to infinity.^[4] By the construction we have

$$f(\theta_{m+1}) \geq g(\theta_{m+1}|\theta_m) \geq g(\theta_m|\theta_m) = f(\theta_m)$$

The marching of θ_m and the surrogate functions relative to the objective function is shown on the Figure



MM algorithm

We can just flip the image upside down, and that would be the methodology while we are doing **Majorize-Minimization**.

116.3 Ways to construct surrogate functions

Basically, we can use any inequalities to construct the desired majorized/minorized version of the objective function, but there are several typical choices

- Jensen’s inequality
- Convexity inequality
- Cauchy–Schwarz inequality
- Inequality of arithmetic and geometric means

116.4 References

- [1] Lange, Kenneth. “The MM Algorithm”.
- [2] Ortega, J.M.; Rheinboldt, W.C. (1970). “Iterative Solutions of Nonlinear Equations in Several Variables”. *New York: Academic*: 253–255.
- [3] Hunter, D.R.; Lange, K. (2000). “Quantile Regression via an MM Algorithm”. *Journal of Computational and Graphical Statistics* **9**: 60–77. doi:10.2307/1390613.
- [4] Wu, C. F. Jeff (Mar 1983). “On the Convergence Properties of the EM Algorithm”

Chapter 117

Multi-swarm optimization

Multi-swarm optimization is a variant of Particle swarm optimization (PSO) based on the use of multiple sub-swarms instead of one (standard) swarm. The general approach in multi-swarm optimization is that each sub-swarm focuses on a specific region while a specific diversification method decides where and when to launch the sub-swarms. The multi-swarm framework is especially fitted for the optimization on multi-modal problems, where multiple (local) optima exist.

117.1 Description

In multi-modal problems it is important to achieve an effective balance between exploration and exploitation. Multi-swarm systems provide a new approach to improve this balance. Instead of trying to achieve a compromise between exploration and exploitation which could weaken both mechanisms of the search process, multi-swarm systems separate them into distinct phases. Each phase is more focused on either exploitation (individual sub-swarms) or exploration (diversification method).

The coordination of the sub-swarms depends on the specific diversification method(s) implemented by the multi-swarm system. Wave of Swarm of Particles (WOSP),^[1] for example, bases its diversification mechanism on the “collision” of particles. When particles get too close they are expelled by a short range force into new waves/sub-swarms, avoiding thus a complete convergence. The Dynamic Multi-Swarm-Particle Swarm Optimizer (DMS-PSO)^[2] periodically regroups the particles of the sub-swarms (after they have converged) into new sub-swarms, the new swarms are started with particles from previous swarms. Locust swarms^[3] are based on a “devour and move on” strategy – after a sub-swarm “devours” a relatively small region of the search space (to find a local optimum) scouts are deployed to look for new promising regions to “move on”.

A distinctive feature of sub-swarms is that their initial positions and initial velocities are not randomly selected as in normal swarms. Instead, they maintain some information from the previous trajectories of the particles. In general, the development of multi-swarm systems leads

to design decisions which did not exist during the original development of particle swarm optimization, such as the number of particles to use in each sub-swarm, the optimal value for the constriction factor and the effects of non-random initial positions and initial velocities. These design decisions have been thoroughly studied and have well-established guidelines – e.g. the use of non-random initial positions and initial velocities leads to improved results in multi-swarm systems, which is not the case for single-swarms.^[4] Other design decisions, such as which diversification method to use or which specific search strategy will select the initial positions and initial velocities of a sub-swarm, have less established guidelines and constitute open questions in the field of multi-swarm systems.

Some of these design decisions can be addressed by relatively independent sub-components which allow different optimization techniques to be inserted. Multi-swarm systems thus provide a useful framework for the development of hybrid algorithms. For example, the UMDA-PSO^[5] multi-swarm system effectively combines components from Particle swarm optimization, Estimation of distribution algorithm, and Differential evolution into a multi-swarm hybrid.

117.2 Current Work

A reading group on Mendeley is available to all interested researchers.

117.3 See also

- Particle swarm optimization
- Swarm intelligence

117.4 References

[1] T. Hendtlass, “WoSP: A Multi-Optima Particle Swarm Algorithm,” in Proceedings IEEE Congress on Evolutionary Computation, 2005, pp. 727–734.

- [2] S. Z. Zhao, J. J. Liang, P. N. Suganthan, and M. F. Tasgetiren, "Dynamic Multi-Swarm Particle Swarm Optimizer with Local Search for Large Scale Global Optimization," in Proceedings IEEE Congress on Evolutionary Computation, 2008, pp. 3845–3852.
- [3] S. Chen, "Locust Swarms – A New Multi-Optima Search Technique", in Proceedings of the IEEE Congress on Evolutionary Computation, 2009, pp. 1745–1752.
- [4] S. Chen and J. Montgomery "Selection Strategies for Initial Positions and Initial Velocities in Multi-optima Particle Swarms", in Proceedings of the Genetic and Evolutionary Computation Conference, 2011 pp. 53–60.
- [5] Antonio Bolufé Röhler and S. Chen, "Multi-swarm hybrid for multi-modal optimization", in Proceedings of the IEEE Congress on Evolutionary Computation, 2012, pp. 1759-1766.

Chapter 118

Natural evolution strategy

Natural evolution strategies (NES) are a family of numerical optimization algorithms for black-box problems. Similar in spirit to evolution strategies, they iteratively update the (continuous) parameters of a *search distribution* by following the **natural gradient** towards higher expected fitness.

118.1 Method

The general procedure is as follows: the *parameterized* search distribution is used to produce a batch of search points, and the **fitness function** is evaluated at each such point. The distribution's parameters (which include *strategy parameters*) allow the algorithm to adaptively capture the (local) structure of the fitness function. For example, in the case of a **Gaussian distribution**, this comprises the mean and the **covariance matrix**. From the samples, NES estimates a search gradient on the parameters towards higher expected fitness. NES then performs a gradient ascent step along the **natural gradient**, a second order method which, unlike the plain gradient, renormalizes the update w.r.t. uncertainty. This step is crucial, since it prevents oscillations, premature convergence, and undesired effects stemming from a given parameterization. The entire process reiterates until a stopping criterion is met.

All members of the NES family operate based on the same principles. They differ in the type of probability distribution and the gradient approximation method used. Different search spaces require different search distributions; for example, in low dimensionality it can be highly beneficial to model the full covariance matrix. In high dimensions, on the other hand, a more scalable alternative is to limit the covariance to the **diagonal** only. In addition, highly multi-modal search spaces may benefit from more **heavy-tailed distributions** (such as **Cauchy**, as opposed to the Gaussian). A last distinction arises between distributions where we can analytically compute the natural gradient, and more general distributions where we need to estimate it from samples.

118.1.1 Search gradients

Let θ denote the parameters of the search distribution $\pi(x | \theta)$ and $f(x)$ the fitness function evaluated at x . NES then pursues the objective of maximizing the *expected fitness under the search distribution*

$$J(\theta) = \mathbb{E}_\theta[f(x)] = \int f(x) \pi(x | \theta) dx$$

through gradient ascent. The gradient can be rewritten as

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \int f(x) \pi(x | \theta) dx \\ &= \int f(x) \nabla_\theta \pi(x | \theta) dx \\ &= \int f(x) \nabla_\theta \pi(x | \theta) \frac{\pi(x | \theta)}{\pi(x | \theta)} dx \\ &= \int [f(x) \nabla_\theta \log \pi(x | \theta)] \pi(x | \theta) dx \\ &= \mathbb{E}_\theta [f(x) \nabla_\theta \log \pi(x | \theta)] \end{aligned}$$

that is, the **expected value** of $f(x)$ times the **log-derivatives** at x . In practice, it is possible to use the **Monte Carlo** approximation based on a finite number of λ samples

$$\nabla_\theta J(\theta) \approx \frac{1}{\lambda} \sum_{k=1}^{\lambda} f(x_k) \nabla_\theta \log \pi(x_k | \theta)$$

Finally, the parameters of the search distribution can be updated iteratively

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

118.1.2 Natural gradient ascent

Instead of using the plain stochastic gradient for updates, NES follows the **natural gradient**, which has been shown to possess numerous advantages over the plain (*vanilla*) gradient, e.g.:

- the gradient direction is independent of the parameterization of the search distribution
- the updates magnitudes are automatically adjusted based on uncertainty, in turn speeding convergence on plateaus and ridges.

The NES update is therefore

$$\theta \leftarrow \theta + \eta \mathbf{F}^{-1} \nabla_{\theta} J(\theta)$$

where \mathbf{F} is the Fisher information matrix. The Fisher matrix can sometimes be computed exactly, otherwise it is estimated from samples, reusing the log-derivatives $\nabla_{\theta} \log \pi(x|\theta)$.

118.1.3 Fitness shaping

NES utilizes rank-based fitness shaping in order to render the algorithm more robust, and *invariant* under monotonically increasing transformations of the fitness function. For this purpose, the fitness of the population is transformed into a set of utility values $u_1 \geq \dots \geq u_{\lambda}$. Let x_i denote the i^{th} best individual. Replacing fitness with utility, the gradient estimate becomes

$$\nabla_{\theta} J(\theta) = \sum_{k=1}^{\lambda} u_k \nabla_{\theta} \log \pi(x_k | \theta)$$

The choice of utility function is a free parameter of the algorithm.

118.1.4 Pseudocode

```

input:  $f$ ,  $\theta_{\text{init}}$  1 repeat 2 for  $k = 1 \dots \lambda$  do //  $\lambda$  is
the population size 3 draw sample  $x_k \sim \pi(\cdot | \theta)$  4 evaluate
fitness  $f(x_k)$  5 calculate log-derivatives  $\nabla_{\theta} \log \pi(x_k | \theta)$ 
6 end 7 assign the utilities  $u_k$  //based on rank 8 estimate
the gradient  $\nabla_{\theta} J \leftarrow \frac{1}{\lambda} \sum_{k=1}^{\lambda} u_k \cdot \nabla_{\theta} \log \pi(x_k | \theta)$  9 es-
timate  $\mathbf{F} \leftarrow \frac{1}{\lambda} \sum_{k=1}^{\lambda} \nabla_{\theta} \log \pi(x_k | \theta) \nabla_{\theta} \log \pi(x_k | \theta)^{\top}$  //
or compute it exactly 10 update parameters  $\theta \leftarrow \theta + \eta \cdot$ 
 $\mathbf{F}^{-1} \nabla_{\theta} J$  //  $\eta$  is the learning rate 11 until stopping cri-
terion is met

```

118.2 See also

- Evolutionary computation
- Covariance matrix adaptation evolution strategy (CMA-ES)

118.3 Bibliography

- D. Wierstra, T. Schaul, J. Peters and J. Schmidhuber (2008). **Natural Evolution Strategies**. IEEE Congress on Evolutionary Computation (CEC).
- Y. Sun, D. Wierstra, T. Schaul and J. Schmidhuber (2009). **Stochastic Search using the Natural Gradient**. International Conference on Machine Learning (ICML).
- T. Glasmachers, T. Schaul, Y. Sun, D. Wierstra and J. Schmidhuber (2010). **Exponential Natural Evolution Strategies**. Genetic and Evolutionary Computation Conference (GECCO).
- T. Schaul, T. Glasmachers and J. Schmidhuber (2011). **High Dimensions and Heavy Tails for Natural Evolution Strategies**. Genetic and Evolutionary Computation Conference (GECCO).
- T. Schaul (2012). **Natural Evolution Strategies Converge on Sphere Functions**. Genetic and Evolutionary Computation Conference (GECCO).

118.4 External links

- Collection of NES implementations in different languages

Chapter 119

Negamax

Negamax search is a variant form of minimax search that relies on the zero-sum property of a two-player game.

This algorithm relies on the fact that $\max(a, b) = -\min(-a, -b)$ to simplify the implementation of the minimax algorithm. More precisely, the value of a position to player A in such a game is the negation of the value to player B. Thus, the player on move looks for a move that maximizes the negation of the value of the position resulting from the move: this successor position must by definition have been valued by the opponent. The reasoning of the previous sentence works regardless of whether A or B is on move. This means that a single procedure can be used to value both positions. This is a coding simplification over minimax, which requires that A select the move with the maximum-valued successor while B selects the move with the minimum-valued successor.

It should not be confused with **negascout**, an algorithm to compute the minimax or negamax value quickly by clever use of **alpha-beta pruning** discovered in the 1980s. Note that alpha-beta pruning is itself a way to compute the minimax or negamax value of a position quickly by avoiding the search of certain uninteresting positions.

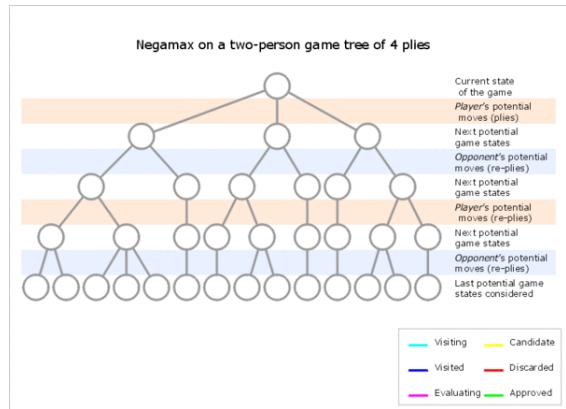
Most adversarial search engines are coded using some form of negamax search.

119.1 NegaMax Base Algorithm

NegaMax operates on the same game trees as those used with the minimax search algorithm. Each node and root node in the tree are game states (such as game board configuration) of a two player game. Transitions to child nodes represent moves available to a player who's about to play from a given node.

The negamax search objective is to find the node score value for the player who is playing at the root node. The pseudocode below shows the negamax base algorithm,^[1] with a configurable limit for the maximum search depth:

```
function negamax(node, depth, color) if depth = 0 or
node is a terminal node return color * the heuristic value
of node bestValue := -∞ foreach child of node val :=
-negamax(child, depth - 1, -color) bestValue := max(
```



An animated pedagogical example showing the plain negamax algorithm (that is, without alpha-beta pruning). The person performing the game tree search is considered to be the one that has to move first from the current state of the game (player in this case)

```
bestValue, val ) return bestValue Initial call for Player
A's root node rootNegamaxValue := negamax( rootNode,
depth, 1) rootMinimaxValue := rootNegamaxValue Initial
call for Player B's root node rootNegamaxValue := negamax(
rootNode, depth, -1) rootMinimaxValue := -rootNegamaxValue
```

The root node inherits its score from one of its immediate child nodes. The child node that ultimately sets the root node's best score also represents the best move to play. Although the pseudocode just retains the best score as *bestValue*, practical implementations will also retain the best move, along with *bestValue*.

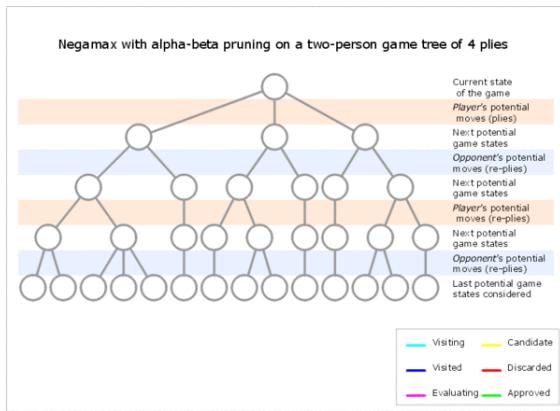
What can be confusing is how the heuristic value of the current node is calculated. In this implementation, this value is always calculated from the point of view of player A, whose color value is one. In other words, higher heuristic values always represent situations more favorable for player A. This is the same behavior as the normal minimax algorithm. The heuristic value is not necessarily the same as a node's return value, *bestValue*, due to value negation by negamax and the color parameter. The negamax node's return value is a heuristic score from the point of view of the node's current player.

Negamax scores match minimax scores for nodes where

player A is about to play, and where player A is the maximizing player in the minimax equivalent. Negamax always searches for the maximum value for all its nodes. Hence for player B nodes, the minimax score is a negation of its negamax score. Player B is the minimizing player in the minimax equivalent.

Variations in negamax implementations may omit the color parameter. In this case, the heuristic evaluation function must return values from the point of view of the node's current player.

119.2 NegaMax with Alpha Beta Pruning



An animated pedagogical example showing the negamax algorithm with alpha-beta pruning. The person performing the game tree search is considered to be the one that has to move first from the current state of the game (player in this case)

Algorithm optimizations for minimax are also equally applicable for Negamax. Alpha-beta pruning can decrease the number of nodes the negamax algorithm evaluates in a search tree in a manner similar with its use with the minimax algorithm.

The pseudocode for depth-limited negamax search with alpha-beta pruning follows:^[1]

```
function negamax(node, depth, α, β, color) if depth = 0 or node is a terminal node return color * the heuristic value of node bestValue := -∞ childNodes := GenerateMoves(node) childNodes := OrderMoves(childNodes)
foreach child in childNodes val := -negamax(child, depth - 1, -β, -α, -color) bestValue := max( bestValue, val ) α := max( α, val ) if α ≥ β break return bestValue Initial call for Player A's root node rootNegamaxValue := negamax( rootNode, depth, -∞, +∞, 1 )
```

Alpha (α) and beta (β) represent lower and upper bounds for child node values at a given tree depth. Negamax sets the arguments α and β for the root node to the lowest and highest values possible. Other search algorithms, such as negascout and MTD-f, may initialize α and β with alternate values to further improve tree search performance.

When negamax encounters a child node value outside an alpha/beta range, the negamax search cuts off (*break* statement in the pseudocode) thereby pruning portions of the game tree from exploration. Cut offs are implicit based on the node return value, *bestValue*. A node value found within the range of its initial α and β is the node's exact (or true) value. This value is identical to the result the negamax base algorithm would return, without cut offs and without any α and β bounds. If a node return value is out of range, then the value represents an upper (if value $\leq \alpha$) or lower (if value $\geq \beta$) bound for the node's exact value. Alpha-beta pruning eventually discards any value bound results. Such values do not contribute nor affect the negamax value at its root node.

This pseudocode shows the fail-soft variation of alpha-beta pruning. Fail-soft never returns α or β directly as a node value. Thus, a node value may be outside the initial α and β range bounds set with a negamax function call. In contrast, fail-hard alpha-beta pruning always limits a node value in the range of α and β .

This implementation also shows optional move ordering prior to the *foreach loop* that evaluates child nodes. Move ordering^[2] is an optimization for alpha beta pruning that attempts to guess the most probable child nodes that yield the node's score. The algorithm searches those child nodes first. The result of good guesses is earlier and more frequent alpha/beta cut offs occur, thereby pruning additional game tree branches and remaining child nodes from the search tree.

119.3 NegaMax with Alpha Beta Pruning and Transposition Tables

Transposition tables selectively memorizes the values of nodes in the game tree. *Transposition* is a term reference that a given game board position can be reached in more than one way with differing game move sequences.

When negamax searches the game tree, and encounters the same node multiple times, a transposition table can return a previously computed value of the node, skipping potentially lengthy and duplicate re-computation of the node's value. Negamax performance improves particularly for game trees with many paths that lead to a given node in common.

The pseudo code that adds transposition table functions to negamax with alpha/beta pruning is given as follows:

```
function negamax(node, depth, α, β, color) alphaOrig := α // Transposition Table Lookup; node is the lookup key for ttEntry ttEntry := TranspositionTableLookup( node ) if ttEntry is valid and ttEntry.depth ≥ depth if ttEntry.Flag = EXACT return ttEntry.Value else if ttEntry.Flag = LOWERBOUND α := max( α, ttEntry.Value ) else if ttEntry.Flag = UPPERBOUND β := min( β,
```

```

ttEntry.Value) endif if  $\alpha \geq \beta$  return ttEntry.Value endif if depth = 0 or node is a terminal node return
color * the heuristic value of node bestValue :=  $-\infty$ 
childNodes := GenerateMoves(node) childNodes := OrderMoves(childNodes) foreach child in childNodes val
:= -negamax(child, depth - 1,  $-\beta$ ,  $-\alpha$ , -color) bestValue
:= max( bestValue, val )  $\alpha$  := max(  $\alpha$ , val ) if  $\alpha \geq \beta$ 
break // Transposition Table Store; node is the lookup
key for ttEntry ttEntry.Value := bestValue if bestValue  $\leq$ 
alphaOrig ttEntry.Flag := UPPEROBOUND else if best-
Value  $\geq \beta$  ttEntry.Flag := LOWERBOUND else ttEn-
try.Flag := EXACT endif ttEntry.depth := depth Trans-
positionTableStore( node, ttEntry ) return bestValue
Initial call for Player A's root node rootNegamaxValue :=
negamax( rootNode, depth,  $-\infty$ ,  $+\infty$ , 1)

```

Alpha/beta pruning and maximum search depth constraints in negamax can result in partial, inexact, and entirely skipped evaluation of nodes in a game tree. This complicates adding transposition table optimizations for negamax. It's insufficient to track only the node's *bestValue*, because *bestValue* may not be the node's true value. The code therefore must preserve and restore the relationship of *bestValue* with alpha/beta parameters and the search depth for each transposition table entry.

Transposition tables are typically lossy and will omit or overwrite previous values of certain game tree nodes in its tables. This is necessary since the number of nodes negamax visits often far exceeds the transposition table size. Lost or omitted table entries are non critical and won't affect the negamax result. However, lost entries may require negamax to re-compute certain game tree node values more frequently, thus affecting performance.

119.5 External links

- Negamax at the Chess Programming Wiki
- A C99 implementation of the Negamax algorithm for the Tic-Tac-Toe game

119.4 References

- George T. Heineman, Gary Pollice, and Stanley Selkow (2008). "Chapter 7:Path Finding in AI". *Algorithms in a Nutshell*. O'reilly Media. pp. 213–217. ISBN 978-0-596-51624-6.
- John P. Fishburn (1984). "Appendix A: Some Optimizations of α - β Search". *Analysis of Speedup in Distributed Algorithms (revision of 1981 PhD thesis)*. UMI Research Press. pp. 107–111. ISBN 0-8357-1527-2.

[1] Breuker, Dennis M. *Memory versus Search in Games*, Maastricht University, October 16, 1998

[2] Schaeffer, Jonathan *The History Heuristic and Alpha-Beta Search Enhancements in Practice*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 1989

Chapter 120

Newton's method

This article is about Newton's method for finding roots. For Newton's method for finding minima, see [Newton's method in optimization](#).

In numerical analysis, **Newton's method** (also known as the **Newton–Raphson method**), named after Isaac Newton and Joseph Raphson, is a method for finding successively better approximations to the **roots** (or zeroes) of a real-valued function.

$$x : f(x) = 0.$$

The Newton–Raphson method in one variable is implemented as follows:

Given a function f defined over the reals x , and its derivative f' , we begin with a first guess x_0 for a root of the function f . Provided the function satisfies all the assumptions made in the derivation of the formula, a better approximation x_1 is

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Geometrically, $(x_1, 0)$ is the intersection with the x -axis of the **tangent** to the **graph** of f at $(x_0, f(x_0))$.

The process is repeated as

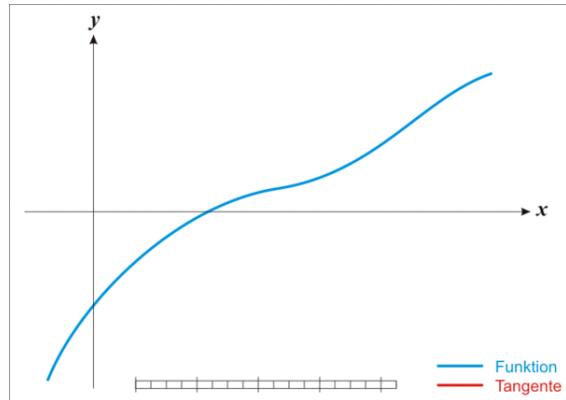
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

until a sufficiently accurate value is reached.

This algorithm is first in the class of **Householder's methods**, succeeded by **Halley's method**. The method can also be extended to complex functions and to systems of equations.

120.1 Description

The idea of the method is as follows: one starts with an initial guess which is reasonably close to the true



The function f is shown in blue and the tangent line is in red. We see that x_{n+1} is a better approximation than x_n for the root x of the function f .

root, then the function is approximated by its **tangent line** (which can be computed using the tools of **calculus**), and one computes the x -intercept of this tangent line (which is easily done with elementary algebra). This x -intercept will typically be a better approximation to the function's root than the original guess, and the method can be **iterated**.

Suppose $f : [a, b] \rightarrow \mathbf{R}$ is a differentiable function defined on the interval $[a, b]$ with values in the real numbers \mathbf{R} . The formula for converging on the root can be easily derived. Suppose we have some current approximation x_n . Then we can derive the formula for a better approximation, x_{n+1} by referring to the diagram on the right. The equation of the **tangent line** to the curve $y = f(x)$ at the point $x=x_n$ is

$$y = f'(x_n)(x - x_n) + f(x_n),$$

where, f' denotes the **derivative** of the function f .

The x -intercept of this line (the value of x such that $y=0$) is then used as the next approximation to the root, x_{n+1} . In other words, setting y to zero and x to x_{n+1} gives

$$0 = f'(x_n)(x_{n+1} - x_n) + f(x_n).$$

Solving for x_{n+1} gives

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

We start the process off with some arbitrary initial value x_0 . (The closer to the zero, the better. But, in the absence of any intuition about where the zero might lie, a “guess and check” method might narrow the possibilities to a reasonably small interval by appealing to the **intermediate value theorem**.) The method will usually converge, provided this initial guess is close enough to the unknown zero, and that $f'(x_0) \neq 0$. Furthermore, for a zero of multiplicity 1, the convergence is at least quadratic (see rate of convergence) in a neighbourhood of the zero, which intuitively means that the number of correct digits roughly at least doubles in every step. More details can be found in the analysis section below.

The **Householder's methods** are similar but have higher order for even faster convergence. However, the extra computations required for each step can slow down the overall performance relative to Newton's method, particularly if f or its derivatives are computationally expensive to evaluate.

120.2 History

The name “Newton's method” is derived from Isaac Newton's description of a special case of the method in *De analysi per aequationes numero terminorum infinitas* (written in 1669, published in 1711 by William Jones) and in *De metodis fluxionum et serierum infinitarum* (written in 1671, translated and published as *Method of Fluxions* in 1736 by John Colson). However, his method differs substantially from the modern method given above: Newton applies the method only to polynomials. He does not compute the successive approximations x_n , but computes a sequence of polynomials, and only at the end arrives at an approximation for the root x . Finally, Newton views the method as purely algebraic and makes no mention of the connection with calculus. Newton may have derived his method from a similar but less precise method by Vieta. The essence of Vieta's method can be found in the work of the Persian mathematician Sharaf al-Din al-Tusi, while his successor Jamshīd al-Kāshī used a form of Newton's method to solve $x^P - N = 0$ to find roots of N (Ypma 1995). A special case of Newton's method for calculating square roots was known much earlier and is often called the **Babylonian method**.

Newton's method was used by 17th-century Japanese mathematician Seki Kōwa to solve single-variable equations, though the connection with calculus was missing.

Newton's method was first published in 1685 in *A Treatise of Algebra both Historical and Practical* by John Wallis. In 1690, Joseph Raphson published a simplified description in *Analysis aequationum universalis*. Raphson again

viewed Newton's method purely as an algebraic method and restricted its use to polynomials, but he describes the method in terms of the successive approximations x_n instead of the more complicated sequence of polynomials used by Newton. Finally, in 1740, Thomas Simpson described Newton's method as an iterative method for solving general nonlinear equations using calculus, essentially giving the description above. In the same publication, Simpson also gives the generalization to systems of two equations and notes that Newton's method can be used for solving optimization problems by setting the gradient to zero.

Arthur Cayley in 1879 in *The Newton-Fourier imaginary problem* was the first to notice the difficulties in generalizing Newton's method to complex roots of polynomials with degree greater than 2 and complex initial values. This opened the way to the study of the theory of iterations of rational functions.

120.3 Practical considerations

Newton's method is an extremely powerful technique—in general the **convergence** is quadratic: as the method converges on the root, the difference between the root and the approximation is squared (the number of accurate digits roughly doubles) at each step. However, there are some difficulties with the method.

120.3.1 Difficulty in calculating derivative of a function

Newton's method requires that the derivative be calculated directly. An analytical expression for the derivative may not be easily obtainable and could be expensive to evaluate. In these situations, it may be appropriate to approximate the derivative by using the slope of a line through two nearby points on the function. Using this approximation would result in something like the **secant method** whose convergence is slower than that of Newton's method.

120.3.2 Failure of the method to converge to the root

It is important to review the proof of quadratic convergence of Newton's Method before implementing it. Specifically, one should review the assumptions made in the proof. For situations where the method fails to converge, it is because the assumptions made in this proof are not met.

Overshoot

If the first derivative is not well behaved in the neighborhood of a particular root, the method may overshoot, and diverge from that root. An example of a function with one root, for which the derivative is not well behaved in the neighborhood of the root, is

$$f(x) = |x|^a, \quad 0 < a < \frac{1}{2}$$

for which the root will be overshoot and the sequence of x will diverge. For $a = 1/2$, the root will still be overshoot, but the sequence will oscillate between two values. For $1/2 < a < 1$, the root will still be overshoot but the sequence will converge, and for $a \geq 1$ the root will not be overshoot at all.

In some cases, Newton's method can be stabilized by using successive over-relaxation, or the speed of convergence can be increased by using the same method.

Stationary point

If a stationary point of the function is encountered, the derivative is zero and the method will terminate due to division by zero.

Poor initial estimate

A large error in the initial estimate can contribute to non-convergence of the algorithm.

Mitigation of non-convergence

In a robust implementation of Newton's method, it is common to place limits on the number of iterations, bound the solution to an interval known to contain the root, and combine the method with a more robust root finding method.

120.3.3 Slow convergence for roots of multiplicity > 1

If the root being sought has multiplicity greater than one, the convergence rate is merely linear (errors reduced by a constant factor at each step) unless special steps are taken. When there are two or more roots that are close together then it may take many iterations before the iterates get close enough to one of them for the quadratic convergence to be apparent. However, if the multiplicity m of the root is known, one can use the following modified algorithm that preserves the quadratic convergence rate:

$$x_{n+1} = x_n - m \frac{f(x_n)}{f'(x_n)}. \quad [1]$$

This is equivalent to using successive over-relaxation. On the other hand, if the multiplicity m of the root is not known, it is possible to estimate m after carrying out one or two iterations, and then use that value to increase the rate of convergence.

120.4 Analysis

Suppose that the function f has a zero at α , i.e., $f(\alpha) = 0$, and f is differentiable in a neighborhood of α .

If f is continuously differentiable and its derivative is nonzero at α , then there exists a neighborhood of α such that for all starting values x_0 in that neighborhood, the sequence $\{x_n\}$ will converge to α .^[2]

If the function is continuously differentiable and its derivative is not 0 at α and it has a second derivative at α then the convergence is quadratic or faster. If the second derivative is not 0 at α then the convergence is merely quadratic. If the third derivative exists and is bounded in a neighborhood of α , then:

$$\Delta x_{i+1} = \frac{f''(\alpha)}{2f'(\alpha)} (\Delta x_i)^2 + O[\Delta x_i]^3,$$

where $\Delta x_i \triangleq x_i - \alpha$.

If the derivative is 0 at α , then the convergence is usually only linear. Specifically, if f is twice continuously differentiable, $f'(\alpha) = 0$ and $f''(\alpha) \neq 0$, then there exists a neighborhood of α such that for all starting values x_0 in that neighborhood, the sequence of iterates converges linearly, with rate $\log_{10} 2$ (Süli & Mayers, Exercise 1.6). Alternatively if $f'(\alpha) = 0$ and $f'(x) \neq 0$ for $x \neq \alpha$, x in a neighborhood U of α , α being a zero of multiplicity r , and if $f \in C^r(U)$ then there exists a neighborhood of α such that for all starting values x_0 in that neighborhood, the sequence of iterates converges linearly.

However, even linear convergence is not guaranteed in pathological situations.

In practice these results are local, and the neighborhood of convergence is not known in advance. But there are also some results on global convergence: for instance, given a right neighborhood U_+ of α , if f is twice differentiable in U_+ and if $f' \neq 0$, $f \cdot f'' > 0$ in U_+ , then, for each x_0 in U_+ the sequence x_k is monotonically decreasing to α .

120.4.1 Proof of quadratic convergence for Newton's iterative method

According to Taylor's theorem, any function $f(x)$ which has a continuous second derivative can be represented by an expansion about a point that is close to a root of $f(x)$. Suppose this root is α . Then the expansion of $f(\alpha)$ about x_n is:

where the Lagrange form of the Taylor series expansion remainder is

$$R_1 = \frac{1}{2!} f''(\xi_n)(\alpha - x_n)^2,$$

where ξ_n is in between x_n and α .

Since α is the root, (1) becomes:

Dividing equation (2) by $f'(x_n)$ and rearranging gives

Remembering that x_{n+1} is defined by

one finds that

$$\underbrace{\alpha - x_{n+1}}_{\epsilon_{n+1}} = \frac{-f''(\xi_n)}{2f'(x_n)} \underbrace{(\alpha - x_n)}_{\epsilon_n}^2.$$

That is,

Taking absolute value of both sides gives

Equation (6) shows that the rate of convergence is quadratic if the following conditions are satisfied:

1. $f'(x) \neq 0; \forall x \in I$ where I interval the is $[\alpha - r, \alpha + r]$ some for $r \geq |(\alpha - x_0)|$;
2. $f''(x)$ finite is, $\forall x \in I$;
3. x_0 sufficiently close to the root α

The term *sufficiently* close in this context means the following:

(a) Taylor approximation is accurate enough such that we can ignore higher order terms,

$$(b) \frac{1}{2} \left| \frac{f''(x_n)}{f'(x_n)} \right| < C \left| \frac{f''(\alpha)}{f'(\alpha)} \right|, \text{ some for } C < \infty,$$

$$(c) C \left| \frac{f''(\alpha)}{f'(\alpha)} \right| \epsilon_n < 1, \text{ for } n \in \mathbb{Z}^+ \cup \{0\} \text{ and } C \text{ (b) condition satisfying .}$$

Finally, (6) can be expressed in the following way:

$$|\epsilon_{n+1}| \leq M \epsilon_n^2$$

where M is the supremum of the variable coefficient of ϵ_n^2 on the interval I defined in the condition 1, that is:

$$M = \sup_{x \in I} \frac{1}{2} \left| \frac{f''(x)}{f'(x)} \right|.$$

The initial point x_0 has to be chosen such that conditions 1 through 3 are satisfied, where the third condition requires that $M |\epsilon_0| < 1$.

120.4.2 Basins of attraction

The basins of attraction—the regions of the real number line such that within each region iteration from any point leads to one particular root—can be infinite in number and arbitrarily small. For example,^[3] for the function $f(x) = x^3 - 2x^2 - 11x + 12$, the following initial conditions are in successive basins of attraction:

- 2.35287527 converges to 4;
- 2.35284172 converges to -3;
- 2.35283735 converges to 4;
- 2.352836327 converges to -3;
- 2.352836323 converges to 1.

120.5 Failure analysis

Newton's method is only guaranteed to converge if certain conditions are satisfied. If the assumptions made in the proof of quadratic convergence are met, the method will converge. For the following subsections, failure of the method to converge indicates that the assumptions made in the proof were not met.

120.5.1 Bad starting points

In some cases the conditions on the function that are necessary for convergence are satisfied, but the point chosen as the initial point is not in the interval where the method converges. This can happen, for example, if the function whose root is sought approaches zero asymptotically as x goes to ∞ or $-\infty$. In such cases a different method, such as bisection, should be used to obtain a better estimate for the zero to use as an initial point.

Iteration point is stationary

Consider the function:

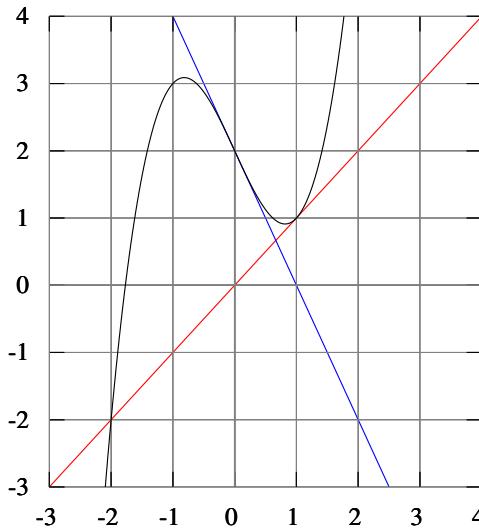
$$f(x) = 1 - x^2.$$

It has a maximum at $x = 0$ and solutions of $f(x) = 0$ at $x = \pm 1$. If we start iterating from the stationary point $x_0 = 0$ (where the derivative is zero), x_1 will be undefined, since the tangent at $(0,1)$ is parallel to the x -axis:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 0 - \frac{1}{0}.$$

The same issue occurs if, instead of the starting point, any iteration point is stationary. Even if the derivative is small but not zero, the next iteration will be a far worse approximation.

Starting point enters a cycle



The tangent lines of $x^3 - 2x + 2$ at 0 and 1 intersect the x-axis at 1 and 0 respectively, illustrating why Newton's method oscillates between these values for some starting points.

For some functions, some starting points may enter an infinite cycle, preventing convergence. Let

$$f(x) = x^3 - 2x + 2$$

and take 0 as the starting point. The first iteration produces 1 and the second iteration returns to 0 so the sequence will alternate between the two without converging to a root. In fact, this 2-cycle is stable: there are neighborhoods around 0 and around 1 from which all points iterate asymptotically to the 2-cycle (and hence not to the root of the function). In general, the behavior of the sequence can be very complex (see [Newton fractal](#)).

120.5.2 Derivative issues

If the function is not continuously differentiable in a neighborhood of the root then it is possible that Newton's method will always diverge and fail, unless the solution is guessed on the first try.

Derivative does not exist at root

A simple example of a function where Newton's method diverges is the cube root, which is continuous and infinitely differentiable, except for $x = 0$, where its derivative is undefined (this, however, does not affect the algorithm, since it will never require the derivative if the solution is already found):

$$f(x) = \sqrt[3]{x}.$$

For any iteration point x_n , the next iteration point will be:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^{\frac{1}{3}}}{\frac{1}{3} x_n^{\frac{1}{3}-1}} = x_n - 3x_n = -2x_n.$$

The algorithm overshoots the solution and lands on the other side of the y-axis, farther away than it initially was; applying Newton's method actually doubles the distances from the solution at each iteration.

In fact, the iterations diverge to infinity for every $f(x) = |x|^\alpha$, where $0 < \alpha < \frac{1}{2}$. In the limiting case of $\alpha = \frac{1}{2}$ (square root), the iterations will alternate indefinitely between points x_0 and $-x_0$, so they do not converge in this case either.

Discontinuous derivative

If the derivative is not continuous at the root, then convergence may fail to occur in any neighborhood of the root. Consider the function

$$f(x) = \begin{cases} 0 & \text{if } x = 0, \\ x + x^2 \sin\left(\frac{2}{x}\right) & \text{if } x \neq 0. \end{cases}$$

Its derivative is:

$$f'(x) = \begin{cases} 1 & \text{if } x = 0, \\ 1 + 2x \sin\left(\frac{2}{x}\right) - 2 \cos\left(\frac{2}{x}\right) & \text{if } x \neq 0. \end{cases}$$

Within any neighborhood of the root, this derivative keeps changing sign as x approaches 0 from the right (or from the left) while $f(x) \geq x - x^2 > 0$ for $0 < x < 1$.

So $f(x)/f'(x)$ is unbounded near the root, and Newton's method will diverge almost everywhere in any neighborhood of it, even though:

- the function is differentiable (and thus continuous) everywhere;
- the derivative at the root is nonzero;
- f is infinitely differentiable except at the root; and
- the derivative is bounded in a neighborhood of the root (unlike $f(x)/f'(x)$).

120.5.3 Non-quadratic convergence

In some cases the iterates converge but do not converge as quickly as promised. In these cases simpler methods converge just as quickly as Newton's method.

Zero derivative

If the first derivative is zero at the root, then convergence will not be quadratic. Indeed, let

$$f(x) = x^2$$

then $f'(x) = 2x$ and consequently $x - f(x)/f'(x) = x/2$. So convergence is not quadratic, even though the function is infinitely differentiable everywhere.

Similar problems occur even when the root is only “nearly” double. For example, let

$$f(x) = x^2(x - 1000) + 1.$$

Then the first few iterates starting at $x_0 = 1$ are 1, 0.500250376, 0.251062828, 0.127507934, 0.067671976, 0.041224176, 0.032741218, 0.031642362; it takes six iterations to reach a point where the convergence appears to be quadratic.

No second derivative

If there is no second derivative at the root, then convergence may fail to be quadratic. Indeed, let

$$f(x) = x + x^{\frac{4}{3}}.$$

Then

$$f'(x) = 1 + \frac{4}{3}x^{\frac{1}{3}}.$$

And

$$f''(x) = \frac{4}{9}x^{-\frac{2}{3}}$$

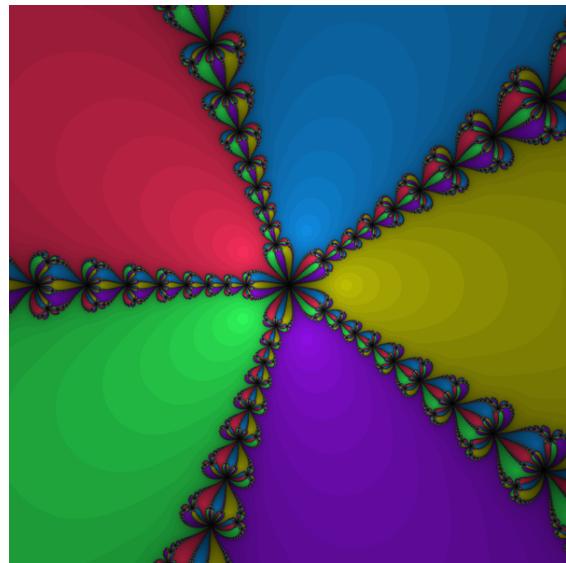
except when $x = 0$ where it is undefined. Given x_n ,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = \frac{\frac{1}{3}x_n^{\frac{4}{3}}}{(1 + \frac{4}{3}x_n^{\frac{1}{3}})}$$

which has approximately 4/3 times as many bits of precision as x_n has. This is less than the 2 times as many which would be required for quadratic convergence. So the convergence of Newton's method (in this case) is not quadratic, even though: the function is continuously differentiable everywhere; the derivative is not zero at the root; and f is infinitely differentiable except at the desired root.

120.6 Generalizations

120.6.1 Complex functions



Basins of attraction for $x^5 - 1 = 0$; darker means more iterations to converge.

Main article: [Newton fractal](#)

When dealing with complex functions, Newton's method can be directly applied to find their zeroes. Each zero has a **basin of attraction** in the complex plane, the set of all starting values that cause the method to converge to that particular zero. These sets can be mapped as in the image shown. For many complex functions, the boundaries of the basins of attraction are **fractals**.

In some cases there are regions in the complex plane which are not in any of these basins of attraction, meaning the iterates do not converge. For example,^[4] if one uses a real initial condition to seek a root of $x^2 + 1$, all subsequent iterates will be real numbers and so the iterations cannot converge to either root, since both roots are

non-real. In this case almost all real initial conditions lead to chaotic behavior, while some initial conditions iterate either to infinity or to repeating cycles of any finite length.

120.6.2 Nonlinear systems of equations

k variables, k functions

One may also use Newton's method to solve systems of k (non-linear) equations, which amounts to finding the zeroes of continuously differentiable functions $F : \mathbf{R}^k \rightarrow \mathbf{R}^k$. In the formulation given above, one then has to left multiply with the inverse of the k -by- k Jacobian matrix $JF(x_n)$ instead of dividing by $f'(x_n)$.

Rather than actually computing the inverse of this matrix, one can save time by solving the system of linear equations

$$J_F(x_n)(x_{n+1} - x_n) = -F(x_n)$$

for the unknown $x_{n+1} - x_n$.

k variables, m equations, with $m > k$

The k -dimensional Newton's method can be used to solve systems of $>k$ (non-linear) equations as well if the algorithm uses the generalized inverse of the non-square Jacobian matrix $J^+ = ((J^T J)^{-1}) J^T$ instead of the inverse of J . If the nonlinear system has no solution, the method attempts to find a solution in the non-linear least squares sense. See Gauss–Newton algorithm for more information.

120.6.3 Nonlinear equations in a Banach space

Another generalization is Newton's method to find a root of a functional F defined in a Banach space. In this case the formulation is

$$X_{n+1} = X_n - [F'(X_n)]^{-1} F(X_n),$$

where $F'(X_n)$ is the Fréchet derivative computed at X_n . One needs the Fréchet derivative to be boundedly invertible at each X_n in order for the method to be applicable. A condition for existence of and convergence to a root is given by the Newton–Kantorovich theorem.

120.6.4 Nonlinear equations over p -adic numbers

In p -adic analysis, the standard method to show a polynomial equation in one variable has a p -adic root is Hensel's

lemma, which uses the recursion from Newton's method on the p -adic numbers. Because of the more stable behavior of addition and multiplication in the p -adic numbers compared to the real numbers (specifically, the unit ball in the p -adics is a ring), convergence in Hensel's lemma can be guaranteed under much simpler hypotheses than in the classical Newton's method on the real line.

120.6.5 Newton-Fourier method

The Newton-Fourier method is Joseph Fourier's extension of Newton's method to provide bounds on the absolute error of the root approximation, while still providing quadratic convergence.

Assume that $f(x)$ is twice continuously differentiable on $[a, b]$ and that f contains a root in this interval. Assume that $f'(x)f''(x) \neq 0$ on this interval (this is the case for instance if $f(a) < 0$, $f(b) > 0$, and $f'(x) > 0$, and $f''(x) > 0$ on this interval). This guarantees that there is a unique root on this interval, call it α . If it is concave down instead of concave up then replace $f(x)$ by $-f(x)$ since they have the same roots.

Let $x_0 = b$ be the right endpoint of the interval and let $z_0 = a$ be the left endpoint of the interval. Given x_n , define $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$, which is just Newton's method as before. Then define $z_{n+1} = z_n - \frac{f(z_n)}{f'(x_n)}$ and note that the denominator has $f'(x_n)$ and not $f'(z_n)$. The iterates x_n will be strictly decreasing to the root while the iterates z_n will be strictly increasing to the root. Also, $\lim_{n \rightarrow \infty} \frac{x_{n+1} - z_{n+1}}{(x_n - z_n)^2} = \frac{f''(\alpha)}{2f'(\alpha)}$ so that distance between x_n and z_n decreases quadratically.

120.6.6 Quasi-Newton methods

When the Jacobian is unavailable or too expensive to compute at every iteration, a Quasi-Newton method can be used.

120.7 Applications

120.7.1 Minimization and maximization problems

Main article: Newton's method in optimization

Newton's method can be used to find a minimum or maximum of a function. The derivative is zero at a minimum or maximum, so minima and maxima can be found by applying Newton's method to the derivative. The iteration becomes:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}.$$

120.7.2 Multiplicative inverses of numbers and power series

An important application is Newton–Raphson division, which can be used to quickly find the reciprocal of a number, using only multiplication and subtraction.

Finding the reciprocal of a amounts to finding the root of the function

$$f(x) = a - \frac{1}{x}$$

Newton's iteration is

$$\begin{aligned} x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} \\ &= x_n - \frac{a - \frac{1}{x_n}}{\frac{1}{x_n^2}} \\ &= x_n (2 - ax_n) \end{aligned}$$

Therefore, Newton's iteration needs only two multiplications and one subtraction.

This method is also very efficient to compute the multiplicative inverse of a power series.

120.7.3 Solving transcendental equations

Many transcendental equations can be solved using Newton's method. Given the equation

$$g(x) = h(x),$$

with $g(x)$ and/or $h(x)$ a transcendental function, one writes

$$f(x) = g(x) - h(x).$$

The values of x that solves the original equation are then the roots of $f(x)$, which may be found via Newton's method.

120.8 Examples

120.8.1 Square root of a number

Consider the problem of finding the square root of a number. Newton's method is one of many methods of computing square roots.

For example, if one wishes to find the square root of 612, this is equivalent to finding the solution to

$$x^2 = 612$$

The function to use in Newton's method is then,

$$f(x) = x^2 - 612$$

with derivative,

$$f'(x) = 2x.$$

With an initial guess of 10, the sequence given by Newton's method is

$$\begin{array}{rclcl} x_1 & = & x_0 - \frac{f(x_0)}{f'(x_0)} & = & 10 - \frac{10^2 - 612}{2 \cdot 10} = 35.6 \\ x_2 & = & x_1 - \frac{f(x_1)}{f'(x_1)} & = & 35.6 - \frac{35.6^2 - 612}{2 \cdot 35.6} = \underline{26.39550561797} \\ x_3 & = & \vdots & = & \vdots = \underline{24.79063549245} \\ x_4 & = & \vdots & = & \vdots = \underline{24.73868829407} \\ x_5 & = & \vdots & = & \vdots = \underline{24.73863375376} \end{array}$$

where the correct digits are underlined. With only a few iterations one can obtain a solution accurate to many decimal places.

120.8.2 Solution of $\cos(x) = x^3$

Consider the problem of finding the positive number x with $\cos(x) = x^3$. We can rephrase that as finding the zero of $f(x) = \cos(x) - x^3$. We have $f'(x) = -\sin(x) - 3x^2$. Since $\cos(x) \leq 1$ for all x and $x^3 > 1$ for $x > 1$, we know that our solution lies between 0 and 1. We try a starting value of $x_0 = 0.5$. (Note that a starting value of 0 will lead to an undefined result, showing the importance of using a starting point that is close to the solution.)

$$\begin{array}{rclcl} x_1 & = & x_0 - \frac{f(x_0)}{f'(x_0)} & = & 0.5 - \frac{\cos(0.5) - (0.5)^3}{-\sin(0.5) - 3(0.5)^2} = 1.112141 \\ x_2 & = & x_1 - \frac{f(x_1)}{f'(x_1)} & = & \vdots = \underline{0.909672} \\ x_3 & = & \vdots & = & \vdots = \underline{0.867263} \\ x_4 & = & \vdots & = & \vdots = \underline{0.865477} \\ x_5 & = & \vdots & = & \vdots = \underline{0.865474} \\ x_6 & = & \vdots & = & \vdots = \underline{0.865474} \end{array}$$

The correct digits are underlined in the above example. In particular, x_6 is correct to the number of decimal places given. We see that the number of correct digits after the decimal point increases from 2 (for x_3) to 5 and 10, illustrating the quadratic convergence.

120.9 Pseudocode

The following is an example of using the Newton's Method to help find a root of a function f which has derivative f' .

The initial guess will be $x_0 = 1$ and the function will be $f(x) = x^2 - 2$ so that $f'(x) = 2x$.

Each new iterative of Newton's method will be denoted by x_1 . We will check during the computation whether the denominator ($yprime$) becomes too small (smaller than ϵ), which would be the case if $f'(x_n) \approx 0$, since otherwise a large amount of error could be introduced.

```
%These choices depend on the problem being solved
x0 = 1 %The initial value
f = @(x) x^2 - 2 %The function whose root we are trying to find
fprime = @(x) 2*x %The derivative of f(x)
tolerance = 10^(-7) %7 digit accuracy is desired
epsilon = 10^(-14) %Don't want to divide by a number smaller than this
maxIterations = 20 %Don't allow the iterations to continue indefinitely
haveWeFoundSolution = false %Have not converged to a solution yet
for i = 1 : maxIterations
    y = f(x0)
    yprime = fprime(x0)
    if(abs(yprime) < epsilon) %Don't want to divide by too small of a number
        % denominator is too small
        break; %Leave the loop end
    x1 = x0 - y/yprime
    %Do Newton's computation
    if(abs(x1 - x0)/abs(x1) < tolerance) %If the result is within the desired tolerance
        haveWeFoundSolution = true
        break; %Done, so leave the loop end
    x0 = x1 %Update x0 to start the process again
end
if (haveWeFoundSolution) ...
    % x1 is a solution within tolerance and maximum number of iterations
else ...
    % did not converge
end
```

120.10 See also

- Aitken's delta-squared process
- Bisection method
- Euler method
- Fast inverse square root
- Fisher scoring
- Gradient descent
- Integer square root
- Laguerre's method
- Leonid Kantorovich, who initiated the convergence analysis of Newton's method in Banach spaces.
- Methods of computing square roots
- Newton's method in optimization
- Richardson extrapolation

- Root-finding algorithm
- Secant method
- Steffensen's method
- Subgradient method

120.11 References

- [1] "Accelerated and Modified Newton Methods".
- [2] Ryaben'kii, Victor S.; Tsynkov, Semyon V. (2006), *A Theoretical Introduction to Numerical Analysis*, CRC Press, p. 243, ISBN 9781584886075.
- [3] Dence, Thomas, "Cubics, chaos and Newton's method", *Mathematical Gazette* 81, November 1997, 403-408.
- [4] Strang, Gilbert, "A chaotic search for i ", "The College Mathematics Journal" 22, January 1991, pp. 3-12 (esp. p. 6).
- Kendall E. Atkinson, *An Introduction to Numerical Analysis*, (1989) John Wiley & Sons, Inc, ISBN 0-471-62489-6
- Tjalling J. Ypma, Historical development of the Newton-Raphson method, *SIAM Review* 37 (4), 531–551, 1995. doi:10.1137/1037125.
- Bonnans, J. Frédéric; Gilbert, J. Charles; Lemaréchal, Claude; Sagastizábal, Claudia A. (2006). *Numerical optimization: Theoretical and practical aspects*. Universitext (Second revised ed. of translation of 1997 French ed.). Berlin: Springer-Verlag. pp. xiv+490. doi:10.1007/978-3-540-35447-5. ISBN 3-540-35445-X. MR 2265882.
- P. Deuflhard, *Newton Methods for Nonlinear Problems. Affine Invariance and Adaptive Algorithms*. Springer Series in Computational Mathematics, Vol. 35. Springer, Berlin, 2004. ISBN 3-540-21099-7.
- C. T. Kelley, *Solving Nonlinear Equations with Newton's Method*, no 1 in Fundamentals of Algorithms, SIAM, 2003. ISBN 0-89871-546-6.
- J. M. Ortega, W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*. Classics in Applied Mathematics, SIAM, 2000. ISBN 0-89871-461-3.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Chapter 9. Root Finding and Nonlinear Sets of Equations Importance Sampling". *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.. See especially Sections 9.4, 9.6, and 9.7.

- Endre Süli and David Mayers, *An Introduction to Numerical Analysis*, Cambridge University Press, 2003. ISBN 0-521-00794-1.
- Kaw, Autar; Kalu, Egwu (2008). “Numerical Methods with Applications” (1st ed.).

120.12 External links

- Hazewinkel, Michiel, ed. (2001), “Newton method”, *Encyclopedia of Mathematics*, Springer, ISBN 978-1-55608-010-4
- Weisstein, Eric W., “Newton’s Method”, *MathWorld*.

Chapter 121

NEWUOA

NEWUOA^[1] is a numerical optimization algorithm by Michael J. D. Powell. It is also the name of Powell's Fortran 77 implementation of the algorithm.

NEWUOA solves unconstrained optimization problems without using derivatives, which makes it a derivative-free algorithm. The algorithm is iterative, and exploits trust region technique. On each iteration, the algorithm establishes a model function Q_k by quadratic interpolation, and then minimizes Q_k within a trust region.

One important feature of NEWUOA algorithm is the least Frobenius norm updating ^[2] technique. Suppose that the objective function f has n variables, and one wants to uniquely determine the quadratic model Q_k by purely interpolating the function values of f , then it is necessary to evaluate f at $(n+1)(n+2)/2$ points. But this is impractical when n is large, because the function values are supposed to be expensive in derivative-free optimization. In NEWUOA, the model Q_k interpolates only m (an integer between $n+2$ and $(n+1)(n+2)/2$, typically of order n) function values of f , and the remaining degree of freedom is taken up by minimizing the Frobenius norm of $\nabla^2 Q_k - \nabla^2 Q_{k-1}$. This technique mimics the least change secant updates ^[3] for Quasi-Newton methods, and can be considered as the derivative-free version of PSB update (Powell's Symmetric Broyden update).^[4]

To construct the models, NEWUOA maintains a set of interpolation points throughout the iterations. The update of this set is another feature of NEWUOA.^[1]

NEWUOA algorithm was developed from UOBYQA (Unconstrained Optimization BY Quadratic Approximation).^{[5][6]} A major difference between them is that UOBYQA constructs quadratic models by interpolating the objective function at $(n+1)(n+2)/2$ points.

NEWUOA software was released on December 16, 2004.^[7] It can solve unconstrained optimization problems of a few hundreds variables to high precision without using derivatives.^[1] In the software, m is set to $2n+1$ by default.^[6]

Other derivative-free optimization algorithms by Powell include COBYLA, UOBYQA, BOBYQA, and LINCOA.^[7] BOBYQA and LINCOA are extensions of

NEWUOA to bound constrained and linearly constrained optimization respectively.

Powell did not explain how he coined the name "NEWUOA" either in the introducing report ^[1] or in the software,^[6] although COBYLA, UOBYQA, BOBYQA and LINCOA are all named by acronyms.

121.1 See also

- TOLMIN
- COBYLA
- UOBYQA
- BOBYQA
- LINCOA

121.2 References

- [1] Powell, M. J. D. (November 2004). "The NEWUOA software for unconstrained optimization without derivatives (Report)". Department of Applied Mathematics and Theoretical Physics, Cambridge University. DAMTP 2004/NA05. Retrieved 2014-01-14.
- [2] Powell, M. J. D. (2004). "Least Frobenius norm updating of quadratic models that satisfy interpolation conditions". *Mathematical Programming* (Springer) **100**: 183–215. doi:10.1007/s10107-003-0490-7.
- [3] Dennis, Jr., J. E.; Schnabel, R. B. (1979). "Least Change Secant Updates for Quasi-Newton Methods". *SIAM Review* (SIAM) **21**: 443–459. doi:10.1137/1021091. Retrieved 2014-01-14.
- [4] Powell, M. J. D. (2013). "Beyond symmetric Broyden for updating quadratic models in minimization without derivatives". *Mathematical Programming* (Springer) **138**: 475–500. doi:10.1007/s10107-011-0510-y.
- [5] Powell, M. J. D. (2002). "UOBYQA: unconstrained optimization by quadratic approximation". *Mathematical Programming* (Springer) **92**: 555–582. doi:10.1007/s101070100290.

- [6] “Source code of NEWUOA software”. Retrieved 2014-01-14.
- [7] “A repository of Powell’s software”. Retrieved 2014-01-14.

121.3 External links

- Source code of NEWUOA software

Chapter 122

Nonlinear programming

In mathematics, **nonlinear programming** (NLP) is the process of solving an optimization problem defined by a system of equalities and inequalities, collectively termed constraints, over a set of unknown real variables, along with an objective function to be maximized or minimized, where some of the constraints or the objective function are nonlinear.^[1] It is the sub-field of Mathematical optimization that deals with problems that are not linear.

122.1 Applicability

A typical nonconvex problem is that of optimising transportation costs by selection from a set of transportation methods, one or more of which exhibit economies of scale, with various connectivities and capacity constraints. An example would be petroleum product transport given a selection or combination of pipeline, rail tanker, road tanker, river barge, or coastal tankship. Owing to economic batch size the cost functions may have discontinuities in addition to smooth changes.

Modern engineering practice involves much numerical optimization. Except in certain narrow but important cases such as passive electronic circuits, engineering problems are non-linear, and they are usually very complicated.

In experimental science, some simple data analysis (such as fitting a spectrum with a sum of peaks of known location and shape but unknown magnitude) can be done with linear methods, but in general these problems, also, are non-linear. Typically, one has a theoretical model of the system under study with variable parameters in it and a model the experiment or experiments, which may also have unknown parameters. One tries to find a best fit numerically. In this case one often wants a measure of the precision of the result, as well as the best fit itself.

122.2 The general non-linear optimization problem (NLP)

The problem can be stated simply as:

$$\max_{x \in X} f(x)$$

or

$$\min_{x \in X} f(x)$$

where

$$f : R^n \rightarrow R$$

$$x \in R^n$$

s.t. (subject to)

$$h_i(x) = 0, i \in I = 1, \dots, p$$

$$g_j(x) \leq 0, j \in J = 1, \dots, m$$

122.3 Possible solutions

- feasible, that is, for an optimal solution x subject to constraints, the objective function f is either maximized or minimized.
- unbounded, that is, for some x subject to constraints, the objective function f is either ∞ or $-\infty$.
- infeasible, that is, there is no solution x that is subject to constraints.

122.4 Methods for solving the problem

If the objective function f is linear and the constrained space is a polytope, the problem is a linear programming problem, which may be solved using well known linear programming solutions.

If the objective function is **concave** (maximization problem), or **convex** (minimization problem) and the constraint set is **convex**, then the program is called convex and general methods from **convex optimization** can be used in most cases.

If the objective function is a ratio of a concave and a convex function (in the maximization case) and the constraints are convex, then the problem can be transformed to a convex optimization problem using **fractional programming** techniques.

Several methods are available for solving nonconvex problems. One approach is to use special formulations of linear programming problems. Another method involves the use of **branch and bound** techniques, where the program is divided into subclasses to be solved with convex (minimization problem) or linear approximations that form a lower bound on the overall cost within the subdivision. With subsequent divisions, at some point an actual solution will be obtained whose cost is equal to the best lower bound obtained for any of the approximate solutions. This solution is optimal, although possibly not unique. The algorithm may also be stopped early, with the assurance that the best possible solution is within a tolerance from the best point found; such points are called ϵ -optimal. Terminating to ϵ -optimal points is typically necessary to ensure finite termination. This is especially useful for large, difficult problems and problems with uncertain costs or values where the uncertainty can be estimated with an appropriate reliability estimation.

Under **differentiability** and **constraint qualifications**, the **Karush–Kuhn–Tucker (KKT)** conditions provide necessary conditions for a solution to be optimal. Under convexity, these conditions are also sufficient. If some of the functions are non-differentiable, **subdifferential** versions of **Karush–Kuhn–Tucker (KKT)** conditions are available.^[2]

122.5 Examples

122.5.1 2-dimensional example

A simple problem can be defined by the constraints

$$x_1 \geq 0$$

$$x_2 \geq 0$$

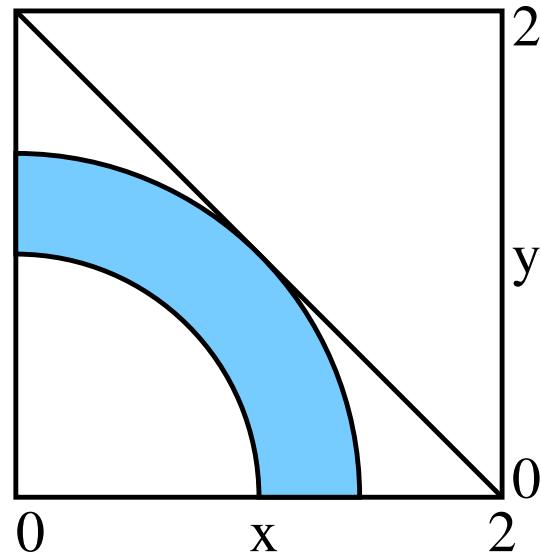
$$x_1^2 + x_2^2 \geq 1$$

$$x_1^2 + x_2^2 \leq 2$$

with an objective function to be maximized

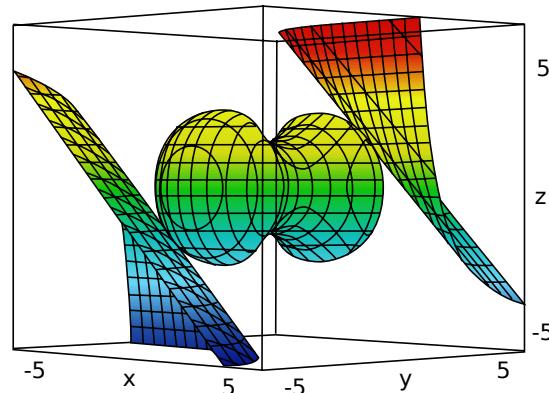
$$f(\mathbf{x}) = x_1 + x_2$$

where $\mathbf{x} = (x_1, x_2)$. Solve 2-D Problem.



The intersection of the line with the constrained space represents the solution

122.5.2 3-dimensional example



The intersection of the top surface with the constrained space in the center represents the solution

Another simple problem can be defined by the constraints

$$x_1^2 - x_2^2 + x_3^2 \leq 2$$

$$x_1^2 + x_2^2 + x_3^2 \leq 10$$

with an objective function to be maximized

$$f(\mathbf{x}) = x_1 x_2 + x_2 x_3$$

where $\mathbf{x} = (x_1, x_2, x_3)$. Solve 3-D Problem.

122.6 Applications

Nonlinear optimization methods are used in engineering, for example to construct computational models of oil reservoirs.^[3]

122.7 See also

- Curve fitting
- Least squares minimization
- Linear programming
- nl (format)
- Mathematical optimization
- List of optimization software
- Werner Fenchel

122.8 References

- [1] Bertsekas, Dimitri P. (1999). *Nonlinear Programming* (Second ed.). Cambridge, MA.: Athena Scientific. ISBN 1-886529-00-0.
- [2] Ruszczyński, Andrzej (2006). *Nonlinear Optimization*. Princeton, NJ: Princeton University Press. pp. xii+454. ISBN 978-0691119151. MR 2199043.
- [3] History matching production data and uncertainty assessment with an efficient TSVD parameterization algorithm, <http://www.sciencedirect.com/science/article/pii/S0920410513003227>

122.9 Further reading

- Avriel, Mordecai (2003). *Nonlinear Programming: Analysis and Methods*. Dover Publishing. ISBN 0-486-43227-0.
- Bazaraa, Mokhtar S. and Shetty, C. M. (1979). *Nonlinear programming. Theory and algorithms*. John Wiley & Sons. ISBN 0-471-78610-1.
- Bertsekas, Dimitri P. (1999). *Nonlinear Programming: 2nd Edition*. Athena Scientific. ISBN 1-886529-00-0.
- Bonnans, J. Frédéric; Gilbert, J. Charles; Lemaréchal, Claude; Sagastizábal, Claudia A. (2006). *Numerical optimization: Theoretical and practical aspects*. Universitext (Second revised ed. of translation of 1997 French ed.). Berlin: Springer-Verlag. pp. xiv+490. doi:10.1007/978-3-540-35447-5. ISBN 3-540-35445-X. MR 2265882.
- Luenberger, David G.; Ye, Yinyu (2008). *Linear and nonlinear programming*. International Series in Operations Research & Management Science **116** (Third ed.). New York: Springer. pp. xiv+546. ISBN 978-0-387-74502-2. MR 2423726.

• Nocedal, Jorge and Wright, Stephen J. (1999). *Numerical Optimization*. Springer. ISBN 0-387-98793-2.

• Jan Brinkhuis and Vladimir Tikhomirov, 'Optimization: Insights and Applications', 2005, Princeton University Press

122.10 External links

- Nonlinear programming FAQ
- Mathematical Programming Glossary
- Nonlinear Programming Survey OR/MS Today
- Overview of Optimization in Industry

Chapter 123

Ordered subset expectation maximization

This article is about an algorithm. For Israeli food corporation, see [Osem \(company\)](#).

In mathematical optimization, the **ordered subset expectation maximization** (OSEM) method is an iterative method that is used in computed tomography.

In applications in medical imaging, the OSEM method is used for positron emission tomography, for single photon emission computed tomography, and for X-ray computed tomography.

The OSEM method is related to the expectation maximization (EM) method of statistics. The OSEM method is also related to methods of [filtered back projection](#).

123.1 References

- Hudson, H.M., Larkin, R.S. (1994) “Accelerated image reconstruction using ordered subsets of projection data”, *IEEE Trans. Medical Imaging*, 13 (4), 601–609 doi:[10.1109/42.363108](https://doi.org/10.1109/42.363108)
- Hutton, Brian F., Hudson, H. Malcolm, Beekman, Freek J. (1997) “A clinical perspective of accelerated statistical reconstruction”, *European Journal of Nuclear Medicine and Molecular Imaging*, 24 (7), 797–808 doi:[10.1007/BF00879671](https://doi.org/10.1007/BF00879671)
- Xuan Liu; Comtat, C. ; Michel, C. ; Kinahan, P. ; Defrise, M. ; Townsend, D. (2001) “Comparison of 3-D reconstruction with 3D-OSEM and with FORE+OSEM for PET”, *IEEE Transactions on Medical Imaging*, 20 (8), 804–814 doi:[10.1109/42.938248](https://doi.org/10.1109/42.938248)

123.2 External links

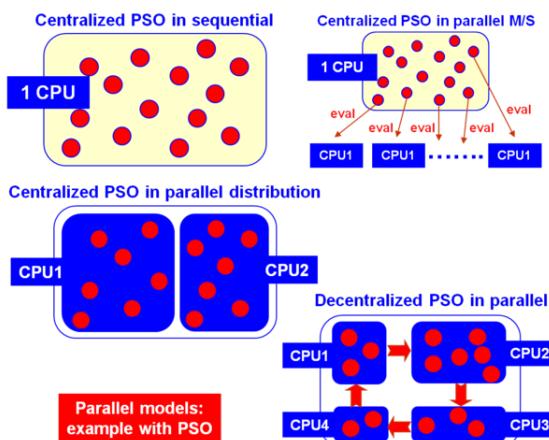
- Home page with IEEE paper and more information
- Software

Chapter 124

Parallel metaheuristic

Parallel metaheuristic is a class of **new** advanced techniques that are capable of reducing both the numerical effort and the run time of a **metaheuristic**. To this end, concepts and technologies from the field of parallelism in **computer science** are used to enhance and even completely modify the behavior of existing metaheuristics. Just as it exists a long list of metaheuristics like **evolutionary algorithms**, **particle swarm**, **ant colony optimization**, **simulated annealing**, etc. it also exists a large set of different techniques strongly or loosely based in these ones, whose behavior encompasses the multiple parallel execution of algorithm components that cooperate in some way to solve a problem on a given parallel hardware platform.

124.1 Background



An example of different implementations of the same PSO metaheuristic model.

In practice, optimization (and searching, and learning) problems are often **NP-hard**, complex, and time consuming. Two major approaches are traditionally used to tackle these problems: exact methods and **metaheuristics**. Exact methods allow to find exact solutions but are often impractical as they are extremely time-consuming for real-world problems (large dimension, hardly constrained, multimodal, time-varying, epistatic problems). Conversely, metaheuristics provide sub-optimal (some-

times optimal) solutions in a reasonable time. Thus, metaheuristics usually allow to meet the resolution delays imposed in the industrial field as well as they allow to study general problem classes instead that particular problem instances. In general, many of the best performing techniques in precision and effort to solve complex and real-world problems are metaheuristics. Their fields of application range from combinatorial optimization, bioinformatics, and telecommunications to economics, software engineering, etc. These fields are full of many tasks needing fast solutions of high quality. See for more details on complex applications.

Metaheuristics fall in two categories: **trajectory-based** metaheuristics and **population-based** metaheuristics. The main difference of these two kind of methods relies in the number of tentative solutions used in each step of the (iterative) algorithm. A trajectory-based technique starts with a single initial solution and, at each step of the search, the current solution is replaced by another (often the best) solution found in its neighborhood. It is usual that trajectory-based metaheuristics allow to quickly find a locally optimal solution, and so they are called **exploitation-oriented** methods promoting intensification in the search space. On the other hand, population-based algorithms make use of a population of solutions. The initial population is in this case randomly generated (or created with a **greedy algorithm**), and then enhanced through an iterative process. At each generation of the process, the whole population (or a part of it) is replaced by newly generated individuals (often the best ones). These techniques are called **exploration-oriented** methods, since their main ability resides in the diversification in the search space.

Most basic metaheuristics are sequential. Although their utilization allows to significantly reduce the temporal complexity of the search process, this latter remains high for real-world problems arising in both academic and industrial domains. Therefore, parallelism comes as a natural way not to only reduce the search time, but also to improve the quality of the provided solutions.

For a comprehensive discussion on how parallelism can be mixed with metaheuristics see .

124.2 Parallel trajectory-based metaheuristics

Metaheuristics for solving optimization problems could be viewed as *walks through neighborhoods* tracing search trajectories through the solution domains of the problem at hands:

Algorithm: Sequential trajectory-based general pseudo-code

```
Generate(s(0)); // Initial solution
t := 0; // Numerical step
while not Termination Criterion(s(t)) do
...s'(t) := SelectMove(s(t)); // Exploration of the neighborhood
...if AcceptMove(s'(t)) then
...s(t) := ApplyMove(s'(t));
...t := t+1;
endwhile
```

Walks are performed by iterative procedures that allow moving from one solution to another one in the solution space (see the above algorithm). This kind of metaheuristics perform the moves in the neighborhood of the current solution, i.e., they have a perturbative nature. The walks start from a solution randomly generated or obtained from another optimization algorithm. At each iteration, the current solution is replaced by another one selected from the set of its neighboring candidates. The search process is stopped when a given condition is satisfied (a maximum number of generation, find a solution with a target quality, stuck for a given time, . . .).

A powerful way to achieve high computational efficiency with trajectory-based methods is the use of parallelism. Different parallel models have been proposed for trajectory-based metaheuristics, and three of them are commonly used in the literature: the parallel *multi-start* model, the parallel exploration and evaluation of the *neighborhood* (or parallel moves model), and the parallel *evaluation* of a single solution (or move acceleration model):

- **Parallel multi-start model:** It consists in simultaneously launching several trajectory-based methods for computing better and robust solutions. They may be heterogeneous or homogeneous, independent or cooperative, start from the same or different solution(s), and configured with the same or different parameters.

- **Parallel moves model:** It is a low-level master-slave model that does not alter the behavior of the heuristic. A sequential search would compute the same result but slower. At the beginning of each iteration, the master duplicates the current solution between distributed nodes. Each one separately manages their candidate/solution and the results are returned to the master.

- **Move acceleration model:** The quality of each move is evaluated in a parallel centralized way. That model is particularly interesting when the evaluation function can

be itself parallelized as it is CPU time-consuming and/or I/O intensive. In that case, the function can be viewed as an aggregation of a certain number of partial functions that can be run in parallel.

124.3 Parallel population-based metaheuristics

Population-based metaheuristic are stochastic search techniques that have been successfully applied in many real and complex applications (epistatic, multimodal, multi-objective, and highly constrained problems). A population-based algorithm is an iterative technique that applies stochastic operators on a pool of individuals: the population (see the algorithm below). Every individual in the population is the encoded version of a tentative solution. An evaluation function associates a fitness value to every individual indicating its suitability to the problem. Iteratively, the probabilistic application of variation operators on selected individuals guides the population to tentative solutions of higher quality. The most well-known metaheuristic families based on the manipulation of a population of solutions are evolutionary algorithms (EAs), ant colony optimization (ACO), particle swarm optimization (PSO), scatter search (SS), differential evolution (DE), and estimation distribution algorithms (EDA).

Algorithm: Sequential population-based metaheuristic pseudo-code

```
Generate(P(0)); // Initial population
t := 0; // Numerical step
while not Termination Criterion(P(t)) do
...Evaluate(P(t)); // Evaluation of the population
...P''(t) := Apply Variation Operators(P'(t)); // Generation of new solutions
...P(t + 1) := Replace(P(t), P''(t)); // Building the next population
...t := t + 1;
endwhile
```

For non-trivial problems, executing the reproductive cycle of a simple population-based method on long individuals and/or large populations usually requires high computational resources. In general, evaluating a *fitness* function for every individual is frequently the most costly operation of this algorithm. Consequently, a variety of algorithmic issues are being studied to design efficient techniques. These issues usually consist of defining new operators, hybrid algorithms, parallel models, and so on.

Parallelism arises naturally when dealing with populations, since each of the individuals belonging to it is an independent unit (at least according to the *Pittsburg* style, although there are other approaches like the *Michigan* one which do not consider the individual as independent units). Indeed, the performance of population-based algorithms is often improved when running in par-

allel. Two parallelizing strategies are specially focused on population-based algorithms:

(1) **Parallelization of computations**, in which the operations commonly applied to each of the individuals are performed in parallel, and

(2) **Parallelization of population**, in which the population is split in different parts that can be simply exchanged or evolved separately, and then joined later.

In the beginning of the parallelization history of these algorithms, the well-known **master-slave** (also known as *global parallelization* or *farming*) method was used. In this approach, a central processor performs the selection operations while the associated slave processors (workers) run the variation operator and the evaluation of the fitness function. This algorithm has the same behavior as the sequential one, although its computational efficiency is improved, especially for time consuming objective functions. On the other hand, many researchers use a pool of processors to speed up the execution of a sequential algorithm, just because independent runs can be made more rapidly by using several processors than by using a single one. In this case, no interaction at all exists between the independent runs.

However, actually most parallel population-based techniques found in the literature utilize some kind of spatial disposition for the individuals, and then parallelize the resulting chunks in a pool of processors. Among the most widely known types of structured metaheuristics, the **distributed** (or coarse grain) and **cellular** (or fine grain) algorithms are very popular optimization procedures.

In the case of distributed ones, the population is partitioned in a set of subpopulations (islands) in which isolated serial algorithms are executed. Sparse exchanges of individuals are performed among these islands with the goal of introducing some diversity into the subpopulations, thus preventing search of getting stuck in local optima. In order to design a distributed metaheuristic, we must take several decisions. Among them, a chief decision is to determine the migration policy: topology (logical links between the islands), migration rate (number of individuals that undergo migration in every exchange), migration frequency (number of steps in every subpopulation between two successive exchanges), and the selection/replacement of the migrants.

In the case of a cellular method, the concept of neighborhood is introduced, so that an individual may only interact with its nearby neighbors in the breeding loop. The overlapped small neighborhood in the algorithm helps in exploring the search space because a slow diffusion of solutions through the population provides a kind of exploration, while exploitation takes place inside each neighborhood. See for more information on cellular Genetic Algorithms and related models.

Also, hybrid models are being proposed in which a two-level approach of parallelization is undertaken. In gen-

eral, the higher level for parallelization is a coarse-grained implementation and the basic island performs a cellular, a master-slave method or even another distributed one.

124.4 See also

- Cellular Evolutionary Algorithms
- Enrique Alba

124.5 References

- G. Luque, E. Alba, Parallel Genetic Algorithms. Theory and Real World Applications, Springer-Verlag, ISBN 978-3-642-22083-8, July 2011
- Alba E., Blum C., Isasi P., León C. Gómez J.A. (eds.), Optimization Techniques for Solving Complex Problems, Wiley, ISBN 978-0-470-29332-4, 2009
- E. Alba, B. Dorronsoro, Cellular Genetic Algorithms, Springer-Verlag, ISBN 978-0-387-77609-5, 2008
- N. Nedjah, E. Alba, L. de Macedo Mourelle, Parallel Evolutionary Computations, Springer-Verlag, ISBN 3-540-32837-8, 2006
- E. Alba, Parallel Metaheuristics: A New Class of Algorithms, Wiley, ISBN 0-471-67806-6, July 2005
- MALLBA
- JGDS
- DEME
- xxGA
- Paradiseo

124.6 External links

- THE Page on Parallel Metaheuristics
- The NEO group at the University of Málaga, Spain

Chapter 125

Particle swarm optimization

In computer science, **particle swarm optimization (PSO)** is a computational method that optimizes a problem by **iteratively** trying to improve a **candidate solution** with regard to a given measure of quality. PSO optimizes a problem by having a population of candidate solutions, here dubbed **particles**, and moving these particles around in the **search-space** according to simple **mathematical formulae** over the particle's **position** and **velocity**. Each particle's movement is influenced by its local best known position but, is also guided toward the best known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions.

PSO is originally attributed to Kennedy, Eberhart and Shi^{[1][2]} and was first intended for simulating social behaviour,^[3] as a stylized representation of the movement of organisms in a bird **flock** or fish **school**. The algorithm was simplified and it was observed to be performing optimization. The book by Kennedy and Eberhart^[4] describes many philosophical aspects of PSO and **swarm intelligence**. An extensive survey of PSO applications is made by Poli.^{[5][6]}

PSO is a **metaheuristic** as it makes few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, metaheuristics such as PSO do not guarantee an optimal solution is ever found. More specifically, PSO does not use the **gradient** of the problem being optimized, which means PSO does not require that the optimization problem be **differentiable** as is required by classic optimization methods such as **gradient descent** and **quasi-newton** methods. PSO can therefore also be used on optimization problems that are partially irregular, noisy, change over time, etc.

125.1 Algorithm

A basic variant of the PSO algorithm works by having a population (called a **swarm**) of **candidate solutions** (called **particles**). These particles are moved around in the search-space according to a few simple formulae. The movements of the particles are guided by their own best

known position in the search-space as well as the entire swarm's best known position. When improved positions are being discovered these will then come to guide the movements of the swarm. The process is repeated and by doing so it is hoped, but not guaranteed, that a satisfactory solution will eventually be discovered.

Formally, let $f: \mathbb{R}^n \rightarrow \mathbb{R}$ be the cost function which must be minimized. The function takes a candidate solution as argument in the form of a **vector of real numbers** and produces a real number as output which indicates the objective function value of the given candidate solution. The gradient of f is not known. The goal is to find a solution \mathbf{a} for which $f(\mathbf{a}) \leq f(\mathbf{b})$ for all \mathbf{b} in the search-space, which would mean \mathbf{a} is the global minimum. Maximization can be performed by considering the function $h = -f$ instead.

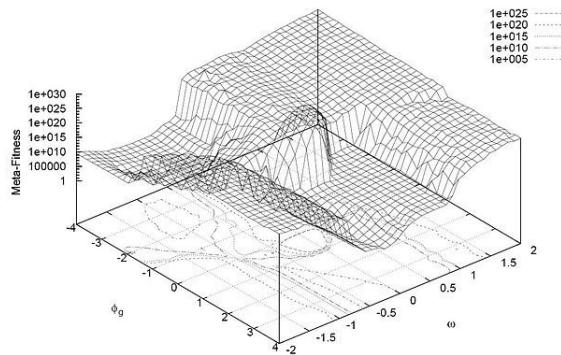
Let S be the number of particles in the swarm, each having a position $\mathbf{x}_i \in \mathbb{R}^n$ in the search-space and a velocity $\mathbf{v}_i \in \mathbb{R}^n$. Let \mathbf{p}_i be the best known position of particle i and let \mathbf{g} be the best known position of the entire swarm. A basic PSO algorithm is then:^[7]

- For each particle $i = 1, \dots, S$ do:
 - Initialize the particle's position with a **uniformly distributed** random vector: $\mathbf{x}_i \sim U(\mathbf{b}_{lo}, \mathbf{b}_{up})$, where \mathbf{b}_{lo} and \mathbf{b}_{up} are the lower and upper boundaries of the search-space.
 - Initialize the particle's best known position to its initial position: $\mathbf{p}_i \leftarrow \mathbf{x}_i$
 - If $(f(\mathbf{p}_i) < f(\mathbf{g}))$ update the swarm's best known position: $\mathbf{g} \leftarrow \mathbf{p}_i$
 - Initialize the particle's velocity: $\mathbf{v}_i \sim U(-\mathbf{b}_{up} - \mathbf{b}_{lo}, |\mathbf{b}_{up} - \mathbf{b}_{lo}|)$
- Until a termination criterion is met (e.g. number of iterations performed, or a solution with adequate objective function value is found), repeat:
 - For each particle $i = 1, \dots, S$ do:
 - Pick random numbers: $r_p, r_g \sim U(0,1)$
 - For each dimension $d = 1, \dots, n$ do:
 - Update the particle's velocity: $\mathbf{v}_{i,d} \leftarrow \omega \mathbf{v}_{i,d} + \varphi_p r_p (\mathbf{p}_{i,d} - \mathbf{x}_{i,d}) + \varphi_g r_g (\mathbf{g}_{d} - \mathbf{x}_{i,d})$

- Update the particle's position: $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i$
- If $(f(\mathbf{x}_i) < f(\mathbf{p}_i))$ do:
 - Update the particle's best known position: $\mathbf{p}_i \leftarrow \mathbf{x}_i$
 - If $(f(\mathbf{p}_i) < f(\mathbf{g}))$ update the swarm's best known position: $\mathbf{g} \leftarrow \mathbf{p}_i$
- Now \mathbf{g} holds the best found solution.

The parameters ω , φ_p , and φ_g are selected by the practitioner and control the behaviour and efficacy of the PSO method, see below.

125.2 Parameter selection



Performance landscape showing how a simple PSO variant performs in aggregate on several benchmark problems when varying two PSO parameters.

The choice of PSO parameters can have a large impact on optimization performance. Selecting PSO parameters that yield good performance has therefore been the subject of much research.^{[8][9][10][11][12][13][14][15]}

The PSO parameters can also be tuned by using another overlaying optimizer, a concept known as meta-optimization.^{[16][17][18]} Parameters have also been tuned for various optimization scenarios.^[19]

125.3 Neighborhoods and Topologies

The basic PSO is easily trapped into a local minimum. This premature convergence can be avoided by not using the entire swarm's best known position \mathbf{g} but just the best known position \mathbf{l} of a sub-swarm "around" the particle that is moved. Such a sub-swarm can be a geometrical one - for example "the m nearest particles" - or, more often, a social one, i.e. a set of particles that is not depending on any distance. In such a case, the PSO variant is said to be local best (vs global best for the basic PSO).

If we suppose there is an information link between each particle and its neighbours, the set of these links builds a graph, a communication network, that is called the topology of the PSO variant. A commonly used social topology is the ring, in which each particle has just two neighbours, but there are many others.^[20] The topology is not necessarily fixed, and can be adaptive (SPSO,^[21] stochastic star,^[22] TRIBES,^[23] Cyber Swarm,^[24] C-PSO^[25]).

125.4 Inner workings

There are several schools of thought as to why and how the PSO algorithm can perform optimization.

A common belief amongst researchers is that the swarm behaviour varies between exploratory behaviour, that is, searching a broader region of the search-space, and exploitative behaviour, that is, a locally oriented search so as to get closer to a (possibly local) optimum. This school of thought has been prevalent since the inception of PSO.^{[2][3][8][12]} This school of thought contends that the PSO algorithm and its parameters must be chosen so as to properly balance between exploration and exploitation to avoid premature convergence to a local optimum yet still ensure a good rate of convergence to the optimum. This belief is the precursor of many PSO variants, see below.

Another school of thought is that the behaviour of a PSO swarm is not well understood in terms of how it affects actual optimization performance, especially for higher-dimensional search-spaces and optimization problems that may be discontinuous, noisy, and time-varying. This school of thought merely tries to find PSO algorithms and parameters that cause good performance regardless of how the swarm behaviour can be interpreted in relation to e.g. exploration and exploitation. Such studies have led to the simplification of the PSO algorithm, see below.

125.4.1 Convergence

In relation to PSO the word *convergence* typically refers to two different definitions:

- Convergence of the sequence of solutions (aka, stability analysis, *converging*) in which all particles have converged to a point in the search-space, which may or may not be the optimum,
- Convergence to a local optimum where all personal bests \mathbf{p} or, alternatively, the swarm's best known position \mathbf{g} , approaches a local optimum of the problem, regardless of how the swarm behaves.

Convergence of the sequence of solutions has been investigated for PSO.^{[11][12][13]} These analyses have resulted in

guidelines for selecting PSO parameters that are believed to cause convergence to a point and prevent divergence of the swarm's particles (particles do not move unboundedly and will converge to somewhere). However, the analyses were criticized by Pedersen^[18] for being oversimplified as they assume the swarm has only one particle, that it does not use stochastic variables and that the points of attraction, that is, the particle's best known position \mathbf{p} and the swarm's best known position \mathbf{g} , remain constant throughout the optimization process. However, it was shown^[26] that these simplifications do not affect the boundaries found by these studies for parameter where the swarm is convergent.

Convergence to a local optimum has been analyzed for PSO in^[27] and^[28]. It has been proven that PSO need some modification to guarantee to find a local optimum.

This means that determining convergence capabilities of different PSO algorithms and parameters therefore still depends on empirical results. One attempt at addressing this issue is the development of an “orthogonal learning” strategy for an improved use of the information already existing in the relationship between \mathbf{p} and \mathbf{g} , so as to form a leading converging exemplar and to be effective with any PSO topology. The aims are to improve the performance of PSO overall, including faster global convergence, higher solution quality, and stronger robustness.^[29] However, such studies do not provide theoretical evidence to actually prove their claims.

125.4.2 Biases

As the basic PSO works dimension by dimension, the solution point is easier found when it lies on an axis of the search space, on a diagonal, and even easier if it is right on the centre.^{[30][31]}

One approach is to modify the algorithm so that it is not any more sensitive to the system of coordinates.^{[32][33][34][35]} Note that some of these methods have a higher computational complexity (are in $O(n^2)$ where n is the number of dimensions) that make the algorithm very slow for large scale optimization.^[28]

The only currently existing PSO variant that is not sensitive to the rotation of the coordinates while is locally convergent has been proposed at 2014.^[28] The method has shown a very good performance on many benchmark problems while its rotation invariance and local convergence have been mathematically proven.

125.5 Variants

Numerous variants of even a basic PSO algorithm are possible. For example, there are different ways to initialize the particles and velocities (e.g. start with zero velocities instead), how to dampen the velocity, only update \mathbf{p}_i

and \mathbf{g} after the entire swarm has been updated, etc. Some of these choices and their possible performance impact have been discussed in the literature.^[10]

New and more sophisticated PSO variants are also continually being introduced in an attempt to improve optimization performance. There are certain trends in that research; one is to make a hybrid optimization method using PSO combined with other optimizers,^{[36][37][38]} e.g., the incorporation of an effective learning method.^[29] Another research trend is to try and alleviate premature convergence (that is, optimization stagnation), e.g. by reversing or perturbing the movement of the PSO particles,^{[15][39][40][41]} another approach to deal with premature convergence is the use of multiple swarms^[42] (multi-swarm optimization). The multi-swarm approach can also be used to implement multi-objective optimization.^[43] Finally, there are developments in adapting the behavioural parameters of PSO during optimization.^[44]

125.5.1 Simplifications

Another school of thought is that PSO should be simplified as much as possible without impairing its performance; a general concept often referred to as Occam's razor. Simplifying PSO was originally suggested by Kennedy^[3] and has been studied more extensively,^{[14][17][18][45]} where it appeared that optimization performance was improved, and the parameters were easier to tune and they performed more consistently across different optimization problems.

Another argument in favour of simplifying PSO is that metaheuristics can only have their efficacy demonstrated empirically by doing computational experiments on a finite number of optimization problems. This means a metaheuristic such as PSO cannot be proven correct and this increases the risk of making errors in its description and implementation. A good example of this^[46] presented a promising variant of a genetic algorithm (another popular metaheuristic) but it was later found to be defective as it was strongly biased in its optimization search towards similar values for different dimensions in the search space, which happened to be the optimum of the benchmark problems considered. This bias was because of a programming error, and has now been fixed.^[47]

Initialization of velocities may require extra inputs. A simpler variant is the accelerated particle swarm optimization (APSO),^[48] which does not need to use velocity at all and can speed up the convergence in many applications. A simple demo code of APSO is available.^[49]

125.5.2 Multi-objective optimization

PSO has also been applied to multi-objective problems,^{[50][51]} in which the objective function comparison

takes pareto dominance into account when moving the PSO particles and non-dominated solutions are stored so as to approximate the pareto front.

125.5.3 Binary, Discrete, and Combinatorial PSO

As the PSO equations given above work on real numbers, a commonly used method to solve discrete problems is to map the discrete search space to a continuous domain, to apply a classical PSO, and then to demap the result. Such a mapping can be very simple (for example by just using rounded values) or more sophisticated.^[52]

However, it can be noted that the equations of movement make use of operators that perform four actions:

- computing the difference of two positions. The result is a velocity (more precisely a displacement)
- multiplying a velocity by a numerical coefficient
- adding two velocities
- applying a velocity to a position

Usually a position and a velocity are represented by n real numbers, and these operators are simply $-$, $*$, $+$, and again $+$. But all these mathematical objects can be defined in a completely different way, in order to cope with binary problems (or more generally discrete ones), or even combinatorial ones.^{[53][54][55][56]} One approach is to redefine the operators based on sets.^[57]

125.6 See also

- Swarm intelligence
- Multi-swarm optimization
- Bees algorithm / Artificial Bee Colony Algorithm
- Particle filter

125.7 References

- [1] Kennedy, J.; Eberhart, R. (1995). "Particle Swarm Optimization". *Proceedings of IEEE International Conference on Neural Networks* **IV**. pp. 1942–1948. doi:10.1109/ICNN.1995.488968.
- [2] Shi, Y.; Eberhart, R.C. (1998). "A modified particle swarm optimizer". *Proceedings of IEEE International Conference on Evolutionary Computation*. pp. 69–73.
- [3] Kennedy, J. (1997). "The particle swarm: social adaptation of knowledge". *Proceedings of IEEE International Conference on Evolutionary Computation*. pp. 303–308.
- [4] Kennedy, J.; Eberhart, R.C. (2001). *Swarm Intelligence*. Morgan Kaufmann. ISBN 1-55860-595-9.
- [5] Poli, R. (2007). "An analysis of publications on particle swarm optimisation applications". *Technical Report CSM-469* (Department of Computer Science, University of Essex, UK).
- [6] Poli, R. (2008). "Analysis of the publications on the applications of particle swarm optimisation". *Journal of Artificial Evolution and Applications* **2008**: 1–10. doi:10.1155/2008/685175.
- [7] Clerc, M. (2012). "Standard Particle Swarm Optimisation". *HAL open access archive*.
- [8] Shi, Y.; Eberhart, R.C. (1998). "Parameter selection in particle swarm optimization". *Proceedings of Evolutionary Programming VII (EP98)*. pp. 591–600.
- [9] Eberhart, R.C.; Shi, Y. (2000). "Comparing inertia weights and constriction factors in particle swarm optimization". *Proceedings of the Congress on Evolutionary Computation* **1**. pp. 84–88.
- [10] Carlisle, A.; Dozier, G. (2001). "An Off-The-Shelf PSO". *Proceedings of the Particle Swarm Optimization Workshop*. pp. 1–6.
- [11] van den Bergh, F. (2001). *An Analysis of Particle Swarm Optimizers* (PhD thesis). University of Pretoria, Faculty of Natural and Agricultural Science.
- [12] Clerc, M.; Kennedy, J. (2002). "The particle swarm - explosion, stability, and convergence in a multidimensional complex space". *IEEE Transactions on Evolutionary Computation* **6** (1): 58–73. doi:10.1109/4235.985692.
- [13] Trelea, I.C. (2003). "The Particle Swarm Optimization Algorithm: convergence analysis and parameter selection". *Information Processing Letters* **85** (6): 317–325. doi:10.1016/S0020-0190(02)00447-7.
- [14] Bratton, D.; Blackwell, T. (2008). "A Simplified Recombinant PSO". *Journal of Artificial Evolution and Applications*.
- [15] Evers, G. (2009). *An Automatic Regrouping Mechanism to Deal with Stagnation in Particle Swarm Optimization* (Master's thesis). The University of Texas - Pan American, Department of Electrical Engineering.
- [16] Meissner, M.; Schmuker, M.; Schneider, G. (2006). "Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural network training". *BMC Bioinformatics* **7** (1): 125. doi:10.1186/1471-2105-7-125. PMC 1464136. PMID 16529661.
- [17] Pedersen, M.E.H. (2010). *Tuning & Simplifying Heuristic Optimization* (PhD thesis). University of Southampton, School of Engineering Sciences, Computational Engineering and Design Group.
- [18] Pedersen, M.E.H.; Chipperfield, A.J. (2010). "Simplifying particle swarm optimization". *Applied Soft Computing* **10** (2): 618–628. doi:10.1016/j.asoc.2009.08.029.

- [19] Pedersen, M.E.H. (2010). “Good parameters for particle swarm optimization”. *Technical Report HL1001* (Hvass Laboratories).
- [20] Mendes, R. (2004). Population Topologies and Their Influence in Particle Swarm Performance (PhD thesis). Universidade do Minho.
- [21] SPSO, Particle Swarm Central
- [22] Miranda, V., Keko, H. and Duque, Á. J. (2008). Stochastic Star Communication Topology in Evolutionary Particle Swarms (EPSO). *International Journal of Computational Intelligence Research (IJCIR)*, Volume 4, Number 2, pp. 105-116
- [23] Clerc, M. (2006). Particle Swarm Optimization. ISTE (International Scientific and Technical Encyclopedia), 2006
- [24] Yin, P., Glover, F., Laguna, M., & Zhu, J. (2011). A Complementary Cyber Swarm Algorithm. *International Journal of Swarm Intelligence Research (IJSIR)*, 2(2), 22-41
- [25] Elshamy, W.; Rashad, H.; Bahgat, A. (2007). “Clubs-based Particle Swarm Optimization”. *IEEE Swarm Intelligence Symposium 2007 (SIS2007)*. Honolulu, HI. pp. 289–296.
- [26] Cleghorn, Christopher W (2014). “Particle Swarm Convergence: Standardized Analysis and Topological Influence”. *Swarm Intelligence Conference*.
- [27] Van den Bergh, F. “A convergence proof for the particle swarm optimiser”. *Fundamenta Informaticae*.
- [28] Bonyadi, Mohammad reza.; Michalewicz, Z. (2014). “A locally convergent rotationally invariant particle swarm optimization algorithm”. *Swarm intelligence* **8** (3): 159–198. doi:10.1007/s11721-014-0095-1.
- [29] Zhan, Z-H.; Zhang, J.; Li, Y; Shi, Y-H. (2011). “Orthogonal Learning Particle Swarm Optimization”. *IEEE Transactions on Evolutionary Computation* **15** (6): 832–847. doi:10.1109/TEVC.2010.2052054.
- [30] Monson, C. K. & Seppi, K. D. (2005). Exposing Origin-Seeking Bias in PSO GECCO'05, pp. 241-248
- [31] Spears, W. M., Green, D. T. & Spears, D. F. (2010). Biases in Particle Swarm Optimization. *International Journal of Swarm Intelligence Research*, Vol. 1(2), pp. 34-57
- [32] Wilke, D. N., Kok, S. & Groenwold, A. A. (2007). Comparison of linear and classical velocity update rules in particle swarm optimization: notes on scale and frame invariance. *International Journal for Numerical Methods in Engineering*, John Wiley & Sons, Ltd., 70, pp. 985-1008
- [33] SPSO 2011, Particle Swarm Central
- [34] Bonyadi, Mohammad reza; Michalewicz, Z. (2014). “SPSO 2011 analysis of stability; local convergence; and rotation sensitivity.”. *GECCO2014 (the best paper award in the track ACSI)*: 9–16.
- [35] Bonyadi, Mohammad reza.; Michalewicz, Z. (2014). “An analysis of the velocity updating rule of the particle swarm optimization algorithm”. *Journal of Heuristics* **20** (4): 417–452. doi:10.1007/s10732-014-9245-2.
- [36] Lovbjerg, M.; Krink, T. (2002). “The LifeCycle Model: combining particle swarm optimisation, genetic algorithms and hillclimbers”. *Proceedings of Parallel Problem Solving from Nature VII (PPSN)*. pp. 621–630.
- [37] Niknam, T.; Amiri, B. (2010). “An efficient hybrid approach based on PSO, ACO and k-means for cluster analysis”. *Applied Soft Computing* **10** (1): 183–197. doi:10.1016/j.asoc.2009.07.001.
- [38] Zhang, Wen-Jun; Xie, Xiao-Feng (2003). DEPSO: hybrid particle swarm with differential evolution operator. *IEEE International Conference on Systems, Man, and Cybernetics (SMCC)*, Washington, DC, USA: 3816-3821.
- [39] Lovbjerg, M.; Krink, T. (2002). “Extending Particle Swarm Optimisers with Self-Organized Criticality”. *Proceedings of the Fourth Congress on Evolutionary Computation (CEC)* **2**. pp. 1588–1593.
- [40] Xinchao, Z. (2010). “A perturbed particle swarm algorithm for numerical optimization”. *Applied Soft Computing* **10** (1): 119–124. doi:10.1016/j.asoc.2009.06.010.
- [41] Xie, Xiao-Feng; Zhang, Wen-Jun; Yang, Zhi-Lian (2002). A dissipative particle swarm optimization. *Congress on Evolutionary Computation (CEC)*, Honolulu, HI, USA: 1456-1461.
- [42] Cheung, N. J., Ding, X.-M., & Shen, H.-B. (2013). OptiFel: A Convergent Heterogeneous Particle Swarm Optimization Algorithm for Takagi-Sugeno Fuzzy Modeling, *IEEE Transactions on Fuzzy Systems*, doi:10.1109/TFUZZ.2013.2278972
- [43] Nobile, M.; Besozzi, D.; Cazzaniga, P.; Mauri, G.; Pescini, D. (2012). “A GPU-Based Multi-Swarm PSO Method for Parameter Estimation in Stochastic Biological Systems Exploiting Discrete-Time Target Series”. *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. Lecture Notes in Computer Science*. **7264**. pp. 74–85.
- [44] Zhan, Z-H.; Zhang, J.; Li, Y; Chung, H.S-H. (2009). “Adaptive Particle Swarm Optimization”. *IEEE Transactions on Systems, Man, and Cybernetics* **39** (6): 1362–1381. doi:10.1109/TSMCB.2009.2015956.
- [45] Yang, X.S. (2008). *Nature-Inspired Metaheuristic Algorithms*. Luniver Press. ISBN 978-1-905986-10-1.
- [46] Tu, Z.; Lu, Y. (2004). “A robust stochastic genetic algorithm (StGA) for global numerical optimization”. *IEEE Transactions on Evolutionary Computation* **8** (5): 456–470. doi:10.1109/TEVC.2004.831258.
- [47] Tu, Z.; Lu, Y. (2008). “Corrections to “A Robust Stochastic Genetic Algorithm (StGA) for Global Numerical Optimization”. *IEEE Transactions on Evolutionary Computation* **12** (6): 781–781. doi:10.1109/TEVC.2008.926734.

- [48] X. S. Yang, S. Deb and S. Fong, Accelerated particle swarm optimization and support vector machine for business optimization and applications, NDT 2011, Springer CCIS 136, pp. 53-66 (2011).
 - [49] <http://www.mathworks.com/matlabcentral/fileexchange/?term=APSO>
 - [50] Parsopoulos, K.; Vrahatis, M. (2002). "Particle swarm optimization method in multiobjective problems". *Proceedings of the ACM Symposium on Applied Computing (SAC)*. pp. 603–607.
 - [51] Coello Coello, C.; Salazar Lechuga, M. (2002). "MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization". *Congress on Evolutionary Computation (CEC'2002)*. pp. 1051–1056.
 - [52] Roy, R., Dehuri, S., & Cho, S. B. (2012). A Novel Particle Swarm Optimization Algorithm for Multi-Objective Combinatorial Optimization Problem. 'International Journal of Applied Metaheuristic Computing (IJAMC)', 2(4), 41-57
 - [53] Kennedy, J. & Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm, Conference on Systems, Man, and Cybernetics, Piscataway, NJ: IEEE Service Center, pp. 4104-4109
 - [54] Clerc, M. (2004). Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem, New Optimization Techniques in Engineering, Springer, pp. 219-239
 - [55] Clerc, M. (2005). Binary Particle Swarm Optimisers: toolbox, derivations, and mathematical insights, Open Archive HAL
 - [56] Jarboui, B., Damak, N., Siarry, P., and Rebai, A.R. (2008). A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. In Proceedings of Applied Mathematics and Computation, pp. 299-308.
 - [57] Chen, Wei-neng; Zhang, Jun (2010). "A novel set-based particle swarm optimization method for discrete optimization problem". *IEEE Transactions on Evolutionary Computation* **14** (2): 278–300. doi:10.1109/tevc.2009.2030331.
- Particle Swarm Optimization (see and listen to Lecture 27)

125.8 External links

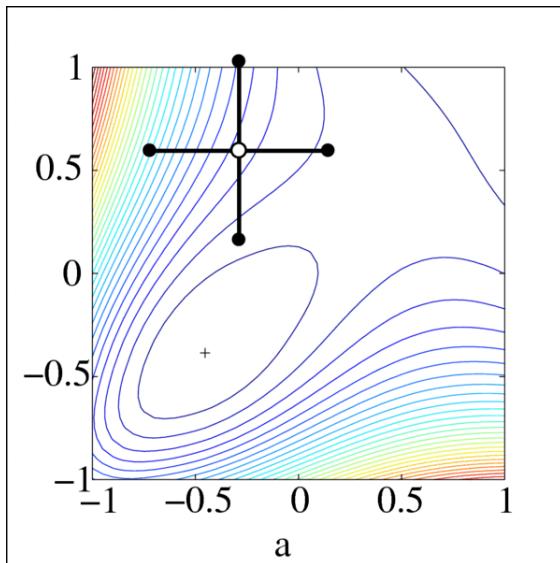
- Particle Swarm Central is a repository for information on PSO. Several source codes are freely available.
- A brief video of particle swarms optimizing three benchmark functions.
- Applications of PSO.
- Automatic Calibration of a Rainfall-Runoff Model Using a Fast and Elitist Multi-objective Particle Swarm Algorithm

Chapter 126

Pattern search (optimization)

For other uses, see [Pattern search](#).

Pattern search (PS) is a family of numerical



Example of convergence of a direct search method on Broyden function

optimization methods that do not require the gradient of the problem to be optimized. Hence PS can be used on functions that are not [continuous](#) or [differentiable](#). Such optimization methods are also known as direct-search, derivative-free, or black-box methods.

The name, pattern search, was coined by Hooke and Jeeves.^[1] An early and simple PS variant is attributed to Fermi and Metropolis when they worked at the Los Alamos National Laboratory as described by Davidon^[2] who summarized the algorithm as follows:

They varied one theoretical parameter at a time by steps of the same magnitude, and when no such increase or decrease in any one parameter further improved the fit to the experimental data, they halved the step size and repeated the process until the steps were deemed sufficiently small.

126.1 Convergence

A convergent pattern-search method was proposed by Yu, who proved that it converged using the theory of positive bases.^[3] Later, Torczon, Lagarias, and coauthors,^{[4][5]} used positive-basis techniques to prove the convergence of another pattern search method, on a specific class of functions. Outside of such classes, pattern search is a [heuristic](#) that can provide useful approximate solutions for some problems, but can fail on others. Outside of such classes, pattern search is not an [iterative method](#) that converges to a solution; indeed, pattern search methods can converge to non-stationary points on some relatively tame problems.^{[6][7]}

126.2 See also

- [Golden section search](#) conceptually resembles PS in its narrowing of the search-range, only for single-dimensional search-spaces.
- [Nelder–Mead method](#) aka. the simplex method conceptually resembles PS in its narrowing of the search-range for multi-dimensional search-spaces but does so by maintaining $n+1$ points for n -dimensional search-spaces whereas PS methods computes $2n+1$ points (the central point and 2 points in each dimension).
- [Luus–Jaakola](#) samples from a [uniform distribution](#) surrounding the current position and uses a simple formula for exponentially decreasing the sampling range.
- [Random search](#) is a related family of optimization methods which sample from a [hypersphere](#) surrounding the current position.
- [Random optimization](#) is a related family of optimization methods which sample from a [normal distribution](#) surrounding the current position.

126.3 References

- [1] Hooke, R.; Jeeves, T.A. (1961). ""Direct search" solution of numerical and statistical problems". *Journal of the Association for Computing Machinery (ACM)* **8** (2): 212–229. doi:10.1145/321062.321069.
- [2] Davidon, W.C. (1991). "Variable metric method for minimization". *SIAM Journal on Optimization* **1** (1): 1–17. doi:10.1137/0801001.
- [3]
 - Yu, Wen Ci. 1979. "Positive basis and a class of direct search techniques". *Scientia Sinica [Zhongguo Kexue]*: 53—68.
 - Yu, Wen Ci. 1979. "The convergent property of the simplex evolutionary technique". *Scientia Sinica [Zhongguo Kexue]*: 69–77.
- [4] Torczon, V.J. (1997). "On the convergence of pattern search algorithms". *SIAM Journal on Optimization* **7** (1): 1–25. doi:10.1137/S1052623493250780.
- [5] Dolan, E.D.; Lewis, R.M.; Torczon, V.J. (2003). "On the local convergence of pattern search". *SIAM Journal on Optimization* **14** (2): 567–583. doi:10.1137/S1052623400374495.
- [6]
 - Powell, Michael J. D. 1973. "On Search Directions for Minimization Algorithms." *Mathematical Programming* 4: 193—201.
- [7]
 - McKinnon, K.I.M. (1999). "Convergence of the Nelder–Mead simplex method to a non-stationary point". *SIAM J Optimization* **9**: 148–158. doi:10.1137/S1052623496303482. (algorithm summary online).

Chapter 127

pSeven

pSeven is a design space exploration software platform developed by DATADVANCE LLC, extending design, simulation and analysis capabilities and assisting in smarter and faster design decisions. It provides a seamless integration with third party **CAD** and **CAE** software tools, powerful multi-objective and robust optimization algorithms, data mining and uncertainty management tools.^[1] pSeven comes under the notion of **PIDO** (Process Integration and Design Optimization) software. Design space exploration functionality is based on the mathematical algorithms of Macros software library,^[2] also developed by Datadvance. pSeven is used for the needs of predicting system behavior, analyzing the existing experimental data, estimating model uncertainties or performing multidisciplinary design optimization. pSeven implements Datadvance strategy to:

- Enable engineers to easily use mathematical algorithms thanks to “Smart Selection” technique, which automatically and adaptively selects the most suitable algorithm for a given problem from a pool of available methods;
- Address complex industrial challenges: architecture trade-off, optimization using expensive simulations.

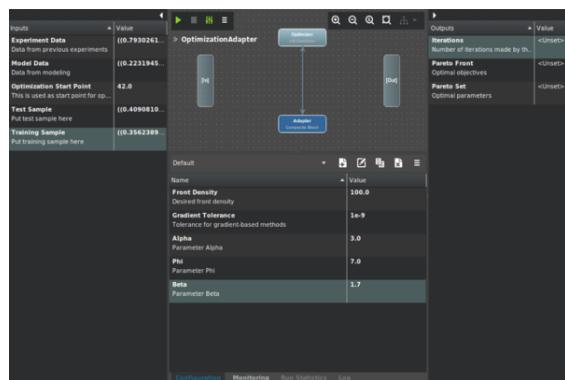
127.1 History

The foundation for Macros library as pSeven’s background was laid in 2003, when the researchers from the Institute for Information Transmission Problems of the Russian Academy of Sciences^[3] started collaborating with Airbus Group to perform R&D in the domains of simulation and data analysis. The first version of Macros library was created in association with EADS Innovation Works in 2009.^[4] Since 2012,^[5] pSeven software platform for simulation automation, data analysis and optimization is developed and marketed by Datadvance, incorporating Macros algorithmic core.

127.2 Functionality

pSeven’s functionality includes three major blocks: **Process integration**, **Design Space Exploration**, **Vizualisation and Postprocessing**.

127.2.1 Process integration



Workflow in pSeven

Process integration capabilities are used to *capture the design process by automating single simulation, trade-off studies and design space exploration*. For that, pSeven provides tools to build and automatically run the workflow, to configure and share workflows with other design team members, to distribute computation over different computing resources, including HPC. Main process integration tools of pSeven:

- Graphical user interface and command-line interface for advanced users
- Comprehensive library of workflow building blocks
- CAD/CAE integration adapters (**SolidWorks**, **CATIA**, **NX**, **PTC Creo**, **KOMPAS-3D**), CAE solvers and other engineering tools (**ANSYS Mechanical**, **ANSYS CFD**, **FloEFD**, **CST Microwave Studio**, **ADAMS**, **Simulink**, **MATLAB**, **Scilab**, **Abaqus**, **Unified FEA**, **Nastran**, **LS-DYNA**, **STAR-CCM+**, **OpenFOAM**, etc.)

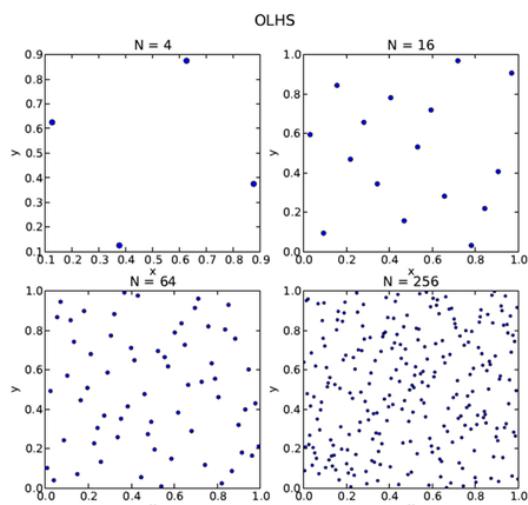
- Workflow as a Ready Tool mechanism
- High Performance Computing (HPC) capabilities (supported batch systems: SLURM, TORQUE, LSF)

127.2.2 Design Space Exploration

Design Space Exploration toolset in pSeven offers a variety of methods for:

- Parametric Studies
- Design of Experiments
- Sensitivity and Dependency Analysis
- Surrogate modeling
- Single and Multi-Objective Optimization
- Uncertainty Management
- Robust and Reliability Based Design Optimization

Design of Experiments



Design of Experiments allows controlling the process of surrogate modeling via adaptive sampling plan, which benefits quality of approximation.

Design of Experiments includes the following techniques:
Space Filling

- Batch techniques (Random, Full-Factorial, Latin hypercube sampling, Optimal LHS)
- Sequential techniques (Random, Halton, Sobol, Faure sequences)

Optimal Designs for RSM

- Composite, D-optimal, IV-optimal, Box Behnken

Adaptive DoE with Uniform, Maximum Variance and Integrated Mean Squared Errors Gain - Maximum Variance criteria.

Design of Experiments allows controlling the process of surrogate modeling via adaptive sampling plan, which benefits quality of approximation. As a result, it *ensures time and resource saving on experiments and smarter decision-making based on the detailed knowledge of the design space.*

Sensitivity and Dependency Analysis

Sensitivity and Dependency Analysis is used to *filter non-informative design parameters in the study, ranking the informative ones with respect to their influence on the given response function* and selecting parameters that provide the best approximation. It is applied to better understand the variables affecting the design process. It comprises the following steps:

- Feature selection
- Feature extraction
- Sensitivity analysis

The techniques below are included in pSeven to perform Sensitivity and Dependency Analysis: Popular correlation coefficients:

- Pearson,
- Spearman,
- Partial Pearson correlation.

Screening indices:

- Can be computed even with very small budget
- In addition to providing ranking indices allow to check linearity and monotonicity of objective with respect to each input

Sobol indices:

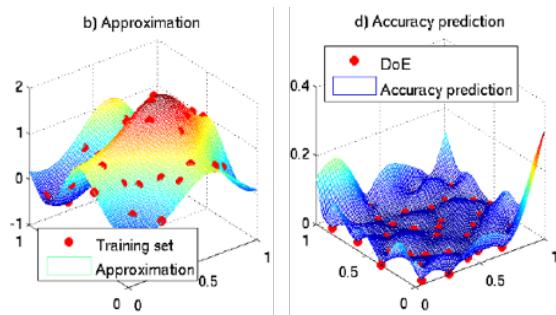
- Total indices – measure total effects (tell which portion of objective would be lost if one would fix considered input)
- Main indices – measure first order effects (tell which portion of objective is explained by considered input if all other inputs are fixed).
- Interaction indices – measure higher order effects (difference between Total and Main).

Taguchi scores:

- Robustness of objective towards noise in each input
- Statistical significance check

Surrogate modeling

Surrogate modeling (or **Approximation**) capabilities in pSeven incorporate several proprietary surrogate modeling techniques, including methods for ordered and structured data, *allowing to understand behavior of user's system with minimal costs, replace expensive computations by surrogate models (metamodels) and make smarter decisions based on detailed knowledge of the design space* [6]



Accuracy assessment of constructed models

- Classic methods (Splines^[7] Linear Regression, Kriging, etc)
- Proprietary methods (Higher Dimensional Approximation, SGP - Sparse Gaussian Processes, Tensor Approximation and incomplete Tensor Approximation, Tensor Gaussian Processes etc.)

These methods provide users with the following features:

- Accuracy assessment of constructed models
- Full control of the model construction time
- Smoothing ^[8]
- Surrogate model export (**C**, MATLAB, Octave)
- Construction of variable fidelity models

“Smart Selection” automatically picks the best suitable technique based on a few user-defined problem settings. *Surrogate modeling allows user getting value from experimental data and perform faster and cheaper system behavior analysis.*

Data Fusion

Data fusion module of pSeven’s functionality is used to better handle and increase efficiency of surrogate modeling. It allows constructing surrogate models using data of different levels of fidelity, handle samples of varied sizes and perform accuracy evaluation. It allows managing a surrogate model construction time, estimate uncertainty for predictions obtained with a Data Fusion-based surrogate model and estimate the quality of models constructed. Data Fusion technology works in two modes:

- sample-based (tool takes high fidelity sample and a low fidelity sample as inputs. These samples consist of points and corresponding values of a considered function)
- blackbox-based (tool takes high fidelity sample and a low fidelity blackbox as inputs. In this mode low fidelity function blackbox provides low fidelity function values at any feasible point from a specified design space)

Data Fusion techniques available in pSeven:

- HFA (High Fidelity Approximation) — uses only high fidelity data,
- DA (Difference Approximation) — approximates difference between low and high fidelity data
- VFGP (Variable Fidelity Gaussian Processes) — builds models using Gaussian processes regression ideas
- SVFGP (Sparse Variable Fidelity Gaussian Processes) is designed to handle large samples with Gaussian processes regression-based technique.

Optimization

Optimization algorithms implemented in pSeven allow solving single and multiobjective constrained optimization problems as well as robust and reliability based design optimization problems. Users can solve both engineering optimization problems with cheap to evaluate semi-analytical models and the problems with expensive (in terms of CPU time) objective functions and constraints. ^{[9][10]} *Smart Selection technique automatically and adaptively selects the most suitable optimization algorithm for a given optimization problem from a pool of optimization methods and algorithms in pSeven.* Main classes of optimization problems which are supported:

Optimization methods available in pSeven:

- Noisy data handling techniques
- Mathematical programming: Mixed-Integer Linear Programming (MILP)

- Mathematical programming: Quadratic programming (QP)
- Mathematical programming: Unconstrained Nonlinear programming (UNLP)
- Mathematical programming: Constrained Non-Linear programming (NLP)
- Constraint Satisfaction Problems (CSP)
- Multi-Objective Optimization [11]
- Surrogate-Based Optimization (SBO):^{[12][13][14]} constrained single-objective problems
- Surrogate-Based Optimization (SBO): constrained multi-objective problems
- Global Search Methods (excluding SBO)
- Robust Optimization

Uncertainty management

Uncertainty Management capabilities in pSeven^[15] are based on OpenTurns library. They are used to *improve the quality of products designed, manage potential risks at the design, manufacturing and operating stages and to guarantee product reliability*. A range of uncertainty management algorithms allows to perform: Quantification of uncertainty sources (based on experimental data or expert knowledge)

- Probabilistic model (non-parametric, e.g. Kernel Smoothing; parametric, e.g. Normal, Beta)
- Auto-selection of distribution type for parameters sample
- Goodness-of-fit tests for sample-based probability models (e.g. Kolmogorov-Smirnov test)
- Dependencies of input parameters (e.g. Spearman correlation coefficient)

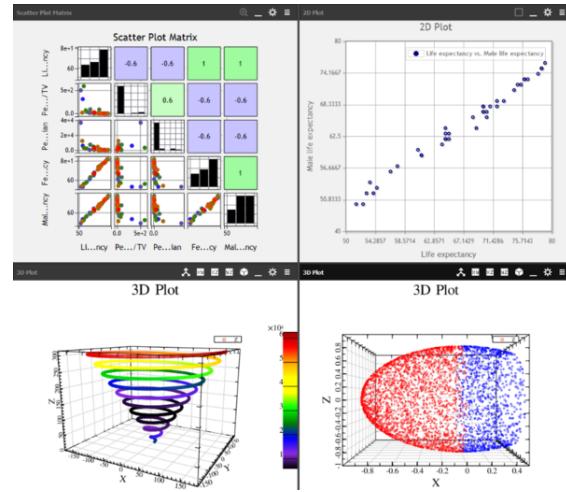
Uncertainty propagation using Monte-Carlo simulation

- Central dispersion, output distribution analysis (Monte Carlo)
- Failure probability (Reliability analysis): approximation (FORM), simulation (Monte Carlo, LHS, Directional sampling)

Reliability analysis

- Monte-Carlo approach
- Surrogate model approach

Reliability-based design optimization



Results visualization tools available in pSeven

127.2.3 Visualization and postprocessing

Visualization and postprocessing tools in pSeven are used to analyze the results of design space exploration procedures and include:

- Plot configuration dialogs
- Structured Parallel coordinates
- Scatter plot Matrices
- 2D- and 3D-plots
- Table view and drag-and drop options

127.3 Application areas and customers

pSeven's application areas are different industries such as aerospace,^{[16][17]} automotive, energy, electronics and electrical appliances, banking, insurance, biotechnology and others. Application examples:

- Multidisciplinary and multi-objective optimization of an aircraft family^[18]
- Sizing of composite structures in order to reduce their mass subject to various mechanical and manufacturing constraints
- Construction of quick and accurate behavioral models (surrogate models) in order to enable efficient and secure exchange of models across Extended Enterprise
- Optimization of the gas path of the steam turbine in order to improve overall turbine efficiency
- Optimization of layered composite armor in order to reduce its weight^[19]

127.4 Modules and Packs

Three pSeven configurations are available, each of them with different functionality modules:

- **pSeven Basic** (Workflow construction, execution and post-processing capabilities)
- **pSeven MDO** (Workflow construction, execution and post-processing capabilities - Macros Technology tools for data analysis and optimization)
- **pSeven Ultimate** (Workflow construction, execution and post-processing capabilities, Macros Technology tools for data analysis and optimization, CAD/CAE software integration tools and HPC integration toolset, Uncertainty management capabilities).
- **Macros algorithmic library** can also be purchased as a standalone product Macros for Python, which includes Generic Tools for Optimization, Approximation, Data Fusion, Dimension Reduction, Design of Experiments and Sensitivity and Dependency Analysis.

127.5 References

- [1] Trade Delegation of the Russian Federation in the United Kingdom
- [2] <https://www.datadvance.net/product/technology/>
- [3] Institute of Information Transmission Problems
- [4] Interview of Sergey Morozov, CTO of Datadvance, "The Moscow Times", issue № 3 (45) 2014
- [5] OraResearch, Design Space Exploration Industry Timeline
- [6] Burnaev E., Prikhodko P., Struzik A., "Surrogate models for helicopter loads problems", Proceedings of 5th European Conference for Aerospace Science"
- [7] Golubev Y., Krymova E., "Splines and Stationary Gaussian Processes", Proceedings of the conference "Information Technologies and Systems", 2012
- [8] Bernstein A., Belyaev M., Burnaev E., Yanovich, Yu., Smoothing of Surrogate Models, Proceedings of the conf. "Information Technologies and Systems", 2011. P. 423-432.
- [9] F. Gubarev, V. Kunin, A. Pospelov, "Lay-up Optimization of Laminated Composites: Mixed Approach with Exact Feasibility Bounds on Lamination Parameters"
- [10] Dmitry Khominich, Fedor Gubarev, Alexis Pospelov, "Shape Optimization of Rotating Disks", 20th Conference of the International Federation of Operational Research Societies, 2014
- [11] Alexis Pospelov, Fedor Gubarev, "Nonlinear Multi-Objective Constrained Optimization", 20th Conference of the International Federation of Operational Research Societies, 2014
- [12] Grihon S., Alestra S., Burnaev E., Prikhodko P., "Surrogate Modeling of Buckling Analysis in Support of Composite Structure Optimization", 1st International Conference on Composites Dynamics
- [13] Alexis I. Pospelov, Fedor V. Gubarev, Alexey M. Nazarenko, "Adaptive Surrogate-Based Multi-Criteria Optimization", 11th World Congress on Computational Mechanics (WCCM XI)
- [14] F.V. Gubarev, A.I. Pospelov, "Towards Large-Scale Surrogate-Based Optimization", 4th International Conference of Engineering Optimization, 2014
- [15] DataAdvance Ships pSeven v4.0 for Data Analysis, Optimization, TenLinks CAD, CAM and CAE news
- [16] Airbus achieves multi-objective optimization of its aircraft families with Datadvice's "Macros" software
- [17] Skolkovo resident Datadvice tows Airbus through structural testing on new A350
- [18] Alestra S., Brand C., Druot T., Morozov S., "Multi-objective Optimization of Aircrafts [sic] Family at Conceptual Design Stage", IPDO 2013 : 4th Inverse problems, design and optimization symposium, 2013 June 26-28, Albi, ed. by O. Fudym, J.-L. Battaglia, G.S. Dulikravich et al., Albi ; Ecole des Mines d'Albi-Carmaux, 2013 (ISBN 979-10-91526-01-2)
- [19] A. Bragov, F. Antonov, S. Morozov, D. Khominich, "Numerical optimization of the multi-layered composite armor", Light-Weight Armour Group (LWAG) conference-2014

127.6 External links

- Official website

Chapter 128

Quantum annealing

For other uses, see [Annealing \(disambiguation\)](#).

Quantum annealing (QA) is a metaheuristic for finding the global minimum of a given objective function over a given set of candidate solutions (candidate states), by a process using quantum fluctuations. Quantum annealing is used mainly for problems where the search space is discrete (combinatorial optimization problems) with many local minima; such as finding the ground state of a spin glass.^[1] It was formulated in its present form in^[2] though a proposal in a different form had appeared in.^[3]

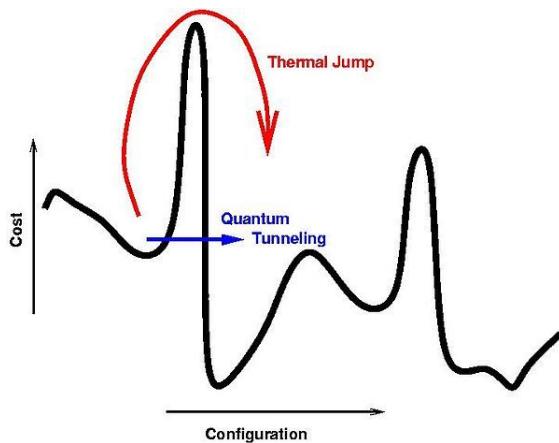
Quantum annealing starts from a quantum-mechanical superposition of all possible states (candidate states) with equal weights. Then the system evolves following the time-dependent Schrödinger equation, a natural quantum-mechanical evolution of physical systems. The amplitudes of all candidate states keep changing, realizing a quantum parallelism, according to the time-dependent strength of the transverse field, which causes quantum tunneling between states. If the rate of change of the transverse-field is slow enough, the system stays close to the ground state of the instantaneous Hamiltonian, i.e., adiabatic quantum computation.^[4] The transverse field is finally switched off, and the system is expected to have reached the ground state of the classical Ising model that corresponds to the solution to the original optimization problem. An experimental demonstration of the success of quantum annealing for random magnets was reported immediately after the initial theoretical proposal.^[5]

128.1 Comparison to simulated annealing

Quantum annealing can be compared to simulated annealing, whose “temperature” parameter plays a similar role to QA’s tunneling field strength. In simulated annealing, the temperature determines the probability of moving to a state of higher “energy” from a single current state. In quantum annealing, the strength of transverse field determines the quantum-mechanical probability to change the amplitudes of all states in parallel. Analyti-

cal^[6] and numerical^[7] evidence suggests that quantum annealing outperforms simulated annealing under certain conditions.

128.2 Quantum mechanics: Analogy & advantage



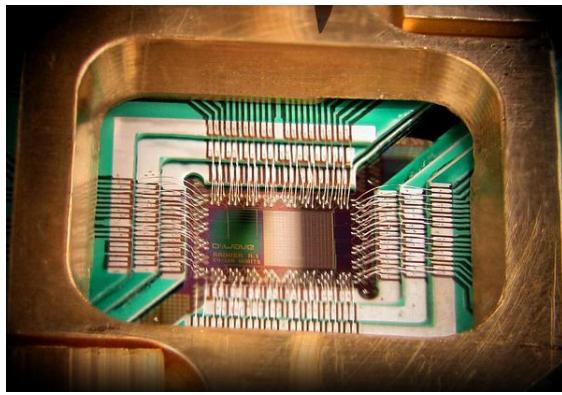
The tunneling field is basically a kinetic energy term that does not commute with the classical potential energy part of the original glass. The whole process can be simulated in a computer using [quantum Monte Carlo](#) (or other stochastic technique), and thus obtain a heuristic algorithm for finding the ground state of the classical glass.

In the case of annealing a purely mathematical *objective function*, one may consider the variables in the problem to be classical degrees of freedom, and the cost functions to be the potential energy function (classical Hamiltonian). Then a suitable term consisting of non-commuting variable(s) (i.e. variables that have non-zero commutator with the variables of the original mathematical problem) has to be introduced artificially in the Hamiltonian to play the role of the tunneling field (kinetic part). Then one may carry out the simulation with the quantum Hamiltonian thus constructed (the original function + non-commuting part) just as described above. Here, there is a choice in selecting the non-commuting term and the efficiency of annealing may depend on that.

It has been demonstrated experimentally as well as theoretically, that quantum annealing can indeed outperform thermal annealing (simulated annealing) in certain cases, especially where the potential energy (cost) landscape consists of very high but thin barriers surrounding shallow local minima. Since thermal transition probabilities ($\sim \exp(-\Delta/k_B T)$; $T \Rightarrow$ Temperature, $k_B \Rightarrow$ Boltzmann constant) depend only on the height Δ of the barriers, for very high barriers, it is extremely difficult for thermal fluctuations to get the system out from such local minima. However, as argued earlier in 1989 by Ray et al. in Ref. [1], the quantum tunneling probability through the same barrier depends not only on the height Δ of the barrier, but also on its width w and is approximately given by $\exp(-\Delta^{1/2}w/\Gamma)$, where Γ is the tunneling field.^[8] If the barriers are thin enough ($w \ll \Delta^{-1/2}$), quantum fluctuations can surely bring the system out of the shallow local minima. For N -spin glasses, $\Delta \sim N$, and with a linear annealing schedule for the transverse field, one gets $\tau \sim \exp(N^{1/2})$ for the annealing time (instead of $\tau \sim \exp(N)$ for thermal annealing).^[9] This $O(N^{1/2})$ advantage in quantum search (compared to the classical effort growing linearly with Δ or N , the problem size) is well established.^[10]

It is speculated that in a quantum computer, such simulations would be much more efficient and exact than that done in a classical computer, because it can perform the tunneling directly, rather than needing to add it by hand. Moreover, it may be able to do this without the tight error controls needed to harness the quantum entanglement used in more traditional quantum algorithms.

128.3 Implementations



Photograph of a chip constructed by D-Wave Systems Inc., mounted and wire-bonded in a sample holder. The D-Wave One's processor is designed to use 128 superconducting logic elements that exhibit controllable and tunable coupling to perform operations.

In 2011, D-Wave Systems announced the first commercial quantum annealer on the market by the name D-Wave One and published a paper in Nature^[11] on its per-

formance. The company claims this system uses a 128 qubit processor chipset.^[12] On May 25, 2011 D-Wave announced that Lockheed Martin Corporation entered into an agreement to purchase a D-Wave One system.^[13] On October 28, 2011 USC's Information Sciences Institute took delivery of Lockheed's D-Wave One.

In May 2013 it was announced that a consortium of Google, NASA Ames and the non-profit Universities Space Research Association purchased an adiabatic quantum computer from D-Wave Systems with 512 qubits.^{[14][15]} An extensive study of its performance as quantum annealer, compared to some classical annealing algorithms, is already available.^[16]

In June 2014, D-Wave announced a new quantum applications ecosystem with computational finance firm 1QB Information Technologies (1QBit) and cancer research group DNA-SEQ to focus on solving real-world problems with quantum hardware.^[17] As the first company dedicated to producing software applications for commercially available quantum computers, 1QBit's research and development arm has focused on D-Wave's quantum annealing processors and has successfully demonstrated that these processors are suitable for solving real-world applications.^[18]

Whether or not the D-Wave machine offers any speedup over a classical computer is highly controversial. A study published in *Science* in June 2014, described as “likely the most thorough and precise study that has been done on the performance of the D-Wave machine”^[19] and “the fairest comparison yet”, found that the D-Wave chip “produced no quantum speedup”.^[20] The researchers, led by Matthias Troyer at the Swiss Federal Institute of Technology, found “no quantum evidence” across the entire range of their tests, and only inconclusive results when looking at subsets of the tests.

D-Wave's architecture differs from traditional quantum computers (none of which exist in practice as of today) in that it has noisy, high error-rate qubits. It is unable to simulate a universal quantum computer and, in particular, cannot execute Shor's algorithm.

128.4 References

- [1] P Ray, BK Chakrabarti, A Chakrabarti “Sherrington-Kirkpatrick model in a transverse field: Absence of replica symmetry breaking due to quantum fluctuations” *Phys. Rev. B* **39**, 11828 (1989)
- [2] T. Kadowaki and H. Nishimori, “Quantum annealing in the transverse Ising model” *Phys. Rev. E* **58**, 5355 (1998)
- [3] A. B. Finilla, M. A. Gomez, C. Sebenik and D. J. Doll, “Quantum annealing: A new method for minimizing multidimensional functions” *Chem. Phys. Lett.* **219**, 343 (1994)
- [4] E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Ludgren and D. Preda, “A Quantum adiabatic evolution algorithm

- applied to random instances of an NP-Complete problem” *Science* **292**, 472 (2001)
- [5] J. Brooke, D. Bitko, T. F. Rosenbaum and G. Aeppli, “Quantum annealing of a disordered magnet”, *Science* **284** 779 (1999)
- [6] S. Morita and H. Nishimori, “Mathematical foundation of quantum annealing”, *J.Math. Phys.* **49**, 125210 (2008)
- [7] G. E. Santoro and E. Tosatti, “Optimization using quantum mechanics: quantum annealing through adiabatic evolution” *J. Phys. A* **39**, R393 (2006)
- [8] A. Das, B.K. Chakrabarti, and R.B. Stinchcombe, “Quantum annealing in a kinetically constrained system”, *Phys. Rev. E* **72** art. 026701 (2005)
- [9] See e.g., S. Mukherjee, and B. K. Chakrabarti, “Multi-variable Optimization: Quantum Annealing & Computation”, arXiv:1408.3262
- [10] J. Roland and N.J. Cerf, “Quantum search by local adiabatic evolution”, *Phys. Rev. A* **65**, 042308 (2002)
- [11] M. W. Johnson et al., “Quantum annealing with manufactured spins”, *Nature* **473** 194 (2011)
- [12] “Learning to program the D-Wave One”. Retrieved 11 May 2011.
- [13] “D-Wave Systems sells its first Quantum Computing System to Lockheed Martin Corporation”. 2011-05-25. Retrieved 2011-05-30.
- [14] N. Jones, Google and NASA snap up quantum computer, *Nature* (2013), doi: 10.1038/nature.2013.12999
- [15] V. N. Smelyanskiy, E. G. Rieffel, S. I. Knysh, C. P. Williams, M. W. Johnson, M. C. Thom, W. G. Macready, K. L. Pudenz, “A Near-Term Quantum Computing Approach for Hard Computational Problems in Space Exploration”, arXiv:1204.2821
- [16] S. Boixo, T. F. Rønnow, S. V. Isakov, Z. Wang, D. Wecker, D. A. Lidar, J. M. Martinis, M. Troyer, “Evidence for quantum annealing with more than one hundred qubits, *Nature Physics*, vol. 10, pp. 218–224 (2014)",
- [17] “D-Wave Systems Building Quantum Application Ecosystem, Announces Partnerships with DNA-SEQ Alliance and 1QBit”. *D-Wave Systems*. Retrieved 22 June 2014.
- [18] “1QBit Research”. *1QBit*. Retrieved 22 June 2014.
- [19] Helmut Katzgraber, quoted in (Cho 2014).
- [20] Cho, Adrian (20 June 2014), “Quantum or not, controversial computer yields no speedup”, *Science* **344** (6190): 1330–1331, doi:10.1126/science.344.6190.1330, PMID 24948715.

128.5 General review articles and books

- G. E. Santoro and E. Tosatti, “Optimization using quantum mechanics: quantum annealing through adiabatic evolution” *J. Phys. A* **39**, R393 (2006).
- A. Das and B. K. Chakrabarti, “Colloquium: Quantum annealing and analog quantum computation” *Rev. Mod. Phys.* **80**, 1061 (2008).
- S. Suzuki, J.-i. Inoue & B. K. Chakrabarti, “Quantum Ising Phases & Transitions in Transverse Ising Models”, Springer, Heidelberg (2013), Chapter 8 on Quantum Annealing.
- V. Bapst, L. Foini, F. Krzakala, G. Semerjian and F. Zamponi, “The quantum adiabatic algorithm applied to random optimization problems: The quantum spin glass perspective”, *Physics Reports* **523** 127 (2013).
- Arnab Das and Bikas K Chakrabarti (Eds.), “Quantum Annealing and Related Optimization Methods”, Lecture Note in Physics, Vol. **679**, Springer, Heidelberg (2005).
- Anjan K. Chandra, Arnab Das and Bikas K Chakrabarti (Eds.), “Quantum Quenching, Annealing and Computation”, Lecture Note in Physics, Vol. **802**, Springer, Heidelberg (2010).
- A. Ghosh and S. Mukherjee, “Quantum Annealing and Computation: A Brief Documentary Note”, arXiv:1310.1339.
- A. Dutta, G. Aeppli, B. K. Chakrabarti, U. Divakaran, T.F. Rosenbaum & D. Sen, “Quantum Phase Transitions in Transverse Field Spin Models: From Statistical Physics to Quantum Information”, Cambridge University Press, Delhi (2015).

Chapter 129

Quasi-Newton method

Quasi-Newton methods are methods used to either find zeroes or local maxima and minima of functions, as an alternative to Newton's method. They can be used if the Jacobian or Hessian is unavailable or is too expensive to compute at every iteration. The “full” Newton's method requires the Jacobian in order to search for zeros, or the Hessian for finding extrema.

129.1 Description of the method

129.1.1 Search for zeroes

Newton's method to find zeroes of a function g of multiple variables is given by: $x_{n+1} = x_n - [J_g(x_n)]^{-1}g(x_n)$ where $J_g(x_n)$ is the **Jacobian matrix** of g evaluated for x_n .

Strictly, any method that replaces the exact Jacobian $J_g(x_n)$ with an approximation is a quasi-Newton method. The chord method (where $J_g(x_n)$ is replaced by $J_g(x_o)$ for all iterations) for instance is a simple example. The methods given below for optimization are other examples. Using methods developed to find extrema in order to find zeroes is not always a good idea as the majority of the methods used to find extrema require that the matrix that is used is symmetrical. While this holds in the context of the search for extrema, it rarely holds when searching for zeroes. Broyden's “good” method and Broyden's “bad” method are two methods commonly used to find extrema that can also be applied to find zeroes. Other methods that can be used are the Column Updating Method, the Inverse Column Updating Method, the Quasi-Newton Least Squares Method and the Inverse Quasi-Newton Least Squares Method.

More recently quasi-Newton methods have been applied to find the solution of multiple coupled systems of equations (e.g. fluid-structure interaction problems). They allow the solution to be found by solving each constituent system separately (which is simpler than the global system) in a cyclic, iterative fashion until the solution of the global system is found.^[1]

129.1.2 Search for extrema

Noting that the search for a minimum or maximum of a single-valued function is nothing else than the search for the zeroes of the gradient of that function, quasi-Newton methods can be readily applied to find extrema of a function. In other words, if g is the gradient of f then searching for the zeroes of the multi-valued function g corresponds to the search for the extrema of the single-valued function f ; the Jacobian of g now becomes the Hessian of f . The main difference is that the Hessian matrix now is a symmetrical matrix, unlike the Jacobian when searching for zeroes. Most quasi-Newton methods used in optimisation exploit this property.

In optimization, **quasi-Newton methods** (a special case of **variable metric methods**) are algorithms for finding local maxima and minima of functions. Quasi-Newton methods are based on Newton's method to find the stationary point of a function, where the gradient is 0. Newton's method assumes that the function can be locally approximated as a quadratic in the region around the optimum, and uses the first and second derivatives to find the stationary point. In higher dimensions, Newton's method uses the gradient and the **Hessian matrix** of second derivatives of the function to be minimized.

In quasi-Newton methods the Hessian matrix does not need to be computed. The Hessian is updated by analyzing successive gradient vectors instead. Quasi-Newton methods are a generalization of the **secant method** to find the root of the first derivative for multidimensional problems. In multiple dimensions the secant equation is under-determined, and quasi-Newton methods differ in how they constrain the solution, typically by adding a simple low-rank update to the current estimate of the Hessian.

The first quasi-Newton algorithm was proposed by William C. Davidon, a physicist working at Argonne National Laboratory. He developed the first quasi-Newton algorithm in 1959: the **DFP updating formula**, which was later popularized by Fletcher and Powell in 1963, but is rarely used today. The most common quasi-Newton algorithms are currently the **SR1 formula** (for symmetric rank one), the **BHHH** method, the widespread **BFGS** method (suggested independently by Broyden, Fletcher,

Goldfarb, and Shanno, in 1970), and its low-memory extension, **L-BFGS**. The Broyden's class is a linear combination of the DFP and BFGS methods.

The SR1 formula does not guarantee the update matrix to maintain positive-definiteness and can be used for indefinite problems. The Broyden's method does not require the update matrix to be symmetric and it is used to find the root of a general system of equations (rather than the gradient) by updating the Jacobian (rather than the Hessian).

One of the chief advantages of quasi-Newton methods over Newton's method is that the Hessian matrix (or, in the case of quasi-Newton methods, its approximation) B does not need to be inverted. Newton's method, and its derivatives such as interior point methods, require the Hessian to be inverted, which is typically implemented by solving a system of linear equations and is often quite costly. In contrast, quasi-Newton methods usually generate an estimate of B^{-1} directly.

As in Newton's method, one uses a second order approximation to find the minimum of a function $f(x)$. The Taylor series of $f(x)$ around an iterate is:

$$f(x_k + \Delta x) \approx f(x_k) + \nabla f(x_k)^T \Delta x + \frac{1}{2} \Delta x^T B \Delta x,$$

where (∇f) is the gradient and B an approximation to the Hessian matrix. The gradient of this approximation (with respect to Δx) is

$$\nabla f(x_k + \Delta x) \approx \nabla f(x_k) + B \Delta x$$

and setting this gradient to zero (which is the objective of optimisation) provides the Newton step:

$$\Delta x = -B^{-1} \nabla f(x_k),$$

The Hessian approximation B is chosen to satisfy

$$\nabla f(x_k + \Delta x) = \nabla f(x_k) + B \Delta x,$$

which is called the *secant equation* (the Taylor series of the gradient itself). In more than one dimension B is underdetermined. In one dimension, solving for B and applying the Newton's step with the updated value is equivalent to the *secant method*. The various quasi-Newton methods differ in their choice of the solution to the secant equation (in one dimension, all the variants are equivalent). Most methods (but with exceptions, such as

Broyden's method) seek a symmetric solution ($B^T = B$); furthermore, the variants listed below can be motivated by finding an update B_{k+1} that is as close as possible to B_k in some norm; that is, $B_{k+1} = \operatorname{argmin}_B \|B - B_k\|_V$ where V is some positive definite matrix that defines the norm. An approximate initial value of $B_0 = I * x$ is often sufficient to achieve rapid convergence. Note that B_0 should be positive definite. The unknown x_k is updated applying the Newton's step calculated using the current approximate Hessian matrix B_k

- $\Delta x_k = -\alpha_k B_k^{-1} \nabla f(x_k)$, with α chosen to satisfy the Wolfe conditions;
- $x_{k+1} = x_k + \Delta x_k$;
- The gradient computed at the new point $\nabla f(x_{k+1})$, and

$$y_k = \nabla f(x_{k+1}) - \nabla f(x_k),$$

is used to update the approximate Hessian B_{k+1} , or directly its inverse $H_{k+1} = B_{k+1}^{-1}$ using the Sherman-Morrison formula.

- A key property of the BFGS and DFP updates is that if B_k is positive definite and α_k is chosen to satisfy the Wolfe conditions then B_{k+1} is also positive definite.

The most popular update formulas are:

Other methods are Pearson's method, McCormick's Method, the Powell symmetric Broyden (PSB) method and Greenstadt's method.^[1]

129.2 Implementations

Owing to their success, there are implementations of quasi-Newton methods in almost all programming languages. The NAG Library contains several routines^[2] for minimizing or maximizing a function^[3] which use quasi-Newton algorithms.

Scipy.optimize has fmin_bfgs.

GNU Octave uses a form of BFGS in its 'fsolve' function, with trust region extensions.

In MATLAB's Optimization Toolbox, the fminunc function uses (among other methods) the BFGS Quasi-Newton method. Many of the constrained methods of the Optimization toolbox use BFGS and the variant L-BFGS. Many user-contributed quasi-Newton routines are available on MATLAB's file exchange.

Mathematica includes quasi-Newton solvers. R's optim general-purpose optimizer routine uses the BFGS method by using method="BFGS". In the SciPy extension to Python, the scipy.optimize.minimize function includes, among other methods, a BFGS implementation.

129.3 See also

- Newton's method in optimization
- Newton's method
- DFP updating formula
- BFGS method
 - L-BFGS
 - OWL-QN
- SR1 formula
- Broyden's Method
- Quasi-Newton Least Squares Method

129.4 References

- [1] Haelterman, Rob (2009). "Analytical study of the least squares quasi-Newton method for interaction problems". *PhD Thesis, Ghent University*. Retrieved 2014-08-14.
- [2] The Numerical Algorithms Group. "Keyword Index: Quasi-Newton". *NAG Library Manual, Mark 23*. Retrieved 2012-02-09.
- [3] The Numerical Algorithms Group. "E04 – Minimizing or Maximizing a Function". *NAG Library Manual, Mark 23*. Retrieved 2012-02-09.

129.5 Further reading

- Bonnans, J. F., Gilbert, J.Ch., Lemaréchal, C. and Sagastizábal, C.A. (2006), *Numerical optimization, theoretical and numerical aspects*. Second edition. Springer. ISBN 978-3-540-35445-1.
- William C. Davidon, VARIABLE METRIC METHOD FOR MINIMIZATION, SIOPT Volume 1 Issue 1, Pages 1–17, 1991.
- Fletcher, Roger (1987), *Practical methods of optimization* (2nd ed.), New York: John Wiley & Sons, ISBN 978-0-471-91547-8.
- Nocedal, Jorge & Wright, Stephen J. (1999). Numerical Optimization. Springer-Verlag. ISBN 0-387-98793-2.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Section 10.9. Quasi-Newton or Variable Metric Methods in Multidimensions". *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.

Chapter 130

Random optimization

Random optimization (RO) is a family of numerical optimization methods that do not require the gradient of the problem to be optimized and RO can hence be used on functions that are not **continuous** or **differentiable**. Such optimization methods are also known as direct-search, derivative-free, or black-box methods.

The name, random optimization, is attributed to Matyas^[1] who made an early presentation of RO along with basic mathematical analysis. RO works by iteratively moving to better positions in the search-space which are sampled using e.g. a **normal distribution** surrounding the current position.

130.1 Algorithm

Let $f: \mathbb{R}^n \rightarrow \mathbb{R}$ be the fitness or cost function which must be minimized. Let $\mathbf{x} \in \mathbb{R}^n$ designate a position or candidate solution in the search-space. The basic RO algorithm can then be described as:

- Initialize \mathbf{x} with a random position in the search-space.
- Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat the following:
 - Sample a new position \mathbf{y} by adding a **normally distributed** random vector to the current position \mathbf{x}
 - If $(f(\mathbf{y}) < f(\mathbf{x}))$ then move to the new position by setting $\mathbf{x} = \mathbf{y}$
- Now \mathbf{x} holds the best-found position.

This algorithm corresponds to a (1+1) Evolution Strategy with constant step-size.

130.2 Convergence and variants

Matyas showed the basic form of RO converges to the optimum of a simple **unimodal function** by using a limit-

proof which shows convergence to the optimum is certain to occur if a potentially infinite number of iterations are performed. However, this proof is not useful in practise because a finite number of iterations can only be executed. In fact, such a theoretical limit-proof will also show that purely random sampling of the search-space will inevitably yield a sample arbitrarily close to the optimum.

Mathematical analyses are also conducted by Baba^[2] and Solis and Wets^[3] to establish that convergence to a region surrounding the optimum is inevitable under some mild conditions for RO variants using other probability distributions for the sampling. An estimate on the number of iterations required to approach the optimum is derived by Dorea.^[4] These analyses are criticized through empirical experiments by Sarma^[5] who used the optimizer variants of Baba and Dorea on two real-world problems, showing the optimum to be approached very slowly and moreover that the methods were actually unable to locate a solution of adequate fitness, unless the process was started sufficiently close to the optimum to begin with.

130.3 See also

- **Random search** is a closely related family of optimization methods which sample from a **hypersphere** instead of a normal distribution.
- **Luus-Jaakola** is a closely related optimization method using a **uniform distribution** in its sampling and a simple formula for exponentially decreasing the sampling range.
- **Pattern search** takes steps along the axes of the search-space using exponentially decreasing step sizes.
- **Stochastic optimization**

130.4 References

[1] Matyas, J. (1965). "Random optimization". *Automation and Remote Control* **26** (2): 246–253.

- [2] Baba, N. (1981). "Convergence of a random optimization method for constrained optimization problems". *Journal of Optimization Theory and Applications* **33** (4): 451–461. doi:10.1007/bf00935752.
- [3] Solis, F.J.; Wets, R.J-B. (1981). "Minimization by random search techniques". *Mathematics of Operation Research* **6** (1): 19–30. doi:10.1287/moor.6.1.19.
- [4] Dorea, C.C.Y. (1983). "Expected number of steps of a random optimization method". *Journal of Optimization Theory and Applications* **39** (3): 165–171. doi:10.1007/bf00934526.
- [5] Sarma, M.S. (1990). "On the convergence of the Baba and Dorea random optimization methods". *Journal of Optimization Theory and Applications* **66** (2): 337–343. doi:10.1007/bf00939542.

Chapter 131

Random search

Random search (RS) is a family of numerical optimization methods that do not require the gradient of the problem to be optimized, and RS can hence be used on functions that are not continuous or differentiable. Such optimization methods are also known as direct-search, derivative-free, or black-box methods.

The name “random search” is attributed to Rastrigin^[1] who made an early presentation of RS along with basic mathematical analysis. RS works by iteratively moving to better positions in the search-space, which are sampled from a hypersphere surrounding the current position.

131.1 Algorithm

Let $f: \mathbb{R}^n \rightarrow \mathbb{R}$ be the fitness or cost function which must be minimized. Let $\mathbf{x} \in \mathbb{R}^n$ designate a position or candidate solution in the search-space. The basic RS algorithm can then be described as:

- Initialize \mathbf{x} with a random position in the search-space.
- Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat the following:
 - Sample a new position \mathbf{y} from the hypersphere of a given radius surrounding the current position \mathbf{x} (see e.g. Marsaglia’s technique for sampling a hypersphere.)
 - If ($f(\mathbf{y}) < f(\mathbf{x})$) then move to the new position by setting $\mathbf{x} = \mathbf{y}$
 - Now \mathbf{x} holds the best-found position.

131.2 Variants

A number of RS variants have been introduced in the literature:

- Fixed Step Size Random Search (FSSRS) is Rastigin’s^[1] basic algorithm which samples from a hypersphere of fixed radius.

- Optimum Step Size Random Search (OSSRS) by Schumer and Steiglitz^[2] is primarily a theoretical study on how to optimally adjust the radius of the hypersphere so as to allow for speedy convergence to the optimum. An actual implementation of the OSSRS needs to approximate this optimal radius by repeated sampling and is therefore expensive to execute.
- Adaptive Step Size Random Search (ASSRS) by Schumer and Steiglitz^[2] attempts to heuristically adapt the hypersphere’s radius: two new candidate solutions are generated, one with the current nominal step size and one with a larger step-size. The larger step size becomes the new nominal step size if and only if it leads to a larger improvement. If for several iterations neither of the steps leads to an improvement, the nominal step size is reduced.
- Optimized Relative Step Size Random Search (ORSSRS) by Schrack and Choi^[3] approximate the optimal step size by a simple exponential decrease. However, the formula for computing the decrease-factor is somewhat complicated.

131.3 See also

- Random optimization is a closely related family of optimization methods which sample from a normal distribution instead of a hypersphere.
- Luus–Jaakola is a closely related optimization method using a uniform distribution in its sampling and a simple formula for exponentially decreasing the sampling range.
- Pattern search takes steps along the axes of the search-space using exponentially decreasing step sizes.

131.4 References

[1] Rastrigin, L.A. (1963). “The convergence of the random search method in the extremal control of a many param-

- eter system”. *Automation and Remote Control* **24** (10): 1337–1342.
- [2] Schumer, M.A.; Steiglitz, K. (1968). “Adaptive step size random search”. *IEEE Transactions on Automatic Control* **13** (3): 270–276. doi:10.1109/tac.1968.1098903.
- [3] Schrack, G.; Choit, M. (1976). “Optimized relative step size random searches”. *Mathematical Programming* **10** (1): 230–244. doi:10.1007/bf01580669.

Chapter 132

Rosenbrock methods

Rosenbrock methods refers to either of two distinct ideas in numerical computation, both named for Howard H. Rosenbrock.

132.1 Numerical solution of differential equations

Rosenbrock methods for stiff differential equations are a family of multistep methods for solving ordinary differential equations that contain a wide range of characteristic timescales.^{[1][2]} They are related to the implicit Runge-Kutta methods^[3] and are also known as Kaps-Rentrop methods.^[4]

132.2 Search method

Rosenbrock search is a numerical optimization algorithm applicable to optimization problems in which the objective function is inexpensive to compute and the derivative either does not exist or cannot be computed efficiently.^[5] The idea of Rosenbrock search is also used to initialize some root-finding routines, such as **fzero** (based on Brent's method) in Matlab. Rosenbrock search is a form of derivate-free search but may perform better on functions with sharp ridges.^[6] The method often identifies such a ridge which, in many applications, leads to a solution.^[7]

132.3 See also

- Rosenbrock function
- Adaptive coordinate descent

132.4 References

[1] H. H. Rosenbrock, “Some general implicit processes for the numerical solution of differential equations”, The Computer Journal (1963) 5(4): 329-330

- [2] Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). “Section 17.5.1. Rosenbrock Methods”. *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.
- [3] http://www.cfm.brown.edu/people/jansh/page5/page10/page40/assets/Yu_Talk.pdf
- [4] <http://mathworld.wolfram.com/RosenbrockMethods.html>
- [5] H. H. Rosenbrock, “An Automatic Method for Finding the Greatest or Least Value of a Function”, The Computer Journal (1960) 3(3): 175-184
- [6] Leader, Jeffery J. (2004). *Numerical Analysis and Scientific Computation*. Addison Wesley. ISBN 0-201-73499-0.
- [7] Shoup, T., Mistree, F., Optimization methods: with applications for personal computers, 1987, Prentice Hall, pg. 120

132.5 External links

- <http://www.applied-mathematics.net/optimization/rosenbrock.html>

Chapter 133

Search-based software engineering

Search-based software engineering (SBSE) applies metaheuristic search techniques such as genetic algorithms, simulated annealing and tabu search to software engineering problems. Many activities in software engineering can be stated as optimization problems. Optimization techniques of operations research such as linear programming or dynamic programming are mostly impractical for large scale software engineering problems because of their computational complexity. Researchers and practitioners use metaheuristic search techniques to find near-optimal or “good-enough” solutions.

SBSE problems can be divided into two types:

- black-box optimization problems, for example, assigning people to tasks (a typical combinatorial optimization problem).
- white-box problems where operations on source code need to be considered.^[1]

133.1 Definition

SBSE converts a software engineering problem and into a computational search problem that can be tackled with a metaheuristic. This involves defining a search space, or the set of possible solutions. This space is typically too large to be explored exhaustively, suggesting a metaheuristic approach. A metric ^[2] (also called a fitness function, cost function, objective function or quality measure) is then used to measure the quality of potential solutions. Many software engineering problems can be reformulated as a computational search problem.^[3]

The term "search-based application", in contrast, refers to using search engine technology, rather than search techniques, in another industrial application.

133.2 Brief history

One of the earliest attempts to apply optimization to a software engineering problem was reported by Webb Miller and David Spooner in 1976 in the area of software

testing.^[4] In 1992, S.Xanthakis and his colleagues applied a search technique to a software engineering problem for the first time.^[5] The term SBSE was first used in 2001 by Harman and Jones.^[6] The research community grew to include more than 800 authors by 2013, spanning approximately 270 institutions in 40 countries.

133.3 Application areas

Search-based software engineering is applicable to almost all phases of the software development process. Software testing has been one of the major applications.^[7] Search techniques have been applied to other software engineering activities, for instance, requirements analysis,^{[8][9]} design,^[10] development,^[11] and maintenance.^[12]

133.3.1 Requirements engineering

Requirements engineering is the process by which the needs of a software's users and environment are determined and managed. Search-based methods have been used for requirements selection and optimisation with the goal of finding the best possible subset of requirements that matches user requests amid constraints such as limited resources and interdependencies between requirements. This problem is often tackled as a multiple-criteria decision-making problem and, generally involves presenting the decision maker with a set of good compromises between cost and user satisfaction.^{[13][14]}

133.3.2 Debugging and maintenance

Identifying a software bug (or a code smell) and then debugging (or refactoring) the software is largely a manual and labor-intensive endeavor, though the process is tool-supported. One objective of SBSE is to automatically identify and fix bugs (for example via mutation testing).

Genetic programming, a biologically-inspired technique that involves evolving programs through the use of

crossover and mutation, has been used to search for repairs to programs by altering a few lines of source code. The GenProg Evolutionary Program Repair software repaired 55 out of 105 bugs for approximately \$8 each in one test.^[15]

Coevolution adopts a “predator and prey” metaphor in which a suite of programs and a suite of unit tests evolve together and influence each other.^[16]

133.3.3 Testing

Search-based software engineering has been applied to software testing, including automatic generation of test cases (test data), test case minimization and test case prioritization. Regression testing has also received some attention.

133.3.4 Optimizing software

The use of SBSE in program optimization, or modifying a piece of software to make it more efficient in terms of speed and resource use, has been the object of successful research. In one instance, a 50,000 line program was genetically improved, resulting in a program 70 times faster on average.^[17]

133.3.5 Project management

A number of decisions that are normally made by a project manager can be done automatically, for example, project scheduling.^[18]

133.4 Tools

Tools available for SBSE include OpenPAT.^[19] and Evo-suite^[20] and a code coverage measurement for Python^[21]

133.5 Methods and techniques

A number of methods and techniques are available, including:

- Profiling^[22] via instrumentation in order to monitor certain parts of a program as it is executed.
- Obtaining an abstract syntax tree associated with the program, which can be automatically examined to gain insights into its structure.
- Applications of program slicing relevant to SBSE include software maintenance, optimization and program analysis.
- Code coverage allows measuring how much of the code is executed with a given set of input data.

- Static program analysis

133.6 Industry acceptance

As a relatively new area of research, SBSE does not yet experience broad industry acceptance. Software engineers are reluctant to adopt tools over which they have little control or that generate solutions that are unlike those that humans produce.^[23] In the context of SBSE use in fixing or improving programs, developers need to be confident that any automatically produced modification does not generate unexpected behavior outside the scope of a system’s requirements and testing environment. Considering that fully automated programming has yet to be achieved, a desirable property of such modifications would be that they need to be easily understood by humans to support maintenance activities.^[24]

Another concern is that SBSE might make the software engineer redundant. Supporters claim that the motivation for SBSE is to enhance the relationship between the engineer and the program.^[25]

133.7 See also

- Program analysis (computer science)
- Dynamic program analysis

133.8 References

- [1] Harman, Mark (2010). “Why Source Code Analysis and Manipulation Will Always be Important”. *10th IEEE Working Conference on Source Code Analysis and Manipulation (SCAM 2010)*. 10th IEEE Working Conference on Source Code Analysis and Manipulation (SCAM 2010). pp. 7–19. doi:10.1109/SCAM.2010.28.
- [2] Harman, Mark; John A. Clark (2004). “Metrics are fitness functions too”. *Proceedings of the 10th International Symposium on Software Metrics, 2004*. 10th International Symposium on Software Metrics, 2004. pp. 58–69. doi:10.1109/METRIC.2004.1357891.
- [3] Clark, John A.; Dolado, José Javier; Harman, Mark; Hierons, Robert M.; Jones, Bryan F.; Lumkin, M.; Mitchell, Brian S.; Mancoridis, Spiros; Rees, K.; Roper, Marc; Shepperd, Martin J. (2003). “Reformulating software engineering as a search problem”. *IEEE Proceedings - Software* **150** (3): 161–175. doi:10.1049/ip-sen:20030559. ISSN 1462-5970.
- [4] Miller, Webb; Spooner, David L. (1976). “Automatic Generation of Floating-Point Test Data”. *IEEE Transactions on Software Engineering* **SE-2** (3): 223–226. doi:10.1109/TSE.1976.233818. ISSN 0098-5589.

- [5] S. Xanthakis, C. Ellis, C. Skourlas, A. Le Gall, S. Katsikas and K. Karapoulios, "Application of genetic algorithms to software testing," in *Proceedings of the 5th International Conference on Software Engineering and its Applications*, Toulouse, France, 1992, pp. 625–636.
- [6] Harman, Mark; Jones, Bryan F. (2001-12-15). "Search-based software engineering". *Information and Software Technology* **43** (14): 833–839. doi:10.1016/S0950-5849(01)00189-6. ISSN 0950-5849. Retrieved 2013-10-31.
- [7] McMinn, Phil (2004). "Search-based software test data generation: a survey". *Software Testing, Verification and Reliability* **14** (2): 105–156. doi:10.1002/stvr.294. ISSN 1099-1689. Retrieved 2013-10-31.
- [8] Greer, Des; Ruhe, Guenther (2004-03-15). "Software release planning: an evolutionary and iterative approach". *Information and Software Technology* **46** (4): 243–253. doi:10.1016/j.infsof.2003.07.002. ISSN 0950-5849. Retrieved 2013-09-06.
- [9] Colares, Felipe; Souza, Jerffeson; Carmo, Raphael; Pádua, Clarindo; Mateus, Geraldo R. (2009). "A New Approach to the Software Release Planning". *XXIII Brazilian Symposium on Software Engineering, 2009. SBES '09*. XXIII Brazilian Symposium on Software Engineering, 2009. SBES '09. pp. 207–215. doi:10.1109/SBES.2009.23.
- [10] Clark, John A.; Jacob, Jeremy L. (2001-12-15). "Protocols are programs too: the meta-heuristic search for security protocols". *Information and Software Technology* **43** (14): 891–904. doi:10.1016/S0950-5849(01)00195-1. ISSN 0950-5849. Retrieved 2013-10-31.
- [11] Alba, Enrique; Chicano, J. Francisco (2007-06-01). "Software project management with GAs". *Information Sciences* **177** (11): 2380–2401. doi:10.1016/j.ins.2006.12.020. ISSN 0020-0255. Retrieved 2013-10-31.
- [12] Antoniol, Giuliano; Di Penta, Massimiliano; Harman, Mark (2005). "Search-based techniques applied to optimization of project planning for a massive maintenance project". *Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005. ICSM'05*. Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005. ICSM'05. pp. 240–249. doi:10.1109/ICSM.2005.79.
- [13] Zhang, Yuanyuan (February 2010). *Multi-Objective Search-based Requirements Selection and Optimisation* (Ph.D.). Strand, London, UK: University of London.
- [14] Y. Zhang and M. Harman and S. L. Lim, "Search Based Optimization of Requirements Interaction Management," Department of Computer Science, University College London, Research Note RN/11/12, 2011.
- [15] Le Goues, Claire; Dewey-Vogt, Michael; Forrest, Stephanie; Weimer, Westley (2012). "A systematic study of automated program repair: Fixing 55 out of 105 bugs for \$8 each". *2012 34th International Conference on Software Engineering (ICSE)*. 2012 34th International Conference on Software Engineering (ICSE). pp. 3–13. doi:10.1109/ICSE.2012.6227211.
- [16] Arcuri, Andrea; Yao, Xin (2008). "A novel co-evolutionary approach to automatic software bug fixing". *IEEE Congress on Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence)*. IEEE Congress on Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). pp. 162–168. doi:10.1109/CEC.2008.4630793.
- [17] Langdon, William B.; Harman, Mark. "Optimising Existing Software with Genetic Programming". *IEEE Transactions on Evolutionary Computation*.
- [18] Minku, Leandro L.; Sudholt, Dirk; Yao, Xin (2012). "Evolutionary algorithms for the project scheduling problem: runtime analysis and improved design". *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference. GECCO '12*. New York, NY, USA: ACM. pp. 1221–1228. doi:10.1145/2330163.2330332. ISBN 978-1-4503-1177-9. Retrieved 2013-10-31.
- [19] Mayo, M.; Spacey, S. (2013). *Predicting Regression Test Failures Using Genetic Algorithm-Selected Dynamic Performance Analysis Metrics (PDF)*. *Proceedings of the 5th International Symposium on Search-Based Software Engineering (SSBSE)* **8084**: 158–171.
- [20] (<http://www.evosuite.org/>)
- [21] (<https://pypi.python.org/pypi/coverage>)
- [22] (<http://java-source.net/open-source/profilers>)
- [23] Jones, Derek (18 October 2013). "Programming using genetic algorithms: isn't that what humans already do ;-)". *The Shape of Code*. Retrieved 31 October 2013.
- [24] Le Goues, Claire; Forrest, Stephanie; Weimer, Westley (2013-09-01). "Current challenges in automatic software repair". *Software Quality Journal* **21** (3): 421–443. doi:10.1007/s11219-013-9208-0. ISSN 1573-1367. Retrieved 2013-10-31.
- [25] Simons, Christopher L. (May 2013). *Whither (away) software engineers in SBSE?*. First International Workshop on Combining Modelling with Search-Based Software Engineering, First International Workshop on Combining Modelling with Search-Based Software Engineering. San Francisco, USA: IEEE Press. pp. 49–50. Retrieved 2013-10-31.

133.9 External links

- Repository of publications on SBSE
- Metaheuristics and Software Engineering
- Software-artifact Infrastructure Repository
- International Conference on Software Engineering

- Genetic and Evolutionary Computation (GECCO)
- Google Scholar page on Search-based software engineering

Chapter 134

Sequence-dependent setup

In Job Shop Scheduling, **sequence-dependent setup** occurs when a machine's setup time or cost for a particular job is determined by not only by that job but also by the previous job that the machine is currently setup for. Information on sequence-dependent setup times or costs for N jobs can be stored in a N -by- N matrix with all diagonal entries being zero. Diagonal entries must be zero since they correspond to the condition that the machine is already setup for the next job; hence the setup time and/or cost is zero.

In a job shop, we are usually concerned with completing all of the jobs in the shortest possible time or least possible cost. The time it takes to complete all of the jobs is called Makespan. If set up time or cost does not exist, or is not sequence-dependent, then in a single machine N -job situation, the makespan or total cost is not sequence-dependent. However, when set up times or costs are sequence-dependent, makespan or total cost too becomes variable. The size and complexity of the problem grow rapidly as the number of jobs and machines in the problem increase.

134.1 See also

- Changeover
- Single-Minute Exchange of Die
- Genetic algorithm scheduling

Chapter 135

Sequential minimal optimization

Sequential minimal optimization (SMO) is an algorithm for solving the quadratic programming (QP) problem that arises during the training of support vector machines. It was invented by John Platt in 1998 at Microsoft Research.^[1] SMO is widely used for training support vector machines and is implemented by the popular LIBSVM tool.^{[2][3]} The publication of the SMO algorithm in 1998 has generated a lot of excitement in the SVM community, as previously available methods for SVM training were much more complex and required expensive third-party QP solvers.^[4]

135.1 Optimization problem

Main article: Support vector machine

Consider a binary classification problem with a dataset $(x_1, y_1), \dots, (x_n, y_n)$, where x_i is an input vector and $y_i \in \{-1, +1\}$ is a binary label corresponding to it. A soft-margin support vector machine is trained by solving a quadratic programming problem, which is expressed in the dual form as follows:

$$\begin{aligned} \max_{\alpha} & \sum_{i=1}^n \alpha_i \\ & - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j K(x_i, x_j) \alpha_i \alpha_j, \\ \text{subject to:} & \\ & 0 \leq \alpha_i \leq C, \quad \text{for } i = 1, 2, \dots, n, \\ & \sum_{i=1}^n y_i \alpha_i = 0 \end{aligned}$$

where C is an SVM hyperparameter and $K(x_i, x_j)$ is the kernel function, both supplied by the user; and the variables α_i are Lagrange multipliers.

135.2 Algorithm

SMO is an iterative algorithm for solving the optimization problem described above. SMO breaks this problem into a series of smallest possible sub-problems, which are then solved analytically. Because of the linear equality constraint involving the Lagrange multipliers α_i , the

smallest possible problem involves two such multipliers. Then, for any two multipliers α_1 and α_2 , the constraints are reduced to:

$$0 \leq \alpha_1, \alpha_2 \leq C,$$

$$y_1 \alpha_1 + y_2 \alpha_2 = k,$$

and this reduced problem can be solved analytically: one needs to find a minimum of a one-dimensional quadratic function. k is the negative of the sum over the rest of terms in the equality constraint, which is fixed in each iteration.

The algorithm proceeds as follows:

1. Find a Lagrange multiplier α_1 that violates the Karush–Kuhn–Tucker (KKT) conditions for the optimization problem.
2. Pick a second multiplier α_2 and optimize the pair (α_1, α_2) .
3. Repeat steps 1 and 2 until convergence.

When all the Lagrange multipliers satisfy the KKT conditions (within a user-defined tolerance), the problem has been solved. Although this algorithm is guaranteed to converge, heuristics are used to choose the pair of multipliers so as to accelerate the rate of convergence.

135.3 See also

- Kernel perceptron

135.4 References

- [1] Platt, John (1998). *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*, CiteSeerX: 10.1.1.43.4376
- [2] Chang, Chih-Chung; Lin, Chih-Jen (2011). “LIBSVM: A library for support vector machines”. *ACM Transactions on Intelligent Systems and Technology* 2 (3).

- [3] Luca Zanni (2006). *Parallel Software for Training Large Scale Support Vector Machines on Multiprocessor Systems*.
- [4] Rifkin, Ryan (2002), “Everything Old is New Again: a Fresh Look at Historical Approaches in Machine Learning”, *Ph.D. thesis*: 18

Chapter 136

Shuffled frog leaping algorithm

The **shuffled frog leaping algorithm (SFLA)** is an optimization algorithm used in artificial intelligence (AI). It is like a [genetic algorithm](#). [1]

136.1 References

- [1] Muzaffar Eusuff , Kevin Lansey & Fayzul Pasha (2006) Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization, Engineering Optimization, 38:2, 129–154, DOI: 10.1080/03052150500384759

Chapter 137

Simultaneous perturbation stochastic approximation

Simultaneous perturbation stochastic approximation (SPSA) is an **algorithmic** method for optimizing systems with multiple unknown parameters. It is a type of **stochastic approximation** algorithm. As an optimization method, it is appropriately suited to large-scale population models, adaptive modeling, simulation optimization, and atmospheric modeling. Many examples are presented at the SPSA website <http://www.jhuapl.edu/SPSA>. A comprehensive recent book on the subject is Bhatnagar et al. (2013). An early paper on the subject is Spall (1987) and the foundational paper providing the key theory and justification is Spall (1992).

SPSA is a descent method capable of finding global minima. Its main feature is the gradient approximation that requires only two measurements of the objective function, regardless of the dimension of the optimization problem. Recall that we want to find the optimal control u^* with loss function $J(u)$:

$$u^* = \arg \min_{u \in U} J(u).$$

Both Finite Differences Stochastic Approximation (FDSA) and SPSA use the same iterative process:

$$u_{n+1} = u_n - a_n \hat{g}_n(u_n),$$

where $u_n = ((u_n)_1, (u_n)_2, \dots, (u_n)_p)^T$ represents the n^{th} iterate, $\hat{g}_n(u_n)$ is the estimate of the gradient of the objective function $g(u) = \frac{\partial}{\partial u} J(u)$ evaluated at u_n , and $\{a_n\}$ is a positive number sequence converging to 0. If u_n is a p -dimensional vector, the i^{th} component of the symmetric finite difference gradient estimator is:

$$\text{FD: } (\hat{g}_n(u_n))_i = \frac{J(u_n + c_n e_i) - J(u_n - c_n e_i)}{2c_n},$$

$1 \leq i \leq p$, where e_i is the unit vector with a 1 in the i^{th} place, and c_n is a small positive number that decreases with n . With this method, $2p$ evaluations of J for each g_n are needed. Clearly, when p is large, this estimator loses efficiency.

Let now Δ_n be a random perturbation vector. The i^{th} component of the stochastic perturbation gradient estimator is:

$$\text{SP: } (\hat{g}_n(u_n))_i = \frac{J(u_n + c_n \Delta_n) - J(u_n - c_n \Delta_n)}{2c_n (\Delta_n)_i}.$$

Remark that FD perturbs only one direction at a time, while the SP estimator disturbs all directions at the same time (the numerator is identical in all p components). The number of loss function measurements needed in the SPSA method for each g_n is always 2, independent of the **dimension** p . Thus, SPSA uses p times fewer function evaluations than FDSA, which makes it a lot more efficient.

Simple experiments with $p=2$ showed that SPSA converges in the same number of iterations as FDSA. The latter follows **approximately the steepest** descent direction, behaving like the gradient method. On the other hand, SPSA, with the random search direction, does not follow exactly the gradient path. In average though, it tracks it nearly because the gradient approximation is an almost **unbiased** estimator of the gradient, as shown in the following lemma.

137.1 Convergence lemma

Denote by

$$b_n = E[\hat{g}_n | u_n] - \nabla J(u_n)$$

the bias in the estimator \hat{g}_n . Assume that $\{(\Delta_n)_i\}$ are all mutually independent with zero-mean, bounded second moments, and $E(|(\Delta_n)_i|^{-1})$ uniformly bounded. Then $b_n \rightarrow 0$ w.p. 1.

137.2 Sketch of the proof

The main idea is to use conditioning on Δ_n to express $E[(\hat{g}_n)_i]$ and then to use a second order Taylor expansion

of $J(u_n + c_n \Delta_n)_i$ and $J(u_n - c_n \Delta_n)_i$. After algebraic manipulations using the zero mean and the independence of $\{(\Delta_n)_i\}$, we get

$$E[(\hat{g}_n)_i] = (g_n)_i + O(c_n^2)$$

The result follows from the hypothesis that $c_n \rightarrow 0$.

Next we resume some of the hypotheses under which u_t converges in probability to the set of global minima of $J(u)$. The efficiency of the method depends on the shape of $J(u)$, the values of the parameters a_k and c_k and the distribution of the perturbation terms Δ_{ki} . First, the algorithm parameters must satisfy the following conditions:

- $a_t > 0$, $a_t \rightarrow 0$ when $t \rightarrow \infty$ and $\sum_{t=1}^{\infty} a_t = \infty$. A good choice would be $a_t = \frac{a}{t}$; $a > 0$;
- $c_t = \frac{c}{t^{\gamma}}$, where $c > 0$, $\gamma \in [\frac{1}{6}, \frac{1}{2}]$;
- $\sum_{t=1}^{\infty} (\frac{a_t}{c_t})^2 < \infty$
- Δ_{ti} must be mutually independent zero-mean random variables, symmetrically distributed about zero, with $\Delta_{ki} < a_1 < \infty$. The inverse first and second moments of the Δ_{ti} must be finite.

A good choice for Δ_{ki} is Bernoulli $+/- 1$ with probability 0.5 (other choices are possible too). The uniform and normal distributions do not satisfy the finite inverse moment conditions, so can not be used.

The loss function $J(u)$ must be thrice continuously differentiable and the individual elements of the third derivative must be bounded: $|J^{(3)}(u)| < a_3 < \infty$. Also, $|J(u)| \rightarrow \infty$ as $u \rightarrow \infty$.

In addition, ∇J must be Lipschitz continuous, bounded and the ODE $\dot{u} = g(u)$ must have a unique solution for each initial condition. Under these conditions and a few others, u_k converges in probability to the set of global minima of $J(u)$ (see Maryak and Chin, 2008).

137.3 References

- Bhatnagar, S., Prasad, H. L., and Prashanth, L. A. (2013), *Stochastic Recursive Algorithms for Optimization: Simultaneous Perturbation Methods*, Springer .
- Hirokami, T., Maeda, Y., Tsukada, H. (2006) “Parameter estimation using simultaneous perturbation stochastic approximation”, Electrical Engineering in Japan, 154 (2), 30–3
- Maryak, J.L., and Chin, D.C. (2008), “Global Random Optimization by Simultaneous Perturbation Stochastic Approximation,” *IEEE Transactions on Automatic Control*, vol. 53, pp. 780-783.

- Spall, J. C. (1987), “A Stochastic Approximation Technique for Generating Maximum Likelihood Parameter Estimates,” *Proceedings of the American Control Conference*, Minneapolis, MN, June 1987, pp. 1161–1167.
- Spall, J. C. (1992), “Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation,” *IEEE Transactions on Automatic Control*, vol. 37(3), pp. 332–341.
- Spall, J.C. (1998). “Overview of the Simultaneous Perturbation Method for Efficient Optimization” 2. *Johns Hopkins APL Technical Digest*, 19(4), 482–492.
- Spall, J.C. (2003) *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*, Wiley. ISBN 0-471-33052-3 (Chapter 7)

Chapter 138

Social cognitive optimization

Social cognitive optimization (SCO) is a population-based metaheuristic optimization algorithm which was developed in 2002.^[1] This algorithm is based on the social cognitive theory, and the key point of the ergodicity is the process of individual learning of a set of agents with their own memory and their social learning with the knowledge points in the social sharing library. It has been used for solving continuous optimization,^{[2][3]} integer programming,^[4] and combinatorial optimization problems. It has been incorporated into the NLPSolver extension of Calc in Apache OpenOffice.

138.1 Algorithm

Let $f(x)$ be a global optimization problem, where x is a state in the problem space S . In SCO, each state is called a *knowledge point*, and the function f is the *goodness function*.

In SCO, there are a population of N_c cognitive agents solving in parallel, with a social sharing library. Each agent holds a private memory containing one knowledge point, and the social sharing library contains a set of N_L knowledge points. The algorithm runs in T iterative learning cycles. By running as a **Markov chain** process, the system behavior in the t th cycle only depends on the system status in the $(t - 1)$ th cycle. The process flow is as follows:

- [1. Initialization]: Initialize the private knowledge point x_i in the memory of each agent i , and all knowledge points in the social sharing library X , normally at random in the problem space S .
- [2. Learning cycle]: At each cycle t ($t = 1, \dots, T$) :
 - [2.1. Observational learning] For each agent i ($i = 1, \dots, N_c$) :
 - [2.1.1. Model selection]: Find a high-quality *model point* x_M in $X(t)$, normally realized using **tournament selection**, which returns the best knowledge point from randomly selected τ_B points.

- [2.1.2. Quality Evaluation]: Compare the private knowledge point $x_i(t)$ and the model point x_M , and return the one with higher quality as the *base point* x_{Base} , and another as the *reference point* x_{Ref} .
- [2.1.3. Learning]: Combine x_{Base} and x_{Ref} to generate a new knowledge point $x_i(t + 1)$. Normally $x_i(t + 1)$ should be around x_{Base} , and the distance with x_{Base} is related to the distance between x_{Ref} and x_{Base} , and boundary handling mechanism should be incorporated here to ensure that $x_i(t + 1) \in S$.
- [2.1.4. Knowledge sharing]: Share a knowledge point, normally $x_i(t + 1)$, to the social sharing library X .
- [2.1.5. Individual update]: Update the private knowledge of agent i , normally replace $x_i(t)$ by $x_i(t + 1)$. Some Monte Carlo types might also be considered.
- [2.2. Library Maintenance]: The social sharing library using all knowledge points submitted by agents to update $X(t)$ into $X(t + 1)$. A simple way is one by one tournament selection: for each knowledge point submitted by an agent, replace the worse one among τ_W points randomly selected from $X(t)$.
- [3. Termination]: Return the best knowledge point found by the agents.

SCO has three main parameters, i.e., the number of agents N_c , the size of social sharing library N_L , and the learning cycle T . With the initialization process, the total number of knowledge points to be generated is $N_L + N_c * (T + 1)$, and is not related too much with N_L if T is large.

Compared to traditional swarm algorithms, e.g. **particle swarm optimization**, SCO can achieve high-quality solutions as N_c is small, even as $N_c = 1$. Nevertheless, smaller N_c and N_L might lead to **premature convergence**. Some variants^[5] were proposed to guarantee the global convergence.

138.2 References

- [1] Xie, Xiao-Feng; Zhang, Wen-Jun; Yang, Zhi-Lian (2002). Social cognitive optimization for nonlinear programming problems. *International Conference on Machine Learning and Cybernetics (ICMLC)*, Beijing, China: 779-783.
- [2] Xie, Xiao-Feng; Zhang, Wen-Jun (2004). Solving engineering design problems by social cognitive optimization. *Genetic and Evolutionary Computation Conference (GECCO)*, Seattle, WA, USA: 261-262.
- [3] Xu, Gang-Gang; Han, Luo-Cheng; Yu, Ming-Long; Zhang, Ai-Lan (2011). Reactive power optimization based on improved social cognitive optimization algorithm. *International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, Jilin, China: 97-100.
- [4] Fan, Caixia (2010). Solving integer programming based on maximum entropy social cognitive optimization algorithm. *International Conference on Information Technology and Scientific Management (ICITSM)*, Tianjing, China: 795-798.
- [5] Sun, Jia-ze; Wang, Shu-yan; chen, Hao (2014). A guarantee global convergence social cognitive optimizer. *Mathematical Problems in Engineering*: Art. No. 534162.

Chapter 139

Space allocation problem

The **Space allocation problem** (SAP) is the process in architecture or in any kind of Space Planning (SP) technique, of determining the position and size of several elements, according to the input specified design program requirements, mainly topological and geometric constraints, and the positioning of openings according to their geometric dimensions, in a two- or three-dimensional space.

Chapter 140

Space mapping

The space mapping methodology was first discovered by John Bandler in 1993. It uses relevant existing knowledge to speed up model generation and design optimization of a system. The knowledge is updated with new validation information from the system when available. It has wide application in modeling and optimization of engineering systems.

140.1 Concept

The space mapping methodology employs a “quasi-global” formulation that intelligently links companion “coarse” (ideal or low-fidelity) and “fine” (practical or high-fidelity) models of different complexities. In engineering design, space mapping aligns a very fast coarse model with the expensive-to-compute fine model so as to avoid direct expensive optimization of the fine model. The alignment can be done either off-line (model enhancement) or on-the-fly with surrogate updates (e.g., aggressive space mapping).

140.2 Development

Following John Bandler's concept in 1993,^{[1][2]} algorithms have utilized Broyden updates (aggressive space mapping),^[3] trust regions,^[4] and artificial neural networks. New developments include implicit space mapping, in which we allow preassigned parameters not used in the optimization process to change in the coarse model, and output space mapping, where a transformation is applied to the response of the model.^[5] A paper reviews the state of the art after the first ten years of development and implementation.^[6] Tuning space mapping utilizes a so-called tuning model—constructed invasively from the fine model—as well as a calibration process that translates the adjustment of the optimized tuning model parameters into relevant updates of the design variables.^[7]

140.3 Category

Space mapping optimization belongs to the class of surrogate-based optimization methods.^[8]

140.4 Terminology

There is a wide spectrum of terminology associated with space mapping: ideal model, coarse model, fine model, companion model, cheap model, expensive model, low fidelity (resolution) model, high fidelity (resolution) model, empirical model, simplified physics model, physics-based model, quasi-global model, physically expressive model, device under test, electromagnetics-based model, simulation model, computational model, tuning model, calibration model, surrogate model, surrogate update, mapped coarse model, surrogate optimization, parameter extraction, target response, optimization space, validation space, neuro-space mapping, implicit space mapping, output space mapping, predistortion (of design specifications), manifold mapping, defect correction, model management, multi-fidelity models, variable fidelity/variable complexity, multigrid methods, coarse grid, fine grid, surrogate-driven, simulation-driven, model-driven.

140.5 Methodology

At the core of the process is a pair of models: one very accurate but too expensive to use directly with a conventional optimization routine, and one significantly less expensive and, accordingly, less accurate. The latter is usually referred to as the “coarse” model. The former is usually referred to as the fine model. A validation space (“reality”) represents the fine model, for example, a high-fidelity physics model. The optimization space, where conventional optimization is carried out, incorporates the coarse (or surrogate) model, for example, the low-fidelity physics or “knowledge” model. In a space-mapping design optimization phase, there is a prediction or “execution” step, where the results of an optimized “mapped coarse model” (updated surrogate) are assigned

to the fine model for validation. After the validation process, if the design specifications are not satisfied, relevant data is transferred to the optimization space (“feedback”), where the mapping-augmented coarse model or surrogate is updated (enhanced, realigned with the fine model) through an iterative optimization process termed “parameter extraction.” The mapping formulation itself incorporates “intuition,” part of the engineer’s so-called “feel” for a problem.^[9]

140.6 Applications

Applications of space mapping continue to appear. Disciplines covered include microwaves, antennas, electronics, photonics, and magnetic systems; civil, mechanical, aeronautical and aerospace engineering systems, including

- Automotive crashworthiness design.^{[10][11]}
- EEG source analysis.^[12]
- Handset antenna design.^{[13][14]}
- Electric machines design and optimization.^[15]

140.7 Simulators

Various simulators can be involved in a space mapping optimization and modeling processes.

- In microwave and RF area
 - Agilent ADS
 - Agilent Momentum
 - Ansys HFSS
 - CST Microwave Studio
 - FEKO
 - Sonnet *em*

140.8 Nonlinear Device Modeling

The space mapping concept has been extended to neural-based space mapping for large-signal statistical modeling of nonlinear microwave devices.^{[16][17]}

140.9 Conferences

Three international workshops have focused significantly on the art, the science and the technology of space mapping.^{[18][19][20]}

140.10 See also

- Mathematical optimization
- Engineering optimization
- Computer-aided design
- Computational electromagnetics
- Semiconductor device modeling
- Multiphysics
- Simulation
- Adaptive control
- Performance tuning
- Linear approximation
- Response surface methodology
- Kriging
- Surrogate model
- Support vector machine

140.11 References

- [1] J.W. Bandler, “Have you ever wondered about the engineer’s mysterious ‘feel’ for a problem?” IEEE Canadian Review, no. 70, pp. 50-60, Summer 2013.
- [2] J.W. Bandler, R.M. Biernacki, S.H. Chen, P.A. Grobelny, and R.H. Hemmers, “Space mapping technique for electromagnetic optimization,” IEEE Trans. Microwave Theory Tech., vol. 42, no. 12, pp. 2536-2544, Dec. 1994.
- [3] J.W. Bandler, R.M. Biernacki, S.H. Chen, R.H. Hemmers, and K. Madsen, “Electromagnetic optimization exploiting aggressive space mapping,” IEEE Trans. Microwave Theory Tech., vol. 43, no. 12, pp. 2874-2882, Dec. 1995.
- [4] M.H. Bakr, J.W. Bandler, R.M. Biernacki, S.H. Chen and K. Madsen, “A trust region aggressive space mapping algorithm for EM optimization,” IEEE Trans. Microwave Theory Tech., vol. 46, no. 12, pp. 2412-2425, Dec. 1998.
- [5] J.W. Bandler, Q.S. Cheng, N.K. Nikolova and M.A. Ismail, “Implicit space mapping optimization exploiting preassigned parameters,” IEEE Trans. Microwave Theory Tech., vol. 52, no. 1, pp. 378-385, Jan. 2004.
- [6] J.W. Bandler, Q. Cheng, S.A. Dakrouby, A.S. Mohamed, M.H. Bakr, K. Madsen and J. Søndergaard, “Space mapping: the state of the art,” IEEE Trans. Microwave Theory Tech., vol. 52, no. 1, pp. 337-361, Jan. 2004.
- [7] S. Koziel, J. Meng, J.W. Bandler, M.H. Bakr, and Q.S. Cheng, “Accelerated microwave design optimization with tuning space mapping,” IEEE Trans. Microwave Theory Tech., vol. 57, no. 2, pp. 383-394, Feb. 2009.

- [8] A.J. Booker, J.E. Dennis, Jr., P.D. Frank, D.B. Serafini, V. Torczon, and M.W. Trosset, "A rigorous framework for optimization of expensive functions by surrogates," *Structural Optimization*, vol. 17, no. 1, pp. 1-13, Feb. 1999.
- [9] J.W. Bandler, "Have you ever wondered about the engineer's mysterious 'feel' for a problem?" The IEEE Canada McNaughton Lecture, IEEE Canadian Conference on Electrical and Computer Engineering (CCECE) (Montréal, Quebec, May 1, 2012).
- [10] M. Redhe and L. Nilsson, "Optimization of the new Saab 9-3 exposed to impact load using a space mapping technique," *Structural and Multidisciplinary Optimization*, vol. 27, no. 5, pp. 411-420, July 2004.
- [11] T. Jansson, L. Nilsson, and M. Redhe, "Using surrogate models and response surfaces in structural optimization—with application to crashworthiness design and sheet metal forming," *Structural and Multidisciplinary Optimization*, vol. 25, no.2, pp 129-140, July 2003.
- [12] G. Crevecoeur, H. Hallez, P. Van Heseb, Y. D'Asselerb, L. Dupréa, and R. Van de Walleb, "EEG source analysis using space mapping techniques," *Journal of Computational and Applied Mathematics*, vol. 215, no. 2, pp. 339-347, May 2008.
- [13] S. Tu, Q.S. Cheng, Y. Zhang, J.W. Bandler, and N.K. Nikolova, "Space mapping optimization of handset antennas exploiting thin-wire models," *IEEE Trans. Antennas Propag.*, vol. 61, no. 7, pp. 3797-3807, July 2013.
- [14] N. Friedrich, "Space mapping outpaces EM optimization in handset-antenna design," *microwaves&RF*, Aug. 30, 2013.
- [15] Khlissa, R.Vivier, S. ; Ospina Vargas, L.A. ; Friedrich, G. "Application of output Space Mapping method for fast optimization using multi-physical modeling"
- [16] L. Zhang, J. Xu, M.C.E. Yagoub, R. Ding, and Q.J. Zhang, "Efficient analytical formulation and sensitivity analysis of neuro-space mapping for nonlinear microwave device modeling," *IEEE Trans. Microwave Theory Tech.*, vol. 53, no. 9, pp. 2752-2767, Sep. 2005.
- [17] L. Zhang, Q.J. Zhang, and J. Wood, "Statistical neuro-space mapping technique for large-signal modeling of nonlinear devices," *IEEE Trans. Microwave Theory Tech.*, vol. 56, no. 11, pp. 2453-2467, Nov. 2008.
- [18] First Int. Workshop on Surrogate Modelling and Space Mapping for Engineering Optimization (Lyngby, Denmark, Nov. 2000).
- [19] Second Int. Workshop on Surrogate Modelling and Space Mapping for Engineering Optimization (Lyngby, Denmark, Nov. 2006).
- [20] Third Int. Workshop on Surrogate Modelling and Space Mapping for Engineering Optimization (Reykjavik, Iceland, Aug. 2012).

Chapter 141

Special ordered set

In discrete optimization, a **special ordered set** (SOS) is an ordered set of variables, used as an additional way to specify integrality conditions in an optimization model. Special order sets are basically a device or tool used in **branch and bound** methods for branching on sets of variables, rather than individual variables, as in ordinary mixed **integer programming**. Knowing that a variable is part of a **set** and that it is **ordered** gives the branch and bound Algorithm a more intelligent way to face the optimization problem, helping to speed up the search procedure. The members of a special ordered set individually may be continuous or discrete variables in any combination. However, even when all the members are themselves continuous, a model containing one or more special ordered sets becomes a discrete optimization problem requiring a **mixed integer optimizer** for its solution.

The ‘only’ benefit of using Special Ordered Sets compared with using only constraints, is that the search procedure will generally be noticeably faster.^[1]

As per J.A. Tomlin,^[2] Special Order Sets provide a powerful means of modeling nonconvex functions and discrete requirements, though there has been a tendency to think of them only in terms of multiple-choice zero-one programming.

141.1 Context of Applications

- Multiple-choice programming
- Global Optimization with continuous separable functions.

141.2 History

The origin of the concept was in the paper of Beale titled “Two transportation problems” (1963)^[3] where he presented a bid evaluation model, however, the term was first explicitly introduced by Beale and Tomlin (1970).^[4] The special order set were first implemented in Scicon’s UMPIRE mathematical programming system.^[5] Special Order sets were an important and recurring theme

in Martin Beale’s work,^[6] and their value came to be recognized to the point where nearly all production mathematical programming systems (MPS’s) implement some version, or subset, of SOS.

141.3 Types of SOS

There are two sorts of Special Ordered Sets:

1. **Special Ordered Sets of type 1 (SOS1 or S1):** are a set of variables, at most **one** of which can take a strictly positive value, all others being at 0. They most frequently apply where a set of variables are actually 0-1 variables: in other words, we have to choose one from a set of possibilities. These might arise for instance where we are deciding on what size of factory to build, when we have a set of options, perhaps small, medium, large or no factory at all, and we have to choose one and only one size.
2. **Special Ordered Sets of type 2 (SOS2 or S2):** an ordered set of non-negative variables, of which at most **two** can be non-zero, and if two are non-zero these must be consecutive in their ordering. Special Ordered Sets of type 2 are typically used to model non-linear functions of a variable in a linear model. They are the natural extension of the concepts of Separable Programming, but when embedded in a Branch and Bound code enable truly global optima to be found, and not just local optima.

141.4 Further Example

- Example sizing a warehouse (Example for SOS Type 1 from Manual of ILOG CPLEX 11.2)

141.5 Notes

[1] Christelle Gueret, Christian Prins, Marc Sevaux, “Applications of optimization with Xpress-MP”, Editions Eyrolles, Paris, France (2000), ISBN 0-9543503-0-8, pag 39-42 [Link to PDF](#)

- [2] J.A. Tomlin, “Special Ordered Sets and an Application to Gas Supply Operations Planning”, Ketron Management Science, Inc., Mountain View, CA 94040-1266, USA
- [3] E.M.L. Beale, “Two transportation problems”, in: G.Kreweras and G.Morlat, eds., “Proceeding of the Third International Conference on Operational Research” (Dunod, Paris and English Universities Press, London, 1963) 780-788
- [4] E.M.L. Beale and J.A. Tomlin, “Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables”, in: J.Lawrence, ed., “Proceedings of the Fifth International Conference on Operational Research” (Tavistock Publications, London, 1970) 447-454
- [5] J.J.H. Forrest, J.P.H Hirst and J.A. Tomlin, “Practical solution of large mixed integer programming problems with UMPIRE”, Management Sci. 20 (1974) 736-773
- [6] M.J.D. Powell, “A biographical memoir of Evelyn Martin Lansdowne Beale, FRS”, Biographical Memoirs of Fellows of the Royal Society 33 (1987)

141.6 References

- The notion of Special Ordered Sets was introduced by E. M. L. Beale and J. A. Tomlin. Special Facilities in a General Mathematical Programming System for Nonconvex Problems Using Ordered Sets of Variables. In J. Lawrence, editor, Operational Research 69, pages 447–454. Tavistock Publishing, London, 1970.
- E. M. L. Beale and J. J. H. Forrest. Global Optimization Using Special Ordered Sets. Mathematical Programming, 10(1):52–69, 1976.
- “Special Ordered Sets and an Application to Gas Supply Operations Planning”, J.A. Tomlin, Ketron Management Science, Inc., Mountain View, CA 94040-1266, USA
- Christelle Gueret, Christian Prins, Marc Sevaus, “Applications of optimization with Xpress-MP”, Editions Eyrolles, Paris, France (2000), ISBN 0-9543503-0-8, pag 39-42

Chapter 142

Stochastic hill climbing

Stochastic hill climbing is a variant of the basic **hill climbing** method. While basic hill climbing always chooses the steepest uphill move, “stochastic hill climbing chooses at random from among the uphill moves; the probability of selection can vary with the steepness of the uphill move.”^[1]

142.1 See also

- Stochastic gradient descent

142.2 References

[1] Russell, Stuart and Norvig, Peter. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 2009

Chapter 143

Swarm intelligence

Swarm intelligence (SI) is the collective behavior of decentralized, self-organized systems, natural or artificial. The concept is employed in work on **artificial intelligence**. The expression was introduced by Gerardo Beni and Jing Wang in 1989, in the context of cellular robotic systems.^[1]

SI systems consist typically of a population of simple agents or **boids** interacting locally with one another and with their environment. The inspiration often comes from nature, especially biological systems. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents lead to the **emergence** of "intelligent" global behavior, unknown to the individual agents. Examples in natural systems of SI include ant colonies, bird flocking, animal herding, bacterial growth, fish schooling and **Microbial intelligence**. The definition of swarm intelligence is still not quite clear. In principle, it should be a multi-agent system that has self-organized behaviour that shows some intelligent behaviour.

The application of swarm principles to **robots** is called **swarm robotics**, while 'swarm intelligence' refers to the more general set of algorithms. 'Swarm prediction' has been used in the context of forecasting problems.

143.1 Example algorithms

143.1.1 Particle swarm optimization

Particle swarm optimization (PSO) is a **global optimization** algorithm for dealing with problems in which a best solution can be represented as a point or surface in an n-dimensional space. Hypotheses are plotted in this space and seeded with an initial **velocity**, as well as a communication channel between the particles.^{[2][3]} Particles then move through the solution space, and are evaluated according to some **fitness** criterion after each timestep. Over time, particles are accelerated towards those particles within their communication grouping which have better fitness values. The main advantage of such an approach over other global minimization strategies such as

simulated annealing is that the large number of members that make up the particle swarm make the technique impressively resilient to the problem of **local minima**.

143.1.2 Ant colony optimization

Ant colony optimization (ACO), introduced by Dorigo in his doctoral dissertation, is a class of **optimization algorithms** modeled on the actions of an **ant colony**. ACO is a **probabilistic technique** useful in problems that deal with finding better paths through graphs. Artificial 'ants'—simulation agents—locate optimal solutions by moving through a **parameter space** representing all possible solutions. Natural ants lay down **pheromones** directing each other to resources while exploring their environment. The simulated 'ants' similarly record their positions and the quality of their solutions, so that in later simulation iterations more ants locate better solutions.^[4]

143.1.3 Ant Lion Optimizer

The **Ant Lion Optimizer** (ALO) mimics the hunting mechanism of antlions in nature. Five main steps of hunting prey such as the random walk of ants, building traps, entrapment of ants in traps, catching preys, and re-building traps are implemented in this algorithm. This algorithm was propose in 2015^[5] (also see **Grey wolf optimizer**).

143.1.4 Artificial bee colony algorithm

Artificial bee colony algorithm (ABC) is a meta-heuristic algorithm introduced by Karaboga in 2005,^[6] and simulates the foraging behaviour of honey bees. The ABC algorithm has three phases: employed bee, onlooker bee and scout bee. In the employed bee and the onlooker bee phases, bees exploit the sources by local searches in the neighbourhood of the solutions selected based on deterministic selection in the employed bee phase and the **probabilistic selection** in the onlooker bee phase. In the scout bee phase which is an analogy of abandoning exhausted food sources in the foraging process, solutions that are not beneficial anymore for search progress

are abandoned, and new solutions are inserted instead of them to explore new regions in the search space. The algorithm has a well-balanced exploration and exploitation ability.

143.1.5 Bacterial colony optimization

The algorithm is based on a lifecycle model that simulates some typical behaviors of *E. coli* bacteria during their whole lifecycle, including chemotaxis, communication, elimination, reproduction, and migration.^[7]

Bacteria communication and self-organization in the context of Network theory has been investigated by Eshel Ben-Jacob research group at Tel Aviv University which developed a fractal model of bacterial colony and identified linguistic and social patterns in colony lifecycle^[8] (also see Ben-Jacob's bacteria, Microbial intelligence and Microbial cooperation).

143.1.6 Differential evolution

Differential evolution is similar to genetic algorithm and pattern search. It uses multiagents or search vectors to carry out search. It has mutation and crossover, but do not have the global best solution in its search equations, in contrast with the particle swarm optimization.

143.1.7 The bees algorithm

The bees algorithm in its basic formulation was created by Pham and his co-workers in 2005,^[9] and further refined in the following years.^[10] Modelled on the foraging behaviour of honey bees, the algorithm combines global explorative search with local exploitative search. A small number of artificial bees (scouts) explores randomly the solution space (environment) for solutions of high fitness (highly profitable food sources), whilst the bulk of the population search (harvest) the neighbourhood of the fittest solutions looking for the fitness optimum. A deterministics recruitment procedure which simulates the waggle dance of biological bees is used to communicate the scouts' findings to the foragers, and distribute the foragers depending on the fitness of the neighbourhoods selected for local search. Once the search in the neighbourhood of a solution stagnates, the local fitness optimum is considered to be found, and the site is abandoned. In summary, the Bees Algorithm searches concurrently the most promising regions of the solution space, whilst continuously sampling it in search of new favourable regions.

143.1.8 Artificial immune systems

Artificial immune systems (AIS) concerns the usage of abstract structure and function of the immune system to computational systems, and investigating the application

of these systems towards solving computational problems from mathematics, engineering, and information technology. AIS is a sub-field of Biologically inspired computing, and natural computation, with interests in Machine Learning and belonging to the broader field of Artificial Intelligence.

143.1.9 Grey wolf optimizer

The Grey wolf optimizer (GWO) algorithm mimics the leadership hierarchy and hunting mechanism of gray wolves in nature proposed by Mirjalili et al. in 2014.^[11] Four types of grey wolves such as alpha, beta, delta, and omega are employed for simulating the leadership hierarchy. In addition, three main steps of hunting, searching for prey, encircling prey, and attacking prey, are implemented to perform optimization.

143.1.10 Bat algorithm

Bat algorithm (BA) is a swarm-intelligence-based algorithm, inspired by the echolocation behavior of microbats. BA uses a frequency-tuning and automatic balance of exploration and exploitation by controlling loudness and pulse emission rates.^[12]

143.1.11 Multi-Verse Optimizer

The main inspirations of the Multi-Verse Optimizer (MVO) are based on three concepts in cosmology: white hole, black hole, and wormhole. The mathematical models of these three concepts are developed to perform exploration, exploitation, and local search respectively.^[13]

143.1.12 Gravitational search algorithm

Gravitational search algorithm (GSA) based on the law of gravity and the notion of mass interactions. The GSA algorithm uses the theory of Newtonian physics and its searcher agents are the collection of masses. In GSA, there is an isolated system of masses. Using the gravitational force, every mass in the system can see the situation of other masses. The gravitational force is therefore a way of transferring information between different masses (Rashedi, Nezamabadi-pour and Saryazdi 2009).^[14] In GSA, agents are considered as objects and their performance is measured by their masses. All these objects attract each other by a gravity force, and this force causes a movement of all objects globally towards the objects with heavier masses. The heavy masses correspond to good solutions of the problem. The position of the agent corresponds to a solution of the problem, and its mass is determined using a fitness function. By lapse of time, masses are attracted by the heaviest mass, which would ideally present an optimum solution in the

search space. The GSA could be considered as an isolated system of masses. It is like a small artificial world of masses obeying the Newtonian laws of gravitation and motion (Rashedi, Nezamabadi-pour and Saryazdi 2009). A multi-objective variant of GSA, called Non-dominated Sorting Gravitational Search Algorithm (NSGSA), was proposed by Nobahari and Nikusokhan in 2011.^[15]

143.1.13 Altruism algorithm

Researchers in Switzerland have developed an algorithm based on Hamilton's rule of kin selection. This algorithm shows how altruism in a swarm of entities can, over time, evolve and result in more effective swarm behaviour.^{[16][17]}

143.1.14 Glowworm swarm optimization

Glowworm swarm optimization (GSO), introduced by Krishnanand and Ghose in 2005 for simultaneous computation of multiple optima of multimodal functions.^{[18][19][20][21]} The algorithm shares a few features with some better known algorithms, such as ant colony optimization and particle swarm optimization, but with several significant differences. The agents in GSO are thought of as glowworms that carry a luminescence quantity called luciferin along with them. The glowworms encode the fitness of their current locations, evaluated using the objective function, into a luciferin value that they broadcast to their neighbors. The glowworm identifies its neighbors and computes its movements by exploiting an adaptive neighborhood, which is bounded above by its sensor range. Each glowworm selects, using a probabilistic mechanism, a neighbor that has a luciferin value higher than its own and moves toward it. These movements—based only on local information and selective neighbor interactions—enable the swarm of glowworms to partition into disjoint subgroups that converge on multiple optima of a given multimodal function.

143.1.15 River Formation Dynamics

River Formation Dynamics (RFD) is based on imitating how water forms rivers by eroding the ground and depositing sediments. After drops transform the landscape by increasing/decreasing the altitude of places, solutions are given in the form of paths of decreasing altitudes. Decreasing gradients are constructed, and these gradients are followed by subsequent drops to compose new gradients and reinforce the best ones. This heuristic optimization method was first presented in 2007 by Rabanal et al.^[22] The applicability of RFD to other NP-complete problems was studied in,^{[23][24][25]} Some other authors have also applied RFD in robot navigation^[26] or in routing protocols.^[27]

143.1.16 Self-propelled particles

Self-propelled particles (SPP), also referred to as the Vicsek model, was introduced in 1995 by Vicsek *et al.*^[28] as a special case of the boids model introduced in 1986 by Reynolds.^[29] A swarm is modelled in SPP by a collection of particles that move with a constant speed but respond to a random perturbation by adopting at each time increment the average direction of motion of the other particles in their local neighbourhood.^[30] SPP models predict that swarming animals share certain properties at the group level, regardless of the type of animals in the swarm.^[31] Swarming systems give rise to emergent behaviours which occur at many different scales, some of which are turning out to be both universal and robust. It has become a challenge in theoretical physics to find minimal statistical models that capture these behaviours.^{[32][33][34]}

143.1.17 Stochastic diffusion search

Stochastic diffusion search (SDS)^{[35][36]} is an agent-based probabilistic global search and optimization technique best suited to problems where the objective function can be decomposed into multiple independent partial-functions. Each agent maintains a hypothesis which is iteratively tested by evaluating a randomly selected partial objective function parameterised by the agent's current hypothesis. In the standard version of SDS such partial function evaluations are binary, resulting in each agent becoming active or inactive. Information on hypotheses is diffused across the population via inter-agent communication. Unlike the stigmergic communication used in ACO, in SDS agents communicate hypotheses via a one-to-one communication strategy analogous to the tandem running procedure observed in *Leptothorax acervorum*.^[37] A positive feedback mechanism ensures that, over time, a population of agents stabilise around the global-best solution. SDS is both an efficient and robust global search and optimisation algorithm, which has been extensively mathematically described.^{[38][39][40]} Recent work has involved merging the global search properties of SDS with other swarm intelligence algorithms.^{[41][42]}

143.1.18 Multi-swarm optimization

Multi-swarm optimization is a variant of particle swarm optimization (PSO) based on the use of multiple subswarms instead of one (standard) swarm. The general approach in multi-swarm optimization is that each subswarm focuses on a specific region while a specific diversification method decides where and when to launch the sub-swarms. The multi-swarm framework is especially fitted for the optimization on multi-modal problems, where multiple (local) optima exist.

143.2 Applications

Swarm Intelligence-based techniques can be used in a number of applications. The U.S. military is investigating swarm techniques for controlling unmanned vehicles. The European Space Agency is thinking about an orbital swarm for self-assembly and interferometry. NASA is investigating the use of swarm technology for planetary mapping. A 1992 paper by M. Anthony Lewis and George A. Bekey discusses the possibility of using swarm intelligence to control nanobots within the body for the purpose of killing cancer tumors.^[43] Conversely al-Rifaie and Aber have used Stochastic Diffusion Search to help locate tumours.^{[44][45]} Swarm intelligence has also been applied for data mining.^[46]

143.2.1 Ant-based routing

The use of Swarm Intelligence in telecommunication networks has also been researched, in the form of ant-based routing. This was pioneered separately by Dorigo et al. and Hewlett Packard in the mid-1990s, with a number of variations since. Basically this uses a probabilistic routing table rewarding/reinforcing the route successfully traversed by each “ant” (a small control packet) which flood the network. Reinforcement of the route in the forwards, reverse direction and both simultaneously have been researched: backwards reinforcement requires a symmetric network and couples the two directions together; forwards reinforcement rewards a route before the outcome is known (but then you pay for the cinema before you know how good the film is). As the system behaves stochastically and is therefore lacking repeatability, there are large hurdles to commercial deployment. Mobile media and new technologies have the potential to change the threshold for collective action due to swarm intelligence (Rheingold: 2002, P175).

The location of transmission infrastructure for wireless communication networks is an important engineering problem involving competing objectives. A minimal selection of locations (or sites) are required subject to providing adequate area coverage for users. A very different ant inspired swarm intelligence algorithm, stochastic diffusion search (SDS), has been successfully used to provide a general model for this problem, related to circle packing and set covering. It has been shown that the SDS can be applied to identify suitable solutions even for large problem instances.^[47]

Airlines have also used ant-based routing in assigning aircraft arrivals to airport gates. At Southwest Airlines a software program uses swarm theory, or swarm intelligence—the idea that a colony of ants works better than one alone. Each pilot acts like an ant searching for the best airport gate. “The pilot learns from his experience what’s the best for him, and it turns out that that’s the best solution for the airline,” Douglas A. Lawson ex-

plains. As a result, the “colony” of pilots always go to gates they can arrive at and depart from quickly. The program can even alert a pilot of plane back-ups before they happen. “We can anticipate that it’s going to happen, so we’ll have a gate available,” Lawson says.^[48]

143.2.2 Crowd simulation

Artists are using swarm technology as a means of creating complex interactive systems or simulating crowds.

Stanley and Stella in: *Breaking the Ice* was the first movie to make use of swarm technology for rendering, realistically depicting the movements of groups of fish and birds using the *Boids* system. Tim Burton’s *Batman Returns* also made use of swarm technology for showing the movements of a group of bats. The *Lord of the Rings* film trilogy made use of similar technology, known as *Massive*, during battle scenes. Swarm technology is particularly attractive because it is cheap, robust, and simple. Airlines have used swarm theory to simulate passengers boarding a plane. Southwest Airlines researcher Douglas A. Lawson used an ant-based computer simulation employing only six interaction rules to evaluate boarding times using various boarding methods.(Miller, 2010, xii-xviii).^[49]

143.2.3 Swarmic art

In a series of works al-Rifaie et al.^[50] have successfully used two swarm intelligence algorithms – one mimicking the behaviour of one species of ants (*Leptothorax acervorum*) foraging (stochastic diffusion search, SDS) and the other algorithm mimicking the behaviour of birds flocking (particle swarm optimization, PSO) – to describe a novel integration strategy exploiting the local search properties of the PSO with global SDS behaviour. The resulting hybrid algorithm is used to sketch novel drawings of an input image, exploiting an artistic tension between the local behaviour of the ‘birds flocking’ - as they seek to follow the input sketch - and the global behaviour of the “ants foraging” - as they seek to encourage the flock to explore novel regions of the canvas. The “creativity” of this hybrid swarm system has been analysed under the philosophical light of the “rhizome” in the context of Deleuze’s “Orchid and Wasp” metaphor.^[51]

In a more recent work of al-Rifaie et al., “Swarmic Sketches and Attention Mechanism”,^[52] introduces a novel approach deploying the mechanism of ‘attention’ by adapting SDS to selectively attend to detailed areas of a digital canvas. Once the attention of the swarm is drawn to a certain line within the canvas, the capability of PSO is used to produce a ‘swarmic sketch’ of the attended line. The swarms move throughout the digital canvas in an attempt to satisfy their dynamic roles – attention to areas with more details – associated to them via their fitness

function. Having associated the rendering process with the concepts of attention, the performance of the participating swarms creates a unique, non-identical sketch each time the ‘artist’ swarms embark on interpreting the input line drawings. In other works while PSO is responsible for the sketching process, SDS controls the attention of the swarm.

In a similar work, “Swarmic Paintings and Colour Attention”,^[53] non-photorealistic images are produced using SDS algorithm which, in the context of this work, is responsible for colour attention.

The “computational creativity” of the above-mentioned systems are discussed in^{[50][54][55][56]} through the two prerequisites of creativity (i.e. freedom and constraints) within the swarm intelligence’s two infamous phases of exploration and exploitation.

143.3 In popular culture

Main article: Group mind (science fiction)

Swarm intelligence-related concepts and references can be found throughout popular culture, frequently as some form of collective intelligence or group mind involving far more agents than used in current applications.

- Science fiction writer Olaf Stapledon may have been the first to discuss swarm intelligences equal or superior to humanity. In *Last and First Men* (1931), a swarm intelligence from Mars consists of tiny individual cells that communicate with each other by radio waves; in *Star Maker* (1937) swarm intelligences founded numerous civilizations.
- *The Invincible* (1964), a science fiction novel by Stanisław Lem where a human spaceship finds intelligent behavior in a flock of small particles that were able to defend themselves against what they found as a menace.
- In the dramatic novel and subsequent mini-series *The Andromeda Strain* (1969) by Michael Crichton, an extraterrestrial virus communicates between individual cells and displays the ability to think and react individually and as a whole, and as such displays a semblance of “swarm intelligence”.
- Ygramul, the Many - an intelligent being consisting of a swarm of many wasp-like insects, a character in the novel *The Neverending Story* (1979) written by Michael Ende. Ygramul is also mentioned in a scientific paper, “Flocks, Herds, and Schools” written by Knut Hartmann (Computer Graphics and Interactive Systems, Otto-von-Guericke University of Magdeburg).^[57]
- *Swarm* (1982), a short story by Bruce Sterling about a mission undertaken by a faction of humans, to understand and exploit a space-faring swarm intelligence.
- In the book *Ender’s Game* (1985), the Formics (known popularly as *Buggers*) are a swarm intelligence with colonies or armadas each directed by a single queen.
- In *Star Trek: The Next Generation* (1989) and *Star Trek: Voyager* (1995), the Borg are a hierarchical swarm intelligence that assimilates humanoid species to grow.
- *The Hacker and the Ants* (1994), a book by Rudy Rucker on AI ants within a virtual environment.
- *Hallucination* (1995), a posthumously-published short story by Isaac Asimov about an alien insect-like swarm, capable of organization and provided with a sort of swarm intelligence.
- The Zerg (1998) of the Starcraft universe demonstrate such concepts when in groups and enhanced by the psychic control of taskmaster breeds.
- *Decipher* (2001) by Stel Pavlou deals with the swarm intelligence of nanobots that guard against intruders in Atlantis.
- In the video game series *Halo*, the Covenant (2001) species known as the Hunters are made up of thousands of worm-like creatures which are individually non-sentient, but, collectively form a sentient being.
- *Prey* (2002), by Michael Crichton deals with the danger of nanobots escaping from human control and developing a swarm intelligence.
- In the *Legends of Dune* series (2002), Omnis becomes a swarm intelligence by taking over almost all of the artificial intelligence that exists in the universe
- The science fiction novel *The Swarm* (2004), by Frank Schätzing, deals with underwater single-celled creatures who act in unison to destroy humanity.
- In the video game *Mass Effect* (2007), a galactic race known as the Quarians created a race of humanoid machines known as the Geth which worked as a swarm intelligence in order to avoid restrictions on true-AI. However the Geth obtained a shared sentience through the combined processing power of every geth unit.
- In *Sandworms of Dune* (2007), the Face Dancers are revealed to have developed into a swarm intelligence represented by Khrone

- In the video game *Penumbra: Black Plague* (2008), the Tuurngait is a hivemind that grows by infecting other organisms with a virus.
- *Kill Decision* (2012), a novel by Daniel Suarez features autonomous drones programmed with the aggressive swarming intelligence of Weaver ants.^[58]

143.4 Notable researchers

- Marco Dorigo
- Russell C. Eberhart
- Luca Maria Gambardella
- James Kennedy

143.5 See also

- Cellular automaton
- Reinforcement learning
- Stochastic optimization
- Differential evolution
- Evolutionary computation
- Global brain
- Harmony search
- Metaheuristic
- Promise theory
- Quorum sensing
- Rule 110
- Swarm Development Group
- Swarming
- SwisTrack
- Stochastic search
- *The Wisdom of Crowds*
- Wisdom of the crowd
- Symmetry breaking of escaping ants
- Microbial Intelligence
-
- Evolutionary algorithm
- Self-organization
- Multi-agent system
- Myrmecology

143.6 References

- [1] Beni, G., Wang, J. Swarm Intelligence in Cellular Robotic Systems, Proceed. NATO Advanced Workshop on Robots and Biological Systems, Tuscany, Italy, June 26–30 (1989)
- [2] Parsopoulos, K. E.; Vrahatis, M. N. (2002). “Recent Approaches to Global Optimization Problems Through Particle Swarm Optimization”. *Natural Computing* **1** (2-3): 235–306. doi:10.1023/A:1016568309421.
- [3] Particle Swarm Optimization by Maurice Clerc, ISTE, ISBN 1-905209-04-5, 2006.
- [4] Ant Colony Optimization by Marco Dorigo and Thomas Stützle, MIT Press, 2004. ISBN 0-262-04219-3
- [5] Seyedali Mirjalili (2015). “The Ant Lion Optimizer”. *Advances in Engineering Software* **83**:: 80–98.
- [6] Karaboga, Dervis (2010). “Artificial bee colony algorithm”. *Scholarpedia* **5** (3): 6915. doi:10.4249/scholarpedia.6915.
- [7] Niu, Ben (2012). “Bacterial colony optimization”. *Discrete Dynamics in Nature and Society*. Volume 2012 (2012) (Article ID 698057).
- [8] Cohen, Inon et al. (1999). “Continuous and discrete models of cooperation in complex bacterial colonies”. *Fractals*. 7.03 (1999):: 235–247.
- [9] Pham DT, Ghanbarzadeh A, Koc E, Otri S, Rahim S and Zaidi M. The Bees Algorithm. Technical Note, Manufacturing Engineering Centre, Cardiff University, UK, 2005.
- [10] Pham, D.T., Castellani, M. (2009), The Bees Algorithm – Modelling Foraging Behaviour to Solve Continuous Optimisation Problems. Proc. ImechE, Part C, 223(12), 2919-2938.
- [11] S. Mirjalili, S. M. Mirjalili, and A. Lewis, “Grey Wolf Optimizer,” Advances in Engineering Software, vol. 69, pp. 46-61, 2014.
- [12] X. S. Yang, A New Metaheuristic Bat-Inspired Algorithm, in: Nature Inspired Cooperative Strategies for Optimization (NISCO 2010) (Eds. J. R. Gonzalez et al.), Studies in Computational Intelligence, Springer Berlin, 284, Springer, 65-74 (2010).
- [13] Mirjalili, S.; Mirjalili, S. M.; Hatamlou, A. (2015). “Multi-Verse Optimizer: a nature-inspired algorithm for global optimization”. doi:10.1007/s00521-015-1870-7.
- [14] Rashedi, E.; Nezamabadi-pour, H.; Saryazdi, S. (2009). “GSA: a gravitational search algorithm”. *Information Science* **179** (13): 2232–2248. doi:10.1016/j.ins.2009.03.004.
- [15] Nobahari, H.; Nikusokhan, M. “Non-dominated Sorting Gravitational Search Algorithm”. *International Conference on Swarm Intelligence*.
- [16] Altruism helps swarming robots fly better genevalunch.com, 4 May 2011.

- [17] Waibel, M; Floreano, D; Keller, L (2011). “A quantitative test of Hamilton’s rule for the evolution of altruism”. *PLoS Biology* **9** (5): e1000615. doi:10.1371/journal.pbio.1000615.
- [18] Krishnanand K.N. and D. Ghose (2005) “Detection of multiple source locations using a glowworm metaphor with applications to collective robotics”. *IEEE Swarm Intelligence Symposium*, Pasadena, California, USA, pp. 84–91.
- [19] Krishnanand, K.N.; Ghose, D. (2009). “Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions”. *Swarm Intelligence* **3** (2): 87–124. doi:10.1007/s11721-008-0021-5.
- [20] Krishnanand, K.N.; Ghose, D. (2008). “Theoretical foundations for rendezvous of glowworm-inspired agent swarms at multiple locations”. *Robotics and Autonomous Systems* **56** (7): 549–569. doi:10.1016/j.robot.2007.11.003.
- [21] Krishnanand, K.N.; Ghose, D. (2006). “Glowworm swarm based optimization algorithm for multimodal functions with collective robotics applications”. *Multi-agent and Grid Systems* **2** (3): 209–222.
- [22] P. Rabanal, I. Rodríguez, and F. Rubio (2007) “Using River Formation Dynamics to Design Heuristic Algorithms”. *Unconventional Computation*, Springer, LNCS 4616, pp. 163–177.
- [23] P. Rabanal, I. Rodríguez, and F. Rubio (2008) “Finding minimum spanning/distances trees by using river formation dynamics”. *Ant Colony Optimization and Swarm Intelligence*, Springer, LNCS 5217, pp. 60–71.
- [24] P. Rabanal, I. Rodríguez, and F. Rubio (2009) “Applying River Formation Dynamics to Solve NP-Complete Problems”. *Nature-Inspired Algorithms for Optimisation*, Springer, SCI 193, pp. 333–368.
- [25] P. Rabanal, I. Rodríguez, and F. Rubio (2013) “Testing restorable systems: formal definition and heuristic solution based on river formation dynamics”. *Formal Aspects of Computing*, Springer, Volume 25, Number 5, pp. 743–768.
- [26] G. Redlarski, A. Palkowski, M. Dąbkowski (2013) “Using River Formation Dynamics Algorithm in Mobile Robot Navigation”. *Solid State Phenomena*, Volume 198, pp. 138–143.
- [27] S. Hameed-Amin, H.S. Al-Raweshidy, R. Sabbar-Abbas (2014) “Smart data packet ad hoc routing protocol”. *Computer Networks*, Volume 62, pp. 162–181.
- [28] Vicsek, T.; Czirok, A.; Ben-Jacob, E.; Cohen, I.; Shochet, O. (1995). “Novel type of phase transition in a system of self-driven particles”. *Physical Review Letters* **75**: 1226–1229. arXiv:cond-mat/0611743. Bibcode:1995PhRvL..75.1226V. doi:10.1103/PhysRevLett.75.1226. PMID 10060237.
- [29] Reynolds, C. W. (1987). “Flocks, herds and schools: A distributed behavioral model”. *Computer Graphics* **21** (4): 25–34. doi:10.1145/37401.37406. CiteSeerX: 10.1.1.103.7187.
- [30] Czirók, A.; Vicsek, T. (2006). “Collective behavior of interacting self-propelled particles”. *Physica A* **281**: 17–29. arXiv:cond-mat/0611742. Bibcode:2000PhyA..281...17C. doi:10.1016/S0378-4371(00)00013-3.
- [31] Buhl, J.; Sumpter, D.J.T.; Couzin, D.; Hale, J.J.; Despland, E.; Miller, E.R.; Simpson, S.J. et al. (2006). “From disorder to order in marching locusts” (PDF). *Science* **312** (5778): 1402–1406. Bibcode:2006Sci...312.1402B. doi:10.1126/science.1125142. PMID 16741126.
- [32] Toner, J.; Tu, Y.; Ramaswamy, S. (2005). “Hydrodynamics and phases of flocks” (PDF). *Annals of Physics* **318**: 170. Bibcode:2005AnPhy.318..170T. doi:10.1016/j.aop.2005.04.011.
- [33] Bertin, E.; Droz, M.; Grégoire, G. (2009). “Hydrodynamic equations for self-propelled particles: microscopic derivation and stability analysis”. *J. Phys. A* **42** (44): 445001. arXiv:0907.4688. Bibcode:2009JPhA...42R5001B. doi:10.1088/1751-8113/42/44/445001.
- [34] Li, Y.X.; Lukeman, R.; Edelstein-Keshet, L. et al. (2007). “Minimal mechanisms for school formation in self-propelled particles” (PDF). *Physica D: Nonlinear Phenomena* **237** (5): 699–720. Bibcode:2008PhyD..237..699L. doi:10.1016/j.physd.2007.10.009.
- [35] Bishop, J.M., Stochastic Searching Networks, Proc. 1st IEE Int. Conf. on Artificial Neural Networks, pp. 329–331, London, UK, (1989).
- [36] Nasuto, S.J. & Bishop, J.M., (2008), Stabilizing swarm intelligence search via positive feedback resource allocation, In: Krasnogor, N., Nicosia, G, Pavone, M., & Pelta, D. (eds), Nature Inspired Cooperative Strategies for Optimization, Studies in Computational Intelligence, vol 129, Springer, Berlin, Heidelberg, New York, pp. 115-123.
- [37] Moglich, M.; Maschwitz, U.; Holldobler, B., Tandem Calling: A New Kind of Signal in Ant Communication, Science, Volume 186, Issue 4168, pp. 1046-1047
- [38] Nasuto, S.J., Bishop, J.M. & Lauria, S., Time complexity analysis of the Stochastic Diffusion Search, Proc. Neural Computation '98, pp. 260-266, Vienna, Austria, (1998).
- [39] Nasuto, S.J., & Bishop, J.M., (1999), Convergence of the Stochastic Diffusion Search, Parallel Algorithms, 14:2, pp: 89-107.
- [40] Myatt, D.M., Bishop, J.M., Nasuto, S.J., (2004), Minimum stable convergence criteria for Stochastic Diffusion Search, Electronics Letters, 22:40, pp. 112-113.
- [41] al-Rifaie, M.M., Bishop, J.M. & Blackwell, T., An investigation into the merger of stochastic diffusion search and particle swarm optimisation, Proc. 13th Conf. Genetic and Evolutionary Computation, (GECCO), pp.37-44, (2012).
- [42] al-Rifaie, Mohammad Majid, John Mark Bishop, and Tim Blackwell. “Information sharing impact of stochastic diffusion search on differential evolution algorithm.” *Memetic Computing* 4.4 (2012): 327-338.

- [43] Lewis, M. Anthony; Bekey, George A. "The Behavioral Self-Organization of Nanorobots Using Local Rules". *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- [44] Identifying metastasis in bone scans with Stochastic Diffusion Search, al-Rifaie, M.M. & Aber, A., Proc. IEEE Information Technology in Medicine and Education, ITME 2012, pp. 519-523.
- [45] al-Rifaie, Mohammad Majid, Ahmed Aber, and Ahmed Majid Oudah. "Utilising Stochastic Diffusion Search to identify metastasis in bone scans and microcalcifications on mammograms." In Bioinformatics and Biomedicine Workshops (BIBMW), 2012 IEEE International Conference on, pp. 280-287. IEEE, 2012.
- [46] Martens, D.; Baesens, B.; Fawcett, T. (2011). "Editorial Survey: Swarm Intelligence for Data Mining". *Machine Learning* **82** (1): 1–42. doi:10.1007/s10994-010-5216-5.
- [47] Whitaker, R.M., Hurley, S.. An agent based approach to site selection for wireless networks. Proc ACM Symposium on Applied Computing, pp. 574–577, (2002).
- [48] "Planes, Trains and Ant Hills: Computer scientists simulate activity of ants to reduce airline delays". *Science Daily*. April 1, 2008. Retrieved December 1, 2010.
- [49] Miller, Peter (2010). *The Smart Swarm: How understanding flocks, schools, and colonies can make us better at communicating, decision making, and getting things done*. New York: Avery. ISBN 978-1-58333-390-7.
- [50] al-Rifaie, MM, Bishop, J.M., & Caines, S., Creativity and Autonomy in Swarm Intelligence Systems, Cognitive Computing 4:3, pp. 320-331, (2012).
- [51] Deleuze G, Guattari F, Massumi B. *A thousand plateaus*. Minneapolis: University of Minnesota Press; 2004.
- [52] al-Rifaie, Mohammad Majid, and John Mark Bishop. "Swarmic sketches and attention mechanism". Evolutionary and Biologically Inspired Music, Sound, Art and Design. Springer Berlin Heidelberg, 2013. 85-96.
- [53] al-Rifaie, Mohammad Majid, and John Mark Bishop. "Swarmic paintings and colour attention". Evolutionary and Biologically Inspired Music, Sound, Art and Design. Springer Berlin Heidelberg, 2013. 97-108.
- [54] al-Rifaie, Mohammad Majid, Mark JM Bishop, and Ahmed Aber. "Creative or Not? Birds and Ants Draw with Muscle." Proceedings of AISB'11 Computing and Philosophy (2011): 23-30.
- [55] al-Rifaie, Mohammad Majid, Ahmed Aber and John Mark Bishop. "Cooperation of Nature and Physiologically Inspired Mechanisms in Visualisation." Biologically-Inspired Computing for the Arts: Scientific Data through Graphics. IGI Global, 2012. 31-58. Web. 22 Aug. 2013. doi:10.4018/978-1-4666-0942-6.ch003
- [56] al-Rifaie MM, Bishop M (2013) Swarm intelligence and weak artificial creativity. In: The Association for the Advancement of Artificial Intelligence (AAAI) 2013: Spring Symposium, Stanford University, Palo Alto, California, U.S.A., pp 14–19
- [57] Flocks, Herds, and Schools
- [58] Kelly, James Floyd. "Book Review and Author Interview: *Kill Decision* by Daniel Suarez". *Wired*. Condé Nast. Retrieved 11 January 2015.

143.7 External links

- International Journal of Swarm Intelligence Research (IJSIR)

143.8 Further reading

- Bernstein, Jeremy. "Project Swarm". *Report on technology inspired by swarms in nature*.
- Bonabeau, Eric; Dorigo, Marco; Theraulaz, Guy (1999). *Swarm Intelligence: From Natural to Artificial Systems*. ISBN 0-19-513159-2.
- Engelbrecht, Andries. *Fundamentals of Computational Swarm Intelligence*. Wiley & Sons. ISBN 0-470-09191-6.
- Fisher, L. (2009). *The Perfect Swarm : The Science of Complexity in Everyday Life*. Basic Books.
- Fister I, XS Yang, I Fister, J Brest and D Fister (2013) "A Brief Review of Nature-Inspired Algorithms for Optimization" *Elektrotehniski Vestnik*, **80** (3): 1–7.
- Horn, Eva; Gisi, Lucas (Ed.) Marco (2009). *Schwärme – Kollektive ohne Zentrum. Eine Wissensgeschichte zwischen Leben und Information*. Bielefeld. ISBN 978-3-8376-1133-5.
- Kaiser, Carolin; Kröckel, Johannes; Bodendorf, Freimut (2010). "Swarm Intelligence for Analyzing Opinions in Online Communities". *Proceedings of the 43rd Hawaii International Conference on System Sciences*. pp. 1–9.
- Kennedy, James; Eberhart, Russell C. *Swarm Intelligence*. ISBN 1-55860-595-9.
- Miller, Peter (July 2007), "Swarm Theory", *National Geographic Magazine*
- Resnick, Mitchel. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. ISBN 0-262-18162-2.
- Ridge, E.; Curry, E. (2007). "A roadmap of nature-inspired systems research and development". *Multagent and Grid Systems* **3** (1): 3–8. CiteSeerX: 10.1.1.67.1030.

- Ridge, E.; Kudenko, D.; Kazakov, D.; Curry, E. (2005). “Moving Nature-Inspired Algorithms to Parallel, Asynchronous and Decentralised Environments”. *Self-Organization and Autonomic Informatics (I)* **135**: 35–49. CiteSeerX: 10.1.1.64.3403.
- *Swarm Intelligence* (journal). Chief Editor: Marco Dorigo. Springer New York. ISSN 1935-3812 (Print) 1935-3820 (Online)
- Waldner, Jean-Baptiste (2007). *Nanocomputers and Swarm Intelligence*. ISTE. ISBN 978-1-84704-002-2.
- Yang, Xin-She (2011). “Metaheuristic Optimization”. *Scholarpedia* **6** (8): 11472. Bibcode:2011SchpJ...611472Y. doi:10.4249/scholarpedia.11472.
- Zimmer, Carl (November 13, 2007). “From Ants to People: an Instinct to Swarm”. *The New York Times*.

Chapter 144

TOLMIN (optimization software)

This article is about TOLMIN, an optimization algorithm/software by Professor Michael J. D. Powell. For other uses of the word “Tolmin”, see Tolmin (disambiguation).

TOLMIN [1][2] is a numerical optimization algorithm by Michael J. D. Powell. It is also the name of Powell's Fortran 77 implementation of the algorithm.

TOLMIN seeks the minimum of a differentiable nonlinear function subject to linear constraints (equality and/or inequality) and simple bounds on variables. Each search direction is calculated so that it does not intersect the boundary of any inequality constraint that is satisfied and that has a “small” residual at the beginning of the line search. The meaning of “small” depends on a parameter called TOL which is automatically adjusted, and which gives the name of the software.

Features of the software include: quadratic approximations of the objective function whose second derivative matrices are updated by means of the BFGS formula, active sets technique, primal-dual quadratic programming procedure for calculation of the search direction.

144.1 See also

- COBYLA
- UOBYQA
- NEWUOA
- BOBYQA
- LINCOA

144.2 References

- [1] Powell, M. J. D. (1989). “A tolerant algorithm for linearly constrained optimization calculations”. *Mathematical Programming* (Springer) **45**: 547-566. doi:10.1007/BF01589118.
- [2] “Source code of TOLMIN software”. Retrieved 2015-04-06.

144.3 External links

- Source code of TOLMIN software
- A repository of Powell's software
- Nonlinear Programming Packages --- TOLMIN

Chapter 145

Tree rearrangement

Tree rearrangements are used in heuristic algorithms devoted to searching for an optimal tree structure. They can be applied to any set of data that are naturally arranged into a tree, but have most applications in computational phylogenetics, especially in maximum parsimony and maximum likelihood searches of phylogenetic trees, which seek to identify one among many possible trees that best explains the evolutionary history of a particular gene or species.

145.1 Basic tree rearrangements

- Nearest neighbor interchange (NNI)
- Subtree pruning and regrafting (SPR)
- Tree bisection and reconnection (TBR)

The simplest tree-rearrangement, known as **nearest-neighbor interchange**, exchanges the connectivity of four subtrees within the main tree. Because there are three possible ways of connecting four subtrees,^[1] and one is the original connectivity, each interchange creates two new trees. Exhaustively searching the possible nearest-neighbors for each possible set of subtrees is the slowest but most optimizing way of performing this search. An alternative, more wide-ranging search, **subtree pruning and regrafting** (SPR), selects and removes a subtree from the main tree and reinserts it elsewhere on the main tree to create a new node. Finally, **tree bisection and reconnection** (TBR) detaches a subtree from the main tree at an interior node and then attempts all possible connections between branches of the two trees thus created. The increasing complexity of the tree rearrangement technique correlates with increasing computational time required for the search, although not necessarily with their performance.^[2]

145.2 Tree fusion

The simplest type of tree fusion begins with two trees already identified as near-optimal; thus, they most likely have the majority of their nodes correct but may fail

to resolve individual tree “leaves” properly; for example, the separation ((A,B),(C,D)) at a branch tip versus ((A,C),(B,D)) may be unresolved.^[1] Tree fusion swaps these two solutions between two otherwise near-optimal trees. Variants of the method use standard **genetic algorithms** with a defined **objective function** to swap high-scoring subtrees into main trees that are high-scoring overall.^[3]

145.3 Sectorial search

An alternative strategy is to detach part of the tree (which can be selected at random, or using a more strategic approach) and to perform TBR/SPR/NNI on this sub-tree. This optimized sub-tree can then be replaced on the main tree, hopefully improving the p-score.^[4]

145.4 Tree drifting

To avoid entrapment in local optima, a 'simulated annealing' approach can be used, whereby the algorithm is occasionally permitted to entertain sub-optimal candidate trees, with a probability related to how far they are from the optimum.^[4]

145.5 Tree fusing

Once a range of equally-optimal trees have been gathered, it is often possible to find a better tree by combining the “good bits” of separate trees. Sub-groups with an identical composition but different topology can be switched and the resultant trees evaluated.^[4]

145.6 References

- [1] Felsenstein J. (2004). *Inferring Phylogenies* Sinauer Associates: Sunderland, MA.
- [2] Takahashi K, Nei M. (2000). Efficiencies of fast algorithms of phylogenetic inference under the criteria of

maximum parsimony, minimum evolution, and maximum likelihood when a large number of sequences are used.
Mol Biol Evol 17(8):1251-8.

- [3] Matsuda H. (1996). Protein phylogenetic inference using maximum likelihood with a genetic algorithm. *Pacific Symposium on Biocomputing* 1996, pp512-23.
- [4] Goloboff, P. (1999). Analyzing Large Data Sets in Reasonable Times: Solutions for Composite Optima. *Cladistics*, 15(4), 415–428. doi:10.1006/clad.1999.0122

Chapter 146

UOBYQA

UOBYQA (Unconstrained Optimization BY Quadratic Approximation)^{[1][2][3]} is a numerical optimization algorithm by Michael J. D. Powell. It is also the name of Powell's Fortran 77 implementation of the algorithm.

UOBYQA solves unconstrained optimization problems without using derivatives, which makes it a derivative-free algorithm. The algorithm is iterative, and exploits trust region technique. On each iteration, the algorithm establishes a quadratic model Q_k by interpolating the objective function at $(n+1)(n+2)/2$ points, and then minimizes Q_k within a trust region.

After UOBYQA, Powell developed NEWUOA, which also solves unconstrained optimization problems without using derivatives. In general, NEWUOA is much more efficient than UOBYQA and is capable of solving much larger problems (with up to several hundreds of variables). A major difference between them is that NEWUOA constructs quadratic models by interpolating the objective function at much less than $(n+1)(n+2)/2$ points (2n + 1 by default^[4]).

146.1 See also

- TOLMIN
- COBYLA
- NEWUOA
- BOBYQA
- LINCOA

146.2 References

- [1] Powell, M. J. D. (December 2000). UOBYQA: unconstrained optimization by quadratic approximation (Report). Department of Applied Mathematics and Theoretical Physics, Cambridge University. DAMTP 2000/NA14. Retrieved 2015-04-06.
- [2] Powell, M. J. D. (2002). “UOBYQA: unconstrained optimization by quadratic approximation”. *Mathematical Programming, Series B* (Springer) **92**: 555–582. doi:10.1007/s101070100290.

- [3] “Source code of UOBYQA software”. Retrieved 2015-04-06.
- [4] “Source code of NEWUOA software”. Retrieved 2014-01-14.

146.3 External links

- Source code of UOBYQA software
- A repository of Powell's software

Chapter 147

Very large-scale neighborhood search

In mathematical optimization, Neighborhood Search is a technique that tries to find good or near-optimal solutions to a mathematical optimisation problem by repeatedly trying to improve the current solution by looking for a better solution which is in the neighbourhood of the current solution. In that sense, the neighborhood of the current solution includes a possibly large number of solutions which are near to the current solution. Obviously, there is a degree of looseness in that definition in that the neighborhood might include just those solutions that require a single change from the current solution, or it might include the larger set of solutions that differ in two or more values from the current solution. A **very large-scale neighborhood search** is a local search algorithm which makes use of a **neighborhood definition**, which is large and possibly exponentially sized.

The resulting algorithms are often far superior to algorithms using small neighborhoods because the local improvements are larger. If the neighbourhood searched is limited to just one or a very small number of changes from the current solution, then it is often very difficult to escape from local minima and additional meta-heuristic techniques may need to be used such as Simulated Annealing or Tabu search to allow the search process to escape from a local minimum. In large neighborhood search techniques, the possible changes from one solution to its neighbor may allow tens or hundreds of values to change, and this means that the size of the neighborhood may itself be sufficient to allow the search process to avoid or escape local minima. As a result, it is often unnecessary to introduce additional meta-heuristic techniques.

147.1 References

- Ahuja, Ravindra K.; Orlin, James B.; Sharma, Dushyant (2000), “Very large-scale neighborhood search”, *International Transactions in Operational Research* 7 (4–5): 301–317, doi:10.1111/j.1475-3995.2000.tb00201.x.

Chapter 148

Zionts–Wallenius method

The **Zionts–Wallenius method** is an interactive method used to find a best solution to a multi-criteria optimization problem.

148.1 Detail

Specifically it can help a user solve a linear programming problem having more than one (linear) objective. A user is asked to respond to comparisons between feasible solutions or to choose directions of change desired in each iteration. Providing certain mathematical assumptions hold, the method finds an optimal solution.

148.2 References

- Zionts, S. and J. Wallenius, “An Interactive Programming Method for Solving the Multiple Criteria Problem,” Management Science. Vol. 22, No. 6, pp. 652–663, 1976.

148.3 Text and image sources, contributors, and licenses

148.3.1 Text

- **Mathematical optimization** *Source:* <http://en.wikipedia.org/wiki/Mathematical%20optimization?oldid=655172900> *Contributors:* Damian Yerrick, AxelBoldt, Zundark, The Anome, Ap, Awaterl, Peterlin, Rade Kutil, Heron, Ryguasu, Patrick, Michael Hardy, Chan siuman, Fred Bauder, David Martland, Karada, Stevan White, Mxn, Hike395, Jurgen, Charles Matthews, Dysprosia, Jitse Niesen, Topbanana, David.Monniaux, Robbot, Ojigiri, Moink, Daniel Dickman, Giftlite, BenFrantzDale, Lethe, Bfinn, Ajgorhoe, Andris, Jason Quinn, Wmahan, KaHa242, MarkSweep, APH, H Padleckas, Fintor, Skifreak, PhotoBox, Mike Rosoft, Discospinster, Mat cross, Paul August, Saraedum, Lycurgus, Aaronbrick, Obradovic Goran, Mdd, Tsirel, Msh210, Diego Moya, Olealexandrov, Pontus, Ishishwar, Artur adib, MIT Trekkie, Oleg Alexandrov, Woohookitty, Myleslong, Smmurphy, Isnow, Marudubshinki, JIP, Jorunn, Rjwilmsi, Tizio, Salix alba, DonSiano, MarcoL, Mathbot, Nimur, Kolbasz, TeaDrinker, Alphachimp, Chobot, Roboto de Ajvol, YurikBot, Wavelength, Deeptrivia, Encyclops, RussBot, Chaos, David R. Ingham, Voyevoda, RKUrsem, Retired username, Hakkinen, Dsol, Gglockner, Bota47, NormDor, Arthur Rubin, MaNeMeBasat, Nojhan, RandallZ, Sardanaphalus, SmackBot, InverseHypercube, Vermorel, Mcld, Misfeldt, Oli Filth, EncMstr, DHN-bot, MaxSem, Jasonb05, Ggpauly, Hua001, Brianboonstra, Tbboohher, Asm4, TwoCs, G.de.Lange, Antonielly, JHunterJ, Xprime, Carlo.milanesi, Hu12, Riedel, Polar Bear, Lavaka, CRGreathouse, CmdrObot, Jonnat, Jackzhp, Van helsing, Thomasmeeks, Solomon Douglas, Stebulus, Schaber, Cydebot, Mikewax, Czenek, Headbomb, Arnab das, KrakatoaKatie, AnAj, VictorAnyakin, JAnDbot, JPRBW, Ravelite, Roleplayer, LSpring, DRHagen, Smartcat, MartinDK, Sabamo, Mrbynum, Charlesreid1, Cic, Armehrabian, Ltk, David Eppstein, Martynas Patasius, Bradgib, Erkan Yilmaz, Mange01, Stochastics, Salih, Burgaz, Nwbeseson, JonMcLoone, Epistemical, Bonadea, LastChanceToBe, VolkovBot, LokiClock, Philip Trueman, Anonymous Dissident, TPlantenga, Broadbot, Srinnath, Johngcarlsson, JFPuget, Sapphic, Struway, Mangogirl2, Wikibuki, Zgueem, Truecobb, Gerakibot, Paolo.dL, CharlesGillingham, Iknowyourider, Dattorro, Rinconsoleao, Procellarum, PipepBot, Justin W Smith, PimBeers, Suegerman, Daveagg, Dmitrey, Autof6, Libertolotti, DragonBot, Dianegarey, Doobliebop, DumZiBoT, Thoughtfire, Oğuz Ergin, Sliders06, Addbot, Ablevy, İlker, Ekijoekoj, MrOllie, Download, HiYoSilver01, Delaszk, Kiril Simeonovski, Zorrobot, Jarble, ConstantLearner, Luckas-bot, Ashburyjohn, Yobot, Diracula, Travis.a.buckingham, Pcap, Lylenorton, Deuxoursendormis, Erel Segal, Galoubet, Citation bot, ArthurBot, Pownuk, Quebec99, Obersachsebot, Xqbot, Carbaholic, P99am, Micemug, Wamanning, Isheden, Almabot, Knillinux, Georg Stillfried, Omnipaedista, RibotBOT, Klochkov.ivan, YuriyMikhaylovskiy, Amaury, Optiy, Mcmlxxxii, Schlitz4U, Eiro06, Ct529, Kamitsaha, FrescoBot, Nageh, X7q, Asadi1978, D'ohBot, Pschaus, Boxplot, HRoestBot, Kiefer.Wolfowitz, MaximizeMinimize, Jmc200, FoxBot, Mdwang, Dsz4, Trappist the monk, Dpbert, Rbdevore, Bpadmakumar, Duoduoduo, BlakeJRiley, Crisgh, Hehsaan, RjwilmsiBot, Hosseininassab, Jowa fan, EmusaBot, John of Reading, Nacopt, Carbo1200, Cfg1777, Metafun, Psogeek, Daryakov, Optimering, Krystofer, Bonnans, ZéroBot, The Nut, AManWithNoPlan, Katie O'Hare, Robbiemorrison, Edesigner, Іванко1, Zfeinst, Syst analytic, Brycehughes, ClueBot NG, Jean-Charles.Gilbert, Chester Markel, Robiminer, Leonardo61, Sahinidis, Harrycaterpillar, Helpful Pixie Bot, Dwassel, Rxnt, Thiscomments voice, MSchlueter, FiveColourMap, Heroestr, Albert triv, Osiris, Centathlete, Mdann52, ChrisGualtieri, Denmartines, Makecat-bot, Limit-theorem, Delafé, PiotrGregorczyk, SakeUPenn, Lazars53, Buzzzman, Cubism44, Thennicke, Βελτιωτοποίησης, Mehr86, Mitsiko, Tyrneaplhth, Loraof and Anonymous: 250
- **Golden section search** *Source:* <http://en.wikipedia.org/wiki/Golden%20section%20search?oldid=655346597> *Contributors:* Charles Matthews, Evgeni Sergeev, Hyacinth, Phil Boswell, Giftlite, Sesse, PAR, Isarl, GregorB, Casey Abell, Finell, SmackBot, Saravask, Ligulembot, Rogerbrent, Lavaka, CRGreathouse, Cydebot, David Eppstein, User A1, Thomasda, Mark Walter, K.menin, TXiKiBoT, Quest for Truth, Sohanz, Shai mach, EverettYou, Avalcarce, Addbot, Favonian, Citation bot, Citation bot 1, Shuroo, Kiefer.Wolfowitz, Shahriar Heidrich, ZéroBot, Habstinat, Widr, CitationCleanerBot, Lambda2012 and Anonymous: 53
- **Powell's method** *Source:* <http://en.wikipedia.org/wiki/Powell%27s%20method?oldid=563534071> *Contributors:* Edward, Michael Hardy, Chuunen Baka, Andreas Kaufmann, Jonsafari, Realkyhick, SmackBot, Cydebot, DuncanHill, Magioladitis, K.menin, Malcolm McLean, Kmcallenberg, AnomieBOT, Isheden, Kiefer.Wolfowitz, ClueBot NG and Anonymous: 6
- **Line search** *Source:* <http://en.wikipedia.org/wiki/Line%20search?oldid=616422203> *Contributors:* Booyabazooka, BenFrantzDale, Ajgorhoe, Pgdn002, Mat cross, O18, Oleg Alexandrov, Eli Osherovich, Doktorbuk, AndrewKay, AhmedHan, A. Pichler, David Cooke, Cydebot, ChrisEich, Dekimasu, Petergans, Phe-bot, Hey i can fly, PerryTachett, Gjacquenot, Addbot, Yobot, Kiefer.Wolfowitz, Michael9422, Harshhpareek, ZéroBot, AManWithNoPlan, Realhamburger, BarrelProof, Gameboy97q and Anonymous: 15
- **Nelder–Mead method** *Source:* <http://en.wikipedia.org/wiki/Nelder%E2%80%93Mead%20method?oldid=640234029> *Contributors:* Schewek, Michael Hardy, Zeno Gantner, Stevenj, Jitse Niesen, ThomasStrohmann, Giftlite, Fintor, NathanHurst, Mdd, Forderud, Qwertys, Rjwilmsi, YurikBot, Encyclops, Bozoid, WAS 4.250, Reject, Lunch, SmackBot, Pedroso, Hongooi, Alexey Poklonskiy, Simiprof, A. Pichler, CBM, Cydebot, Fyedernoggersnodden, Thijs!bot, KrakatoaKatie, Smartcat, Sabamo, Bakken, Gwern, Yohan naftali, K.menin, JonMcLoone, Mathdeveloper, Arash the Archer, Mermalda, Tdi k, Csolbach, Heinrich Puschmann, DumZiBoT, Good Olfactory, Dhiraj113, Addbot, MrOllie, CompStatMech, Luckas-bot, Yobot, Kotika98, Confluente, Kiefer.Wolfowitz, KillerGardevoir, Gebreyo, Matus2, IJungreis, OhGodItsSoAmazing, Zhangzk, Monkbot, BernhardKarlAdolf and Anonymous: 61
- **Successive parabolic interpolation** *Source:* <http://en.wikipedia.org/wiki/Successive%20parabolic%20interpolation?oldid=643210489> *Contributors:* Btyner, Cydebot, K.menin, Kiefer.Wolfowitz, Jujutacular and Anonymous: 4
- **Trust region** *Source:* <http://en.wikipedia.org/wiki/Trust%20region?oldid=647340035> *Contributors:* Jurgen, Charles Matthews, Jitse Niesen, BenFrantzDale, Andrejj, Camw, ERcheck, Derek farn, Myasuda, Cydebot, He7d3r, Kolyma, Sameer0s, Addbot, Dumeineguete, Kiefer.Wolfowitz, Tom.Reding, Jean-Charles.Gilbert, FiveColourMap, Julia Abril, Susilehtola and Anonymous: 11
- **Wolfe conditions** *Source:* <http://en.wikipedia.org/wiki/Wolfe%20conditions?oldid=608251997> *Contributors:* Mark Foskey, Willem, Jitse Niesen, BenFrantzDale, Mat cross, O18, Hgkamat, Gareth Jones, Eli Osherovich, JJL, TheTito, Cydebot, Fbahr, Sohanz, Addbot, Drybid, Leonid Vohnitsky, Nocheenlatierra, Grigola2, Kiefer.Wolfowitz, RjwilmsiBot, Lugia2453 and Anonymous: 28
- **Broyden–Fletcher–Goldfarb–Shanno algorithm** *Source:* <http://en.wikipedia.org/wiki/Broyden%E2%80%93Fletcher%E2%80%93Goldfarb%E2%80%93Shanno%20algorithm?oldid=643159351> *Contributors:* Michael Hardy, Booyabazooka, Komap, Stevenj, Charles Matthews, Jitse Niesen, Benwing, Gdm, Nograpes, O18, Nicolasbock, TenOfAllTrades, Oleg Alexandrov, Male1979, Qwertys, Wihenao, Roystgnr, Paulck, Dtrebbeben, Janto, Alanb, Mebden, Lunch, Itub, JJL, SmackBot, Dycotiles, A. Pichler, Lavaka, Cydebot, Gnfnrf, Headbomb, Isilanes, Sabamo, Baccyak4H, R'n'B, LordAnubisBOT, K.menin, Steel1943, Mobiusinversion, Smarchesini, Eingangskontrolle, Gowerrobert, Addbot, Chemuser, Yobot, AnomieBOT, Citation bot, Omnipaedista, Drybid, X7q, Citation bot 1, Kiefer.Wolfowitz, Rushbugled13, AManWithNoPlan, Tomásdearg92, Ijdavis, BG19bot, Getreal123, Jugesh s, Chagmony, DrKBall, Austrartsua, Tomlof and Anonymous: 57

- **Limited-memory BFGS** *Source:* <http://en.wikipedia.org/wiki/Limited-memory%20BFGS?oldid=639137632> *Contributors:* Michael Hardy, Glenn, Jitse Niesen, Benwing, Bobo192, Rshin, Soultacon, Marasmusine, Qwertyus, Brendan642, Eli Osherovich, Avraham, SmackBot, Chris the speller, Danpovey, Memming, Noegenesis, Lavaka, Cydebot, Fredludlow, Abeppu, LongHei, K.menin, Muaddib131, Magnus, NathanHagen, Nocedal, Addbot, Delaszk, Yobot, JimVC3, FrescoBot, Lostella, Kiefer.Wolfowitz, AManWithNoPlan, Donner60, Helpful Pixie Bot, DavidRideout, Rebecca roelofs and Anonymous: 45
- **Davidon–Fletcher–Powell formula** *Source:* <http://en.wikipedia.org/wiki/Davidon%E2%80%93Fletcher%E2%80%93Powell%20formula?oldid=596391949> *Contributors:* Michael Hardy, Jitse Niesen, Oleg Alexandrov, A. Pichler, Cydebot, K.menin, Smarchesini, Hess88, Good Olfactory, Yobot, Omnipaedista, Kiefer.Wolfowitz, Rushbugled13, AManWithNoPlan and Anonymous: 6
- **Symmetric rank-one** *Source:* <http://en.wikipedia.org/wiki/Symmetric%20rank-one?oldid=633219794> *Contributors:* Jitse Niesen, BenFrantzDale, Qwertyus, Roystgnr, Noegenesis, Cydebot, Headbomb, Whatamidoing, K.menin, Smarchesini, Bte99, Yobot, Kiefer.Wolfowitz, RjwilmsiBot, Helpful Pixie Bot, Monkbot and Anonymous: 3
- **Gauss–Newton algorithm** *Source:* <http://en.wikipedia.org/wiki/Gauss%E2%80%93Newton%20algorithm?oldid=653277600> *Contributors:* Charles Matthews, Jitse Niesen, Gifflite, BenFrantzDale, Frau Holle, Vipul, NK, 3mta3, OlegAlexandrov, Oleg Alexandrov, BD2412, Rjwilmsi, Gaius Cornelius, BOT-Superzerocool, Tim314, Bugloaf, Kostmo, Berland, Adriferr, Cydebot, Thijssbot, OrenBochman, Science History, Smartcat, Baccyak4H, Noytza, K.menin, Jmath666, Petergans, Melcombe, BOTarate, Addbot, LaaknorBot, Zhaojun, Luckas-bot, Yobot, AnomieBOT, X7q, Citation bot 1, Buchtak, Kiefer.Wolfowitz, John of Reading, Slawekb, Zueignung, Michaelnikolaou, ClueBot NG, Alex.j.flint, Helpful Pixie Bot, BG19bot, Schrosby, Toranome, Cassiorconti, Mehr86, Monkbot and Anonymous: 59
- **Gradient descent** *Source:* <http://en.wikipedia.org/wiki/Gradient%20descent?oldid=654463566> *Contributors:* Komap, TakuyaMurata, Salsa Shark, Cherkash, Hike395, Jitse Niesen, Altenmann, Gifflite, BenFrantzDale, Mark T, Adam McMaster, Jackol, Pgan002, Gene s, Fintor, Nowozin, Mat cross, O18, Arcencil, Joris Gillis, OlegAlexandrov, MFH, Waldir, BD2412, DoubleBlue, Kri, Bgwhite, Roboto de Ajvol, JJL, SmackBot, InverseHypercube, Tgdwyer, Mcld, Nbarth, Imperiu, Nillerdk, T-borg, Cowbert, Rainwarrior, Simguru, Simiprof, A. Pichler, Lavaka, Cydebot, Playtime, Headbomb, NocNokNeo, OrenBochman, Ben pcc, VictorAnyakin, .anacondabot, Sabamo, Americanhero, Martynas Patasius, Conquerist, DarkFalls, LordAnubisBOT, K.menin, Epistemical, TXiKiBoT, LiranKatzir, M7ammad, SieBot, Lagrange613, Alecs.y.rodez, Dlrohrer2003, He7d3r, Rubybrian, Qwfp, Addbot, Enormator, Coolwangyx, Yobot, Legendre17, AnomieBOT, Isheden, Bpgagangras, Hamamelis, CES1596, FrescoBot, X7q, HamburgerRadio, Kiefer.Wolfowitz, Math446, Lelouchlvcc, MladenWiki, Sajeewasp, RjwilmsiBot, Joseolasa, John of Reading, Mliu7, Profetylen, Slawekb, 2andrewknyazev, Chewings72, Peryeat, Jean-Charles.Gilbert, LisaXu, Shape Prior, Helpful Pixie Bot, BG19bot, Colbert Sesanker, Qetuth, Herve.lombaert, Thomas.P.Hayes, 18[], Gwenwoods and Anonymous: 77
- **Levenberg–Marquardt algorithm** *Source:* <http://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt%20algorithm?oldid=653094329> *Contributors:* Fniesen, Michael Hardy, Mdupont, Charles Matthews, Jitse Niesen, Gutza, Romanm, Rorro, Connelly, Gifflite, BenFrantzDale, Podgib, Two Bananas, Simoneau, Frau Holle, Billlion, Photoniique, Oleg Alexandrov, Bytyer, Qwertyus, Rjwilmsi, Rillian, Who, Karipuf, Avraham, Alanb, Willy², Ploncomi, Serg3d2, Jcarroll, Ckerr, Misfeldt, Berland, Biehl, Feraudyh, Dicklyon, Wizard191, A. Pichler, Rocketman768, CmdrObot, Cydebot, KrakatoaKatie, Stangaa, Chrisempson, Baccyak4H, Gwern, Supernatent, K.menin, Nicolaum, Vanished user 47736712, Mr. PIM, Petergans, Lourakis, Cramanaona, Tzangent, PixelBot, Frau K, Pixelator30, Kolyma, WikHead, Soulzone, Addbot, DOI bot, Sket16, Lightbot, Sun juana, Yobot, Kilom691, MihalOrela, Jose278, Vindematrix, Xqbot, Geobiophys, Superdingo101, GrouchoBot, Locobot, FrescoBot, Maverick9711, Citation bot 1, Kiefer.Wolfowitz, Itsworker, Kijninsert, Orzelf, RjwilmsiBot, Jwx, John of Reading, Scientific29, Mikhail Ryazanov, Wcherowi, Fasteddiempaw, BG19bot, Axxelw, Ulbin, HongChenPeter, AvengerDr, Mehr86, Monkbot, Cvfishyu, Lucien.Wang, Raullaasner and Anonymous: 114
- **Nonlinear conjugate gradient method** *Source:* <http://en.wikipedia.org/wiki/Nonlinear%20conjugate%20gradient%20method?oldid=655144338> *Contributors:* Edward, Jitse Niesen, Gifflite, BenFrantzDale, Paul August, Eli Osherovich, SmackBot, TobyK, Jim.belk, A. Pichler, Cydebot, Kaiserkarl13, K.menin, Billinghurst, Smarchesini, YetAnotherBunny, Sbsta, CES1596, Kiefer.Wolfowitz, Logari81, Cauchym and Anonymous: 18
- **Newton's method in optimization** *Source:* <http://en.wikipedia.org/wiki/Newton%20method%20in%20optimization?oldid=634724470> *Contributors:* CesarB, Jitse Niesen, Evgeni Sergeev, Nonick, Gifflite, Fintor, O18, LutzL, OlegAlexandrov, Oleg Alexandrov, Male1979, Rjwilmsi, Mathbot, KSmrq, Goffrie, NormDor, SmackBot, Oli Filth, Effoff, Berland, A. Pichler, Bwibbwz, Cydebot, Mibchronicles, Headbomb, Ben pcc, Sabamo, Janislaw, RichardSocher, Thomas.kn, Smarchesini, Shai mach, Yobot, Midinasturazz, Isheden, Strepon, Citation bot 1, Kiefer.Wolfowitz, Jujutacular, Mattrbunique, Bonnans, ChrisGualtieri, Hugopp and Anonymous: 39
- **Barrier function** *Source:* <http://en.wikipedia.org/wiki/Barrier%20function?oldid=638794223> *Contributors:* Jitse Niesen, Richard Stephens, Mecanismo, Marner, Oleg Alexandrov, TeaDrinker, SmackBot, Serg3d2, Fikus, Cydebot, VolkovBot, Addbot, Citation bot, Kiefer.Wolfowitz, Trappist the monk, Zfeinst, Massly, Brad7777, OhGodItsSoAmazing, Paul2520 and Anonymous: 3
- **Penalty method** *Source:* <http://en.wikipedia.org/wiki/Penalty%20method?oldid=609926299> *Contributors:* Michael Hardy, Jitse Niesen, BenFrantzDale, Ajgorhoe, Scirinæ, Oleg Alexandrov, Rjwilmsi, Serg3d2, Cydebot, KrakatoaKatie, EagleFan, Neeku, Ilanko, He7d3r, Addbot, DrilBot, Kiefer.Wolfowitz, Duoduoduo, EmausBot, Zfeinst, ChuispastonBot, Atomician, Amitiimb, Ajbilan and Anonymous: 11
- **Augmented Lagrangian method** *Source:* <http://en.wikipedia.org/wiki/Augmented%20Lagrangian%20method?oldid=654292462> *Contributors:* Michael Hardy, Chris Howard, Gaius Cornelius, Johndburger, Serg3d2, Lavaka, Cydebot, Magioladitis, Bluecacao, HughD, Jfessler, Mild Bill Hiccup, Garland3, Yobot, Kiefer.Wolfowitz, AManWithNoPlan, Bpdmit, BG19bot, Tgru, Chiyuan, Fergus the widget, Velvel2 and Anonymous: 8
- **Sequential quadratic programming** *Source:* <http://en.wikipedia.org/wiki/Sequential%20quadratic%20programming?oldid=615919983> *Contributors:* Michael Hardy, Panoptical, SmackBot, Disavian, Deditos, Affable guy, Cydebot, Patrick O'Leary, Headbomb, IdoRosen, David Eppstein, R'n'B, Fecurt, Addbot, Alexander.mitsos, Luckas-bot, KommX, Kiefer.Wolfowitz, Thrufir, Louis Winthorpe III and Anonymous: 13
- **Successive linear programming** *Source:* <http://en.wikipedia.org/wiki/Successive%20linear%20programming?oldid=621726812> *Contributors:* RJFJR, Oleg Alexandrov, Qwertyus, Lockley, Action potential, SmackBot, Cydebot, David Eppstein, Sdudah, KJN256, Yobot, LilHelpa, Kiefer.Wolfowitz, Thrufir, Robbie Morrison, Jordicuest, CallForSanity, Monkbot, Tyrneaplith and Anonymous: 1
- **Cutting-plane method** *Source:* <http://en.wikipedia.org/wiki/Cutting-plane%20method?oldid=649729607> *Contributors:* The Anome, HenHei, Zaslav, Remuel, Oleg Alexandrov, Tribaal, SmackBot, RDBury, Reedy, Lavaka, Cydebot, BrotherE, Magioladitis, Sabamo, David Eppstein, Falcor84, LordAnubisBOT, K.menin, Cobi, Appoose, JohngCarlsson, PipepBot, Addbot, SpellingBot, Gabrio, IFOR-ETHZ, AnomieBOT, Xqbot, Bellerophon, FrescoBot, Kiefer.Wolfowitz, Jan Pöschko, Duoduoduo, Ashutosh y0078, Pobrasil, Ypnypn, Lovasoa, Brateevsky and Anonymous: 29

- **Frank–Wolfe algorithm** *Source:* <http://en.wikipedia.org/wiki/Frank%20Wolfe%20algorithm?oldid=650048137> *Contributors:* Michael Hardy, Charles Matthews, Nowozin, MarSch, Encyclops, SmackBot, Stifle, Jon Awbrey, J. Finkelstein, Cydebot, Thijs!bot, SalvNaut, K.menin, Wikisawesome, Bhaktivinode, Addbot, Yobot, Isheden, DrilBot, Kiefer.Wolfowitz, EmausBot, Yuzhang49, AManWithNoPlan, Snottbot, BG19bot, CitationCleanerBot, Saung Tadashi, Mathmon, Martinjaggi, Sumingwu and Anonymous: 9
- **Subgradient method** *Source:* <http://en.wikipedia.org/wiki/Subgradient%20method?oldid=650048048> *Contributors:* Oleg Alexandrov, Dv82matt, A1octopus, Riedel, CRGreathouse, Cydebot, Headbomb, DavidCBryant, TXiKiBoT, Johngcarlsson, ShinuK, Addbot, X7q, Kiefer.Wolfowitz, John745, Bpdmit, PiotrGregorczyk, Moritzge, Monkbot and Anonymous: 6
- **Ellipsoid method** *Source:* <http://en.wikipedia.org/wiki/Ellipsoid%20method?oldid=572820613> *Contributors:* Jitse Niesen, Giftlite, Jérôme, YurikBot, Klutzy, Arthur Rubin, Meld, Mhym, Riedel, Gmath, CRGreathouse, Headbomb, Rlopez, Tremilux, David Eppstein, Anaxial, Lonjers, Johngcarlsson, Pitoutom, Svick, Fgdorais, Sharma337, Addbot, Yobot, AnomieBOT, Isheden, X7q, Camiort, Kiefer.Wolfowitz, Rionda, Ben Shamos, Zfeinst, Mark viking and Anonymous: 12
- **Karmarkar's algorithm** *Source:* <http://en.wikipedia.org/wiki/Karmarkar%20algorithm?oldid=650807549> *Contributors:* Jitse Niesen, Giftlite, Edcolins, Gdm, Ary29, NathanHurst, Rich Farmbrough, BlueNovember, Mahanga, Woohookitty, Qwertus, MarcoL, RussBot, SmackBot, Drlog, Tekhnofiend, Iannmacm, Rijkbeni, Silvescu, Norm mit, Gmath, CRGreathouse, Stormwurm, Cydebot, BetacommandBot, Headbomb, Mr pand, Seaphoto, Magioladitis, Kope, Vanderbei, Mange01, K.menin, Addbot, Lightbot, Yobot, Materialscientist, Citation bot, Xqbot, Anna Frodesiak, Citation bot 1, Kiefer.Wolfowitz, Duoduoduo, RjwilmsiBot, ZéroBot, Sallypally, Rezabot, Anthony bajaj, Pushpakb, BG19bot, Monkbot and Anonymous: 29
- **Simplex algorithm** *Source:* <http://en.wikipedia.org/wiki/Simplex%20algorithm?oldid=653132655> *Contributors:* Heron, Michael Hardy, Hike395, Barak, Charles Matthews, Dcoetze, Viz, Jitse Niesen, Abhishek, Phil Boswell, Pfortuny, Jaredwf, Sanders muc, Altenmann, Enochlau, Giftlite, BenFrantzDale, Nerd65536, KeithTyler, Shahab, NathanHurst, Discospinster, Paul August, C S, Dungodung, Mdd, HasharBot, Aegis Maelstrom, Forderud, Kenyon, Oleg Alexandrov, Myleslong, Oliphant, Ruud Koot, Jac, Magister Mathematicae, Qwertus, Rjwilmsi, Baryonic Being, Mathbot, New Thought, Nimur, Predictor, Chobot, YurikBot, Wavelength, RobotE, RussBot, Tong, Petter Strandmark, WAS 4.250, Arthur Rubin, JoanneB, Reject, SmackBot, RDGuy, Numsgil, CrazyTerabyte, Lubos, Adpete, Nbarth, Mcaruso, Jjbeard, Rrburke, Brian Parker, Epachamo, Nmnogueira, Tbbrooher, Jim.belk, Denshade, Yoderj, Cjohnzen, CRGreathouse, Stebulus, Cydebot, Wikid77, Headbomb, Pvjpj, VictorAnyakin, Apavlo, Smartcat, David Eppstein, Edurant, DirkOliverTheis, J.delanoy, Xyz9000, LordAnubisBOT, K.menin, WebHamster, JonMcLoone, Mehmetgencer, Epistemical, Tomerfiliba, Akhram, Cnsdtzsf, Madmax.ptz, AhnoktaBOT, Afluent Rider, JayC, Gerdemb, Bob the third, Finity, Tutor dave, SieBot, Marton78, Rgrimson, Paolo.dL, Alexey.salmikov, Ctxppc, Svick, Mghunt, Pecondon, Johnuniq, XLinkBot, Addbot, Sandskies, MrOllie, Abovechief, Lightbot, Vasil', Luckas-bot, Yobot, Angeljon121, Brightgalrs, Twri, Gsmgm, Xqbot, Isheden, Raulshc, Guillorama, LukeOrland, Kxx, Jajhall, Alexeicolin, Kreisaisjelis, Citation bot 1, Wingandaprayer, Kiefer.Wolfowitz, Duoduoduo, DARTH SIDIOUS 2, TryamMan, EmausBot, Inframaut, Ricardogobbo, AManWithNoPlan, Furries, Zfeinst, ClueBot NG, Aadornellesf, Sleight, Kasirbot, Nullzero, Helpful Pixie Bot, Simplexsolver, Toffanin, AK456, Lanjagudda, Hermann Döppes, Mhauwe, Monkbot and Anonymous: 144
- **Revised simplex method** *Source:* <http://en.wikipedia.org/wiki/Revised%20simplex%20method?oldid=635462869> *Contributors:* Michael Hardy, Kxx, Tyrneaplith and Anonymous: 2
- **Criss-cross algorithm** *Source:* <http://en.wikipedia.org/wiki/Criss-cross%20algorithm?oldid=644476212> *Contributors:* Charles Matthews, Phil Boswell, Ruud Koot, GregorB, Qwertus, Rjwilmsi, Cydebot, Headbomb, Magioladitis, David Eppstein, Addbot, Tassedethe, Yobot, Quebec99, Kiefer.Wolfowitz, Trappist the monk, RjwilmsiBot, Frietjes, Helpful Pixie Bot, Jckrgn600, Monkbot, Tyrneaplith and Anonymous: 3
- **Lemke's algorithm** *Source:* <http://en.wikipedia.org/wiki/Lemke%20algorithm?oldid=607238388> *Contributors:* Numsgil, CmdrObot, Cydebot, David Eppstein, Vacary, Kpstewart, Kiefer.Wolfowitz and Immaciek
- **Approximation algorithm** *Source:* <http://en.wikipedia.org/wiki/Approximation%20algorithm?oldid=651505318> *Contributors:* Danny, Dcoetze, Fredrik, Ojigiri, Giftlite, Mellum, Andris, NathanHurst, Jnestori, Haham hanuka, Arthena, Culix, Oleg Alexandrov, Decrease789, Ruud Koot, GregorB, Rjwilmsi, Chobot, YurikBot, Bota47, Tribaal, Bmatheny, SmackBot, DKalkin, Pnamjoshi, Myasuda, NotQuiteEXCComplete, LachlanA, Dricherby, Tiagofassoni, David Eppstein, LordAnubisBOT, Ashishgoel.1973, Rafox, Vavi, Paolo.dL, Whodoesthis, RMFan1, Kolyma, C. lorenz, Addbot, CarsracBot, Numbo3-bot, Yobot, Citation bot, Anonash, Kiefer.Wolfowitz, RobinK, EmausBot, ZéroBot, Howard nyc, Helpful Pixie Bot, BattyBot, Olonic, Mariolucic, Mark viking, Metin.balaban, Infinitestory and Anonymous: 30
- **Dynamic programming** *Source:* <http://en.wikipedia.org/wiki/Dynamic%20programming?oldid=654165815> *Contributors:* FvdP, Imran, Nealmc, Michael Hardy, Chan siuman, Chris, Kku, Paddu, Kaeslin, Julesd, Hike395, 1baumann, Dcoetze, Furykef, Shantavira, Jaredwf, Fredrik, Altenmann, Phatsphere, Babbage, Mybot99999, LX, Dbroadwell, Ancheta Wis, Giftlite, Beefman, Mark T, Ketil, Leonard G., Waxmop, Nayuki, Sydneyfong, Utcursch, LiDaobing, HorsePunchKid, Beland, Karl-Henner, Sam Hocevar, Jmeppley, Pm215, Oskar Sigvardsson, Freakofnurture, AshtonBenson, Officiallyover, Mdd, Wongljie, Wtmitchell, Oleg Alexandrov, Conskeptical, Miaow Miaow, Oliphant, JonH, Miss Madeline, Smmurphy, Zzyzx11, Qwertus, Grammarbot, Chipuni, VKokielov, Mathbot, Intgr, Mahlon, Atif.hussain, Sodin, Kri, Chobot, Meonkeys, YurikBot, Vector, Gene.arboit, Hyad, Guslacerda, Cancan101, Hgranqvist, Gaius Cornelius, TheMandarin, Ethan, Grafen, Nils Grimsmo, Mikeblas, Zwobot, Simicich, Lt-wiki-bot, Zerodamage, Cedar101, Tom Duff, MrGBug, Allens, Crystallina, Cannin, SmackBot, KocjoBot, Spireguy, Eupedia, Chris the speller, SSJemmett, Bluebot, TripleF, Tamfang, Szarka, Zirconscot, Cybercobra, Mlpkr, Abi79, Fingew, Ben Moore, Lim Wei Quan, Dicklyon, Hackerb9, Pjrm, Mwj, Aeons, A. Pichler, CRGreathouse, Jackzhp, Cydebot, SavantEdge, Zahlentheorie, Nowhere man, Patrick O'Leary, JMatthews, Headbomb, Shobitb, Jeff Edmonds, LachlanA, Stannered, Jirislaby, Widefox, Matforddavid, SamIAMNot, Spinmeister, Erxnmedia, JAnDbot, .anacondabot, Magioladitis, Cic, David Eppstein, Vssun, JaGa, Falcor84, Ztobor, Dcwills, R'n'B, Edschofowler, Huggie, Jiuguang Wang, Vanished user 47736712, Remember the dot, Signalhead, VolkvBot, FatUglyJo, Philip Trueman, Rponamgi, Sriganeshs, RustyWP, AHMartin, Prakash Nadkarni, Tonya49, Richienumnum, Rinconsoleao, Michael Tangermann, ClueBot, Justin W Smith, The Thing That Should Not Be, Pekrau, Alex.altmaster, Zhouhowe, Mumiemonstret, Spmeyn, Npansare, Humanengr, Sniedo, Dthomsen8, Brentsmith101, FlagrantUsername, Addbot, D h benson, Aceituno, Rajkumar.p84, Gnorthup, Pixiefeet, Muhammadsuleman, Numbo3-bot, Lightbot, Bluebusy, Jarble, StandardPerson, Legobot, Luckas-bot, Yobot, Terrifictriffid, PMLawrence, AnomieBOT, Erel Segal, Rubinbot, Citation bot, ArthurBot, Xqbot, Vegpuff, Addihockey10, Drilnoth, Isheden, Miym, GrouchoBot, ChristopherKingChemist, Prunesqualer, FrescoBot, Zarei.h, Citation bot 1, Shuroo, Bunyk, Kiefer.Wolfowitz, Jschnur, Damian.frank, SeldonPlan, EmausBot, Eight40, Tommy2010, Guahnala, Mastergreg82, Zfeinst, Agrepin, Orange Suede Sofa, JFB80, 28bot, ClueBot NG, Ulflund, Satellizer, Thebomb556, Nullzero, MeshColour, Helpful Pixie Bot, Leopd, BG19bot, Compfreak7, Pozix604, Correctureguru, Ahmad.s.Hosseini, Failedwizard, ChrisGaultieri, Aglescu andrei, Makecat-bot, Isarra (HG), Walej, Will Faught, Beero1000, MilicaBar, Milicawiki, Sandefowler, Kwatron, Eronisko, Ljacqu, Wandering007, Tremendoustushar, Searchofpeace4all and Anonymous: 311

- **Greedy algorithm** *Source:* <http://en.wikipedia.org/wiki/Greedy%20algorithm?oldid=653492667> *Contributors:* AxelBoldt, Hfastedge, CatherineMunro, Notheruser, PeterBrooks, Charles Matthews, Dcoetzee, Malcohol, Jaredwf, Meduz, Sverdrup, Wlievens, Enochlau, Gifflite, Smjg, Kim Bruning, Jason Quinn, Pgdn002, Andycjp, Andreas Kaufmann, TomRitchford, Discospinster, ZeroOne, Nabla, Diomidis Spinellis, Nandhp, Obradovic Goran, Haham hanuka, CKlunck, Swapspace, Ralphy, Ryanmcaniel, Hammertime, Mechonbarsa, CloudNine, Mindmatrix, LOL, Cruccone, Ruud Koot, Que, Sango123, FlaBot, New Thought, Kri, Pavel Kotrc, YurikBot, Wavelength, Hairy Dude, TheMandarin, Nethgirb, Bota47, Marcosw, Darrel francis, SmackBot, Brianyoumans, Unyoyega, KocjoBot, NickShaforstoff, Trezatium, SynergyBlades, DHN-bot, Omgleus, MichaelBillington, Mlpkr, Wleizer, Cjhonz, Mcstrother, Suanshinghal, Cydebot, Jibbist, Thijs!bot, Wikipd77, Nkarthiks, Escarbot, Uselesswarrior, Clan-destine, Salgueiro, Chamale, Jddriesen, Albany NY, Magioladitis, Eleschinski2000, Avicennasis, David Eppstein, Mange01, Zangkannt, Policron, BernardZ, Maghnus, TXiKiBoT, ArzelaAscoli, Monty845, Hobartimus, Denisarona, HairyFotr, Meekywiki, Enmc, Addbot, Legobot, Fraggie81, Materialscientist, Shrishaster, Hhulzk, Rickproser, Scarlettail, Kuchayrameez, Srijanrshetty, Amaniitk, Boky90 and Anonymous: 106
- **Integer programming** *Source:* <http://en.wikipedia.org/wiki/Integer%20programming?oldid=654671532> *Contributors:* Michael Hardy, Delirium, Shahab, Mdd, SmackBot, DouglasCalvert, Vanisaac, CRGreathouse, CBM, Matforddavid, David Eppstein, JohnBlackburne, TXiKiBoT, Nxavar, YSSYguy, Vikasatk, Daveapp, Bender2k14, Tehanolamousone, Vog, Yobot, Erel Segal, HanPritch, Isheden, Miym, Omnipaedista, RibotBOT, Erik9bot, FrescoBot, X7q, Kiefer.Wolfowitz, TobeBot, Dinamik-bot, Duoduo, RjwilmsiBot, Ghostofkendo, Number473, AManWithNoPlan, Ivaенко1, Tijfo098, Amontjoy, Trevor x1968, Joerg Bader, MerlinIwBot, Helpful Pixie Bot, Anubhab91, Matencia29, Cliff12345, Bmears11, Garamond Lethe, YiFeiBot, Ashirbadm, Sseemayer, Vieque, Davidchavezc512 and Anonymous: 29
- **Branch and bound** *Source:* <http://en.wikipedia.org/wiki/Branch%20and%20bound?oldid=653423186> *Contributors:* The Anome, Michael Hardy, Karada, Hike395, Jits Niesen, Greenrd, Altenmann, Gifflite, Mellum, Leonard G., Jorge Stolfi, Gdr, Togo, Avatar, Kbhb3rd, Djlayton4, LOL, Dionyziz, Plrk, Qwertyus, Rjwilmsi, Maxal, Chobot, YurikBot, Stefan Udrea, SmackBot, Eskimbot, OnixWP, Mlpkr, Bwpach, CmdrObot, Jokes Free4Me, Cydebot, Alaiob, Meredyth, Itibas, A3nm, David Eppstein, Rodrigo braz, K.menin, STBotD, Mr murthy, Jamelan, WRK, Jahead, AlanUS, Fghijklm, Extensive, Denisarona, Cngoulimis, HairyFotr, Adrianwn, Algomastr, AgnosticPreachersKid, Addbot, Favonian, Lightbot, Luckas-bot, AnomieBOT, TheWeakWilled, Alejandro edera, Erik9bot, Milimetrl88, Kiefer.Wolfowitz, WikitanvirBot, AvicAWB, Elaz85, ClueBot NG, Meldraft, JYBot, Ajmath62, Comp.arch, Luc Jaulin and Anonymous: 68
- **Branch and cut** *Source:* <http://en.wikipedia.org/wiki/Branch%20and%20cut?oldid=644490614> *Contributors:* Bearcat, Altenmann, MacGyverMagic, Rjwilmsi, Chobot, Peter Grey, Ott2, TheMadBaron, John Broughton, NiTenIchiRyu, Bluebot, Sebdo, Cydebot, David Eppstein, K.menin, VolkovBot, Addbot, Ptbotgourou, KommX, Kiefer.Wolfowitz, Trappist the monk, Oerpli, ChuispastonBot, Bmears11, Pdutoit and Anonymous: 10
- **Bellman–Ford algorithm** *Source:* <http://en.wikipedia.org/wiki/Bellman%E2%80%93Ford%20algorithm?oldid=654815967> *Contributors:* Tomo, Michael Hardy, Docu, Ciphergoth, Poor Yorick, Charles Matthews, Dcoetzee, Itai, Fvw, Mazin07, Jaredwf, Fredrik, Altenmann, Bkell, Enochlau, Gifflite, BenFrantzDale, Brona, Stern, Andris, Wmahan, Gadfiuum, Sam Hocevar, Rspeer, Orbst, Jellyworld, Helix84, HasharBot, B3virq3b, Pion, Oleg Alexandrov, Brookie, Stderr.dk, LOL, BlankVerse, Ruud Koot, GregorB, Waldir, Agthorr, Sigkill, Qwertyus, Rjwilmsi, Salix alba, Ucucha, FlaBot, Ecb29, Mathbot, Nihiltres, Jftuga, AlexCovarrubias, Quuxplusone, Istanton, CiaPan, Chobot, FrankTobia, Robot0 de Ajvol, YurikBot, Wavelength, Rsrikanth05, Josteinaj, Nils Grimsimo, BOT-Superzerocool, Bota47, SmackBot, Posix4e, P b1999, Mcld, Skizzik, DHN-bot, Konstable, Anabus, Mathmike, N Shar, Solon.KR, SpyMagician, Drdevil44, Pjrm, CBM, Thijs!bot, Ebpr123, Headbomb, Williamf, Heineman, Law17, JAnDbot, Harish victory, Magioladitis, Abednigo, Stdazi, David Eppstein, Gwern, J.delanoy, LordAnubisBOT, Monsday, VolkovBot, Zholdas, TXiKiBoT, Ferengi, Aaron Rotenberg, Jamelan, SQL, AlleborgoBot, Aednichols, YonaBot, ToePeu.bot, VVVBot, Tvidas, Naroza, Arlekean, PipepBot, Justin W Smith, Pskjs, Alexbot, PixelBot, Aene, DumZiBoT, Writer130, Addbot, Tsunanet, Iceblock, Protonk, Luckas-bot, Yobot, Ptbotgourou, Linket, Backslash Forwardslash, PanLevan, ArthurBot, Str8no1, Miym, Mario777Zelda, Shuroo, RobinK, Dinamik-bot, ToneDaBass, Nostalgus, EmausBot, Wikipelli, Bjozen, Guahnala, AManWithNoPlan, ClueBot NG, Nullzero, Happyuk, Cyberbot II, Lone boatman, Aladdin.chettouh, Carlwitt, Gauravxpresa, Jianhui67, Jonchen42, Subshiri and Anonymous: 167
- **Borůvka's algorithm** *Source:* <http://en.wikipedia.org/wiki/Bor%C5%AFvka%7Bs%20algorithm?oldid=653267410> *Contributors:* Pierre-Abbat, Davif, Hike395, Charles Matthews, Dcoetzee, Dysprosia, Jogloran, Grendelkhan, Jaredwf, Altenmann, Captain Segfault, Gifflite, Rich Farmbrough, Zaslav, Oleg Alexandrov, Angr, Qwertyus, Rjwilmsi, YurikBot, Michael Slone, Toncek, Cedar101, SmackBot, Nkojuharov, Mgreenbe, Alieserab, Kendrick7, Agamir, Rps, David Eppstein, Cobi, Adam Zivner, Falcongl, AlleborgoBot, Alexbot, PixelBot, Aurinegro, Addbot, Lightbot, Luckas-bot, TABOT-zerem, Senitiel, Citation bot 1, Kaezo, EmausBot, Elanguescence, WikitanvirBot, ZéroBot, Swfung8, Jk2q3jrkls, Electriccatfish2 and Anonymous: 17
- **Dijkstra's algorithm** *Source:* <http://en.wikipedia.org/wiki/Dijkstra%7bs%20algorithm?oldid=654991034> *Contributors:* AxelBoldt, LC, Css, Shd, Matusz, Edemaine, Ezubaric, Someone else, Michael Hardy, Nixdorf, Kku, Cyde, Julesd, Aragorn2, Cema, Hashar, Charles Matthews, Timwi, Dcoetzee, Dysprosia, Gutza, Hao2lian, Itai, Csurgine, Shizhao, Owen, Quidquam, Jaredwf, Altenmann, MathMartin, Bkell, Wildcat dunny, Clementi, Decrypt3, Gifflite, Christopher Parham, BenFrantzDale, Tesse, Brona, Robert Southworth, Leonard G., AJim, Guanaco, Sundar, Esrogs, MarkSweep, Watcher, RISHARTHA, Gerrit, MementoVivere, Kooo, Kndiaye, ZeroOne, BACbKA, Diego UFCG, Vecrumba, RoyBoy, Aydee, Ewedistrict, Foobaz, Jellyworld, Quill18, Obradovic Goran, Haham hanuka, 4v4l0n42, HasharBot, Lawpjic, Jeltz, B3virq3b, Velella, Mikeo, K3rb, LunaticFringe, Oleg Alexandrov, Mahanga, ProBoj!, Shreevatsa, LOL, Oliphant, Dammaz74, Jacobolus, MattGiica, Drostie, Pol098, Ruud Koot, Noogz, GregorB, Dionyziz, Agthorr, Kesla, Graham87, Qwertyus, Laurinkus, Grammarbot, Rjwilmsi, Assimil8or, Dosman, B6s, TheRingess, Eric Burnett, Grantsteven, Mathiastck, Jorvis, Choess, Fresheneesz, King of Hearts, Chobot, Bgwhite, FrankTobia, YurikBot, Wavelength, Borgx, Angus Lepper, Michael Slone, Wierdy1024, CambridgeBay-Weather, Pseudomonas, Zhaladshar, Anog, Shizny, Nethgirb, Tomisti, Sarkar112, Abu adam, Zr2d2, GraemeL, Alanb, HereToHelp, DoriSmith, Sidonath, NetRoller 3D, Thijsj, Dudzcom, SmackBot, KocjoBot, Mgreenbe, Optikos, Gaiacarra, Oli Filth, MalafayaBot, Kostmo, DHN-bot, Frap, GRRuban, B^4, Ryan Roos, Illnab1024, Ycl6, Tobe, Slakr, SQGibbon, Scorinthia, MTSbot, RamiWissa, Norm mit, BranStark, Iridescent, Lavaka, Joniscool98, JForget, Ahy1, Ezrakilty, VTBassMatt, Arrenlex, Huazheng, T0ljan, Systelus, Boemanneke, ThomasGHenry, Thijs!bot, RodrigoCamargo, Crazy george, Sprhodes, Williamf, WikiSlasher, AntiVandalBot, Behco, Mccraig, Spencer, Dougher, Deflective, Harish victory, Gordonnovak, Jhev, Mwarren us, Radim Baca, JBocco, SiobhanHansa, Rami R, Stdazi, B3N, David Eppstein, Martynas Patasius, PoliticalJunkie, Piojo, Obscurans, Yonidebot, Ryanli, AlcoholVat, Turketwh, Joelimlimit, Bcnof, Sk2613, BernardZ, PesoSwe, JohnBlackburne, Andreasneumann, Soyutuny, TXiKiBoT, MusicScience, Dmforcier, Mcculley, Millecke, Mkw813, Kbkb, AlleborgoBot, Marneisam, Davekaminski, Rhanekom, Subh83, SieBot, Wphamilton, Adamarnesen, Sephiroth storm, Keilana, Digwuren, Beatle Fab Four, Gerel, Ctxppc, Svick, AlanUS, Sokari, ClueBot, Justin W Smith, Foxj, CGamesPlay, Pskjs, Cicconetti, Adamianash, Nanobear, Daveapp, Coralmizu, ElonNarai, Dr.Koljan, Peatar, Peasaep, Sniedo, DumZiBoT, XLinkBot, SilvonenBot, Hell112342, Apalamarchuk, SteveJothen, Addbot, DarrylNester, Alquantor, Alex.mccarthy, Jason.Rafe.Miller, KorinoChikara,

MrOllie, Herry12, Torla42, AgadaUrbanit, Vwm, Matěj Grabovský, Jarble, Luckas-bot, ZX81, Yobot, Vevek, AnomieBOT, Erel Segal, Arjun G. Menon, Rubinbot, Jim1138, Galoubet, Materialscientist, 90 Auto, Citation bot, ArthurBot, Amenel, Airlcorn2, LordArtemis, Crefrog, Davub, Jongman.koo, Thayts, Geron82, X7q, Recognizance, Ibmua, D'ohBot, Jewillco, Shuroo, Frankrod44, Jonesey95, The Arbiter, Skyerise, MondalorBot, Yutsi, Mikrosam Akademija 2, Merlion444, Dmitri666, Circoutprabu, Faure.thomas, EmausBot, Dreske, Pixelu, Kh naba, Blueshifting, Wikipelli, Pshanka, Pxtrme75, Woshiqiqiye, Allan speck, Sheepatgrass, Venkatrun95, ClueBot NG, Muon, Sambayless, Helpful Pixie Bot, Mr.TAMER.Shlash, Happyuk, Pilode, Arsstyleh, BattyBot, Xerox 5B, IkamusumeFan, Trunks175, Tehwikiwner, IgushevEdward, Thom2729, Megharajv, Lone boatman, MindAfterMath, Jochen Burghardt, Aladdin.chettouh, Sean-halle, Ryangerard, I am One of Many, Olivernina, Chehabz, Gauravxpress, Quenhiran, Juliusz Gonera, Alyssaq, Yujianzhao, Monkbot, Thegreekgonzo, Kanargas, Zairwolf, Giovatardu, Prakashmeansvictory, Z5eacom, Kuperfirn, Jackson tale and Anonymous: 497

- **Floyd–Warshall algorithm** Source: <http://en.wikipedia.org/wiki/Floyd%20algorithm?oldid=640979039> Contributors: Oliver, Michael Hardy, Shyamal, Poor Yorick, Dcoetze, Joy, Phil Boswell, Robbot, Jaredwf, Altenmann, Bkell, Giftlite, Smurfix, Mellum, Dfrankow, LiDaobing, MarkSweep, Two Bananas, Simoneau, Treyshonuff, Pandemias, Barfooz, Bobo192, Jellyworld, Greenleaf, Minghong, Obradovic Goran, Minority Report, Rabarberski, Kenyon, Ruud Koot, Opium, GregorB, Kesla, Qwertyus, Rjwilmsi, Nneoneo, W3bbo, Ropez, FlaBot, Quuxplusone, Intgr, Nicapicella, CiaPan, Chobot, YurikBot, Netvor, Taejo, Gaius Cornelius, Closedmouth, Donhalcon, JLaTondre, SmackBot, Cachedio, Fintler, Sr3d, Gutworth, DHN-bot, Xyzzy n, Mini-Geek, Luv2run, Pilotguy, J. Finkelstein, Soumya92, Strainu, Wickethewok, 16@r, Makyen, Hjfreyer, Pjrm, Polymerbringer, CBM, Chrisahn, Harrigan, Julian Mendez, C. Siebert, Martinkunev, MSBOT, Magioladitis, Algebra12320, David Eppstein, Raknarf44, Thomasda, Mqchen, Cquimper, VolkovBot, Davekaminski, SieBot, Jerryobject, Md. aftabuddin, Volkan YAZICI, Svick, AlanUS, Msg555, Justin W Smith, Yekaixiong, Nanobear, Daveagg, Greenmatter, SchreiberBike, Kletos, Maco1421, SteveJothen, Addbot, Beardybloke, Lightbot, Teles, Jarble, Leycec, Yobot, Roman Munich, AnomieBOT, Citation bot, ArthurBot, Nishantjr, Pvza85, Shadowjams, Frankrod44, Jixani, Stargazer7121, RobinK, Trappist the monk, Thái Nhi, Kanitani, Mwk soul, Specs112, Canuckian89, DrAndrewColes, Nostalgus, EmausBot, WikitanvirBot, Aenima23, AlexandreZ, Pxtrme75, Fatal1955, Shmor, ClueBot NG, KitMarlow, Anharrington, Dittymathew, Exercisephys, Sakanarm, Buaagg, Roseperrone, Jochen Burghardt, Shnowflake, Hemant19cse, Aureooms, Juancarlosgolos, Subshiri and Anonymous: 174
- **Johnson's algorithm** Source: <http://en.wikipedia.org/wiki/Johnson%20algorithm?oldid=564595231> Contributors: Charles Matthews, Altenmann, MarkSweep, Karl-Henner, Ruud Koot, Bob5972, Netvor, Gaius Cornelius, Josteinaj, Welsh, Cyrus Grisham, Danielx, AlexTG, Brahle, SmackBot, Octahedron80, Ligulembot, Algebra12320, David Eppstein, Gwern, VolkovBot, Abtinb, VVVBot, Kl4m, Nanobear, Addbot, CarsracBot, Jarble, Luckas-bot, Twexcom, AnomieBOT, Rubinbot, Citation bot, Drilnoth, Locobot, MicGrigni, Citation bot 1, MorganGreen, MondalorBot, EmausBot, ChuispastonBot and Anonymous: 21
- **Kruskal's algorithm** Source: <http://en.wikipedia.org/wiki/Kruskal%20algorithm?oldid=654784204> Contributors: LC, The Anome, Ap, Andre Engels, Shellreef, Wapcaplet, Poor Yorick, Hike395, Dcoetze, Dysprosia, Zero0000, Robbot, Jaredwf, YahoKa, R3m0t, Altenmann, MathMartin, Bkell, Marekpetrik, Giftlite, Levin, MarkSweep, Shen, Sarcelles, Chmod007, Treyshonuff, Oskar Sigvardsson, Robotje, Jonsafari, Runner1928, 4v4l0n42, HasharBot, Msh210, Michael.jaeger, Arthena, Spangineer, Kenyon, Kvikram, Boothy443, Kruskal, GregorB, Isnow, M412k, Qwertyus, Misrerou, AysZ88, FlaBot, Ecb29, Mathbot, Margosbot, Mmtux, YurikBot, Wavelength, Michael Sloane, Alcides, NawlinWiki, Gareth Jones, Mysid, Bota47, Klutz, Abu adam, LeonardoRob0t, DoriSmith, SmackBot, Mgreenbe, Jax Omen, Bluebot, Oli Filth, Cronholm144, Ycl6, Ahy1, Wakimakirolls, Shreyasjoshi, Simeon, Headbomb, Tokataro, Seaphoto, SiobhanHansa, Magioladitis, David Eppstein, Panarchy, CommonsDelinker, MRFraga, Jeri Aulurtve, Michael Angelkovich, Dzenanz, VolkovBot, JohnBlackburne, Jgarrett, CarpeCerevisi, Jamelan, Pranay Varma, Adamsandberg, Synthebot, SieBot, Phe-bot, OKBot, Denisarona, ClueBot, Justin W Smith, PixelBot, Abrech, Mikaey, Deineka, Addbot, DarrylNester, MrOllie, Maciek.nowakowski, Lightbot, Chipchap, Jarble, Luckas-bot, THEN WHO WAS PHONE?, AnomieBOT, JackieBot, NFD9001, BioTronic, Thehelpfulbot, FrescoBot, HJ Mitchell, Hhbs, Mitchellduffy, Pinethicket, Zetifree, Sanketpatel.301090, Orenburg1, EmausBot, Dixtosa, Eda eng, Kapil.xerox, Omargamil, ClueBot NG, Widr, Abc45624, Ahmad.829, Hammadhaleem, Happyuk, MiqayellMinasyan, Sigurd120, Aliok ao, Trunks175, IgushevEdward, YFdyh-bot, Mbarrenecheajr, Lone boatman, MindAfterMath, Gauravxpress, Monmnohas, Bantu syam, Schullz, EChamilakis, Sumit210 and Anonymous: 138
- **Dinic's algorithm** Source: <http://en.wikipedia.org/wiki/Dinic%20algorithm?oldid=652707333> Contributors: Michael Hardy, Giftlite, Andreas Kaufmann, Urod, Rjwilmsi, Octahedron80, Sytelus, Thijs!bot, Headbomb, JAnDbot, Magioladitis, Bbi5291, R'n'B, Sun Creator, NuclearWarfare, MystBot, Addbot, Tchashasaposse, Luckas-bot, Evergrey, Gawi, Gilo1969, Omnipaedista, X7q, EmausBot, Kasirbot, Helpful Pixie Bot, BG19bot, YFdyh-bot, Milicevic01 and Anonymous: 16
- **Edmonds–Karp algorithm** Source: <http://en.wikipedia.org/wiki/Edmonds%20algorithm?oldid=644830335> Contributors: Michael Hardy, Nixdorf, Poor Yorick, Zero0000, Robbot, Chopchopwhitey, Giftlite, Martani, Sesse, Pt, Simonfl, Cburnett, RJFJR, Parodox, Amelio Vázquez, Kristjan Wager, Mathbot, Hashproduct, YurikBot, Cquan, Gareth Jones, Nils Grimsmo, Htmnssn, Katieh5584, SmackBot, Nkojuharov, Pkirlin, Darth Panda, Mihai Capotă, Cosmi, Headbomb, JAnDbot, Magioladitis, Balloonguy, Gwern, Glrx, Aaron Hurst, Rei-bot, Jamelan, SieBot, Denisarona, Kubek15, Pugget, Pmezard, Addbot, DOI bot, WuBot, Luckas-bot, Yobot, LiuZhaoliang, Citation bot, ArthurBot, **فَلْيَزَادِكَانْ**, Kxx, Ohad trabelsi, TPReal, Jfmantis, John of Reading, WikitanvirBot, TuHan-Bot, ZéroBot, ClueBot NG, BG19bot, Nilofar pirooz, Zmwangx, Jmgibson3312 and Anonymous: 43
- **Ford–Fulkerson algorithm** Source: <http://en.wikipedia.org/wiki/Ford%20algorithm?oldid=654011174> Contributors: Hephaestos, Michael Hardy, Randywombat, Poor Yorick, Dcoetze, Almi, Zero0000, Jaredwf, Gandalf61, Babbage, Prara, Giftlite, Manuel Anastácio, CryptoDerk, Andreas Kaufmann, Treyshonuff, Benbread, Cburnett, Bobrayner, Knowledge-is-power, Dionyziz, Mammiling, Ecb29, YurikBot, Petter Strandmark, Gareth Jones, Nils Grimsmo, Toncek, Cedar101, SmackBot, DHN-bot, Neo139, Leland McInnes, EdGl, Nausher, Alex Selby, Riedel, Tawkerbot2, Harrigan, Myasuda, Arauzo, Cyhawk, Tawkerbot4, Tuna027, BetacommandBot, Thijs!bot, Heineman, Widefox, Beta16, JAnDbot, BrotherE, SiobhanHansa, Mange01, VolkovBot, Quiark, AlleborgoBot, Phe-bot, Gerel, OKBot, Svick, DFRussia, Colossus13, Mark1421, Dekart, Rrusin, Addbot, **پیارن**, JanKuipers, Legobot, Luckas-bot, Yobot, Ptbot-gourou, Materialscientist, Liorma, Shashwat986, Dinamik-bot, EmausBot, Tonymater, Kapil.xerox, JordiGH, Dopehead9, ClueBot NG, Island Monkey, Queeg, Manoguru, Mad728, Tfr.didi, Afshinzarei, Mc mosa, User 38, Wfbergmann, B1993alram, OlivePasta and Anonymous: 83
- **Push–relabel maximum flow algorithm** Source: <http://en.wikipedia.org/wiki/Push%20algorithm?oldid=647555713> Contributors: Damian Yerrick, Andreas Kaufmann, Rich Farmbrough, RussBlau, Dylan Thurston, Bgwhite, RussBot, Nils Grimsmo, SmackBot, Dicklyon, Tawkerbot2, Stefan Knauf, Headbomb, JAnDbot, Gwern, Terrek, Sunderland06, Debamf, Svick, MenoBot, Niceguyedc, Addbot, JanKuipers, Yobot, Citation bot, Kxx, FrescoBot, Shafaet, Sverigekullen, Slon02, EmausBot, AManWithNoPlan, Wmainer, Frijtjes, Nullzero, Helpful Pixie Bot, Babak.barati, Kcm1700, Drrill, Macofe, Sinha K, Kamac124, Adreliha and Anonymous: 40

- **Evolutionary algorithm** *Source:* <http://en.wikipedia.org/wiki/Evolutionary%20algorithm?oldid=653474897> *Contributors:* Magnus Manske, Netesq, Michael Hardy, Lexor, Cyde, Karada, Nine Tail Fox, Ronz, Extro, Kimiko, Samsara, JackH, TittoAssini, Stirling Newberry, Sobek, Duncharris, Gmlk, StephanRudlof, Srived, Jwdietrich2, Chrischan, Dbachmann, Alex Kosorukoff, Diomidis Spinellis, Markus.Waibel, The RedBurn, Diego Moya, Kotasik, Artur adib, TerminalPreppie, Mserge, Ruud Koot, Marudubshinki, Jannetta, Gaius Cornelius, RKUrsem, Mikeblas, Wjousts, Nojhan, SmackBot, Mneser, KocjoBot, Brick Thrower, Lorian, Apankrat, Oli Filth, Hongooi, Gragus, Uzb1t, Chlewbot, Lambiam, JHunterJ, Simiprof, Wleizer, Mr3641, George100, Cydebot, Peterdjones, Pruetboonma, Kborer, KrakatoaKatie, Dvunkannon, JAnDbot, Triponi, Ph.eyes, Djoshi116, Xeno, JamesBWatson, Tedickey, Sshrdp, Armehrabian, Auka, Gwern, MartinBot, Bissinger, Obscurans, M samadi, STBotD, Riccardopolis, JohnBlackburne, Lordvolton, Kjells, Jesin, ThomHImself, SieBot, Zwegem, Algorithms, Kumioko (renamed), Adrianwn, Ernstblumberg, Bahriyebasturk, Addbot, Tothwolf, MrVanBot, Luckas-bot, Yobot, Themfromspace, Pink!Teen, KamikazeBot, The locster, Armchair info guy, Paskorc, Andmats, Buenasdiaz, Aleph Infinity, Perada, Lh389, Mark Renier, Mohdavary, Boxplot, Luebuwei, Full-date unlinking bot, Jonkerz, Duoduoduo, Wo.luren, BertSeghers, Helwr, EmausBot, WikitanvirBot, Dzkd, Chire, Donner60, Honeybee-sci, ClueBot NG, Jr271, Evolvsolid, Alimirjalili, TheProfessor, Ferrari-sailor, Cerabot, Yamaha5, YiFeiBot, Paheld, Cottap, Boky90, Shahrzadsmd, Muaafa, Construct science and Anonymous: 117
- **Hill climbing** *Source:* <http://en.wikipedia.org/wiki/Hill%20climbing?oldid=650172537> *Contributors:* Shd, ChangChienFu, Frecklefoot, Docu, LittleDan, Poor Yorick, Hike395, Furrykef, McKay, Sander123, Chopchopwhitey, DavidCary, Stern, Pgan002, Frau Holle, Freakofnurture, Loganberry, Number 0, Bobo192, Billymac00, Drangon, Diego Moya, Cburnett, RainbowOfLight, Oleg Alexandrov, WadeSimMiser, Deepstratagem, Qwertus, Strait, FlaBot, Eubot, Mathbot, YurikBot, Taejo, Stephenb, QmunkE, Ankurdave, SmackBot, Mcld, Nbarth, L337p4wn, BrownHairedGirl, Daniel5127, Cydebot, Majid ahmadi, KrakatoaKatie, Isilanes, Pichote, JAnDbot, Meredyth, Father Goose, David Eppstein, Piojo, KamiFireheln, K.menin, SJP, Jjljd, VolkovBot, Aaron Rotenberg, Sapphic, Svick, CharlesGillingham, Headlessplatter, ClueBot, Yamakiri, Hello Control, PixelBot, Paradiso, Addbot, MrOllie, Luckas-bot, Amirobot, Kingpin13, Materialscientist, Hullo exclamation mark, Kamitsaha, Citation bot 1, Pinethicket, Ficuep, Dreske, Tommy2010, ClueBot NG, Laughsinthestocks, Marcuswikipedia, Happyuk, Lone boatman, Demagun and Anonymous: 81
- **Local search (optimization)** *Source:* [http://en.wikipedia.org/wiki/Local%20search%20\(optimization\)?oldid=638328121](http://en.wikipedia.org/wiki/Local%20search%20(optimization)?oldid=638328121) *Contributors:* Michael Hardy, Angela, Charles Matthews, Hh, Aliekens, Sander123, Urhixidur, Nowozin, Diomidis Spinellis, Arthena, Diego Moya, Versageek, Oleg Alexandrov, Mindmatrix, LOL, Ruud Koot, Jorunn, Rjwilmsi, Tizio, Theorem, Mathbot, Hm2k, Silvery, RupertDick, Tribaal, Davepape, Blixathetcat, David Poole, Commander Keane bot, Amine Brikci N, Ckatz, Cydebot, Thij's!bot, Bobblehead, Seaphoto, Antelan, Father Goose, David Eppstein, Infovarius, Sharp Tac, WikiStrider, Banerjia, Addbot, MrOllie, Luckas-bot, Yobot, Amirobot, Isheden, GrouchoBot, Kiefer.Wolfowitz, Dinamik-bot, Miracle Pen, Mean as custard, EmausBot, Dvpx, Optimering, SporkBot, Jander-mile, ClueBot NG, Leonardo61, Toniojj, Jr271, Marcuswikipedia, Marcos Baker, Monkbot and Anonymous: 35
- **Simulated annealing** *Source:* <http://en.wikipedia.org/wiki/Simulated%20annealing?oldid=646321023> *Contributors:* Bryan Derksen, The Anome, Taw, Shd, Arvindn, Waveguy, Heron, Edward, Michael Hardy, Kku, Tomagenet, Cyp, Ronz, Hike395, Dcoetze, Jitse Niesen, Rls, Robbot, Jaredwf, Chocolateboy, Stewartadcock, Aniu, Brw12, Alex R S, Moink, Cutler, Giftlite, Jyrl, Jorend, Jorge Stolff, Ferdinandus, KaHa242, Beland, NathanHurst, Rich Farmbrough, DocMax, Nickj, Smalljim, Diego Moya, Hu, Pontus, HenkvD, Jheald, Kinema, Mpdehnel, Oleg Alexandrov, Woohookitty, Ruud Koot, Norro, Qwertus, Rjwilmsi, Wvangeit, Miskin, FlaBot, Ian Pitchford, Mathbot, CiaPan, Chobot, Whosasking, Chris Capoccia, Gaius Cornelius, Jugander, Retired username, Ospalh, Petri Krohn, Vicarious, Cffrost, Itub, SmackBot, Oli Filth, Conway71, Tamfang, SeanAhern, Mshivers, Brunato, Minna Sora no Shita, ManiF, Jim.belk, Fer-audyh, Experiment123, CRGreathouse, Menkaur, Ingber, Cydebot, Pureferret, Markluffel, Gimmetrow, Thij's!bot, Kablammo, Arnab das, KrakatoaKatie, Михајло Анђелковић, AnAj, Itistoday, Magioladitis, Bruceporteous, Meredyth, DWeissman, Armehrabian, Francob, James mcl, R'n'B, Stochastics, Alfran, Nthomas4, K.menin, Arbec, Ryanbrooks, BrianOfRugby, JonMcLoone, Mobeets, Red Act, Nxavar, Broadbot, Mundhenk, Khurramnazir, BotMultichill, VVVBot, Rinconsoleao, Chardson, Stopstopstopstop, Andreas Schuderer, Calimo, Arjayay, Dgwillet, Gropoli, XLinkBot, Koumz, Cmr08, Tinyrock, Addbot, Ryanne Dolan, MrOllie, Chillydawg, Chipchap, Sergioledesma, Yobot, E mraedarab, AnomieBOT, Kingpin13, Materialscientist, Citation bot, P99am, Control.valve, Miym, MuffledThud, Eugene-elgato, Captain Fortran, FrescoBot, Citation bot 1, Brambleclawx, Wyverald, RjwilmsiBot, BertSeghers, EmausBot, WikitanvirBot, Flamerrecca, Rashedi es, Slawekb, Daryakov, Optimering, SporkBot, GeOffrey de smet, ChuispastonBot, ClueBot NG, Editdorigo, CocuBot, Leonardo61, ChristophE, Frietjes, Barpa, Ishamdo, SciCompTeacher, Happyuk, Mdann52, ChrisGualtieri, IjonTichyIjonTichy, Lone boatman, SFK2, Obeeflet, Mdbauer, RickyFeynman, R robotics, Mostafa-azami, Geb0541 and Anonymous: 205
- **Tabu search** *Source:* <http://en.wikipedia.org/wiki/Tabu%20search?oldid=655308606> *Contributors:* The Anome, Heron, Kku, Karada, Haakon, Ronz, Disdero, Charles Matthews, Gangadhar, Secretlondon, Altenmann, Kenneth.sorensen, Mboverload, Gdr, DragonflySixtyseven, Dostal, Diego Moya, Pion, Japanese Searobin, Pixeltoo, AshishG, Rjwilmsi, Tizio, Kwaxi, FlaBot, YurikBot, RobotE, Hairy Dude, Elkman, Itub, Jasonb05, Ser Amantio di Nicolao, Pangiya, Cydebot, KrakatoaKatie, Dricherby, Meredyth, David Eppstein, Slvnpxc, JayApril, Francob, NerdyNSK, K.menin, Bonadea, Epktsang, Brech, Gomgr, Tabusearchtutorial, Mumiemonstret, Hello Control, XLinkBot, Paradiso, Junji1337, Addbot, MrOllie, Greg4cr, Luckas-bot, Yobot, Miym, Ijustloveit13, RjwilmsiBot, Hba2, EmausBot, WikitanvirBot, Daryakov, GeOffrey de smet, Wikihal, Robiminer, Leonardo61, Jr271, BG19bot, Anubhab91, JYBot, Qsq, Fredwglover, Colpain, Alex Pavay and Anonymous: 59
- **Active set method** *Source:* <http://en.wikipedia.org/wiki/Active%20set%20method?oldid=595647469> *Contributors:* Jitse Niesen, Ajgorhoe, NathanHurst, Oleg Alexandrov, Bluemoose, Qwertus, Bryan Barnard2, Tribaal, SmackBot, Cydebot, Thij's!bot, Fbahr, Hellno2, Cberry01, Addbot, Luckas-bot, AnomieBOT, Ciphers, Citation bot 1, Kiefer.Wolfowitz, Duoduoduo, True bsmile, Alex.j.flint and Anonymous: 5
- **Adaptive coordinate descent** *Source:* <http://en.wikipedia.org/wiki/Adaptive%20coordinate%20descent?oldid=615959693> *Contributors:* Dekart, Sionk, Trappist the monk, Mark Arsten, ArticlesForCreationBot, Evolutionarycomputation, Riley Huntley and Anonymous: 3
- **Alpha-beta pruning** *Source:* <http://en.wikipedia.org/wiki/Alpha%E2%80%93beta%20pruning?oldid=654887540> *Contributors:* Mav, The Anome, Arvindn, Michael Hardy, Delirium, Bart Massey, Dcoetze, Doradus, Furrykef, Thesilverbail, Rasmus Faber, Hadal, Wikibot, Giftlite, ScudLee, Flok, Wolfkeeper, Bfnn, Steven jones, Beland, Kogorman, Avatar, Asqueella, Evand, Nomis80, BlueNovember, Blahedo, David Haslam, GregorB, Qwertus, Rjwilmsi, Eptalon, Intgr, Fresheneesz, Pete.Hurd, Kri, RussBot, Michael Slone, Msikma, Gnusbiz, Hirak 99, Cedar101, Nelhage, MathsIsFun, Willtron, Thijswijs, SmackBot, Eskimbot, Betacommand, Onorem, Zophar1, Janzert, Luke Gustafson, Dicklyon, Plattler01, KJS77, Stephen B Streater, Eric Le Bigot, Bumbulski, AndrewHowse, Cydebot, Jecraig@yahoo.com, Headbomb, AllUltima, Heineman, Makaimc, JAnDbot, IanOsgood, Magioladitis, Albmont, David Eppstein, Jez9999, Circular17, Cobi, DorganBot, Kriskra, VolkovBot, Veganfanatic, Pitel, Euryalus, Lightmouse, Truehalley, CharlesGillingham, Melcombe, Kvihill, Dsnpst, SoxBot III, DOI bot, Maschelos, Alexrakia, LinkFA-Bot, Newfraferz87, Tide rolls, OJEnglish, Yobot, DavdB, Puhfyn, Citation bot, GrouchoBot, Ranmocy, Sophus Bie, BenzolBot, Stephen Morley, Citation bot 1, Kiefer.Wolfowitz, Libor Vilímek, RjwilmsiBot, MiNi-WoLF, John of Reading, Arkenflame, H3llBot, Bad Romance, ClueBot NG, Incompetence, Pengyifan, Hahahaha, Swierczek, KLBot2, Jose.m.c.torres, Dexbot, Lancebop, Pipemore, GamePlayerAI, AzrgExplorers, Monkbot and Anonymous: 140

- **Artificial bee colony algorithm** *Source:* <http://en.wikipedia.org/wiki/Artificial%20bee%20colony%20algorithm?oldid=653669366> *Contributors:* Michael Hardy, Andreas Kaufmann, Diego Moya, Leondz, Ruud Koot, Rjwilmsi, Tony1, Smooth O, Courcelles, Cydebot, JamesR, Jiguang Wang, K.menin, Truthanado, WRK, Bahriyebasturk, DumZiBoT, WikHead, Addbot, Fluffernutter, Yobot, Buddy23Lee, Minimac, BG19bot, Michaelmalak, Eugenecheung, SPhotographer, Tymnt410, Notsoimp2012 and Anonymous: 23
- **Auction algorithm** *Source:* <http://en.wikipedia.org/wiki/Auction%20algorithm?oldid=644580056> *Contributors:* Edward, Andreas Kaufmann, LutzL, Cydebot, Wikid77, Trevormwood, Addbot, MrOllie, LilHelpa, FrescoBot, Dpbert, Bpdmit, Helpful Pixie Bot, AHusain314, Xmarynka and Anonymous: 2
- **Automatic label placement** *Source:* <http://en.wikipedia.org/wiki/Automatic%20label%20placement?oldid=641634243> *Contributors:* Dreamyshade, Taw, Arvindn, Heron, Natevw, Robbot, Kmote, Sligocki, Oleg Alexandrov, Woohookitty, YurikBot, Rdnewman, CLW, SmackBot, Kazkaskazkasako, CmdrObot, Cydebot, KrakatoaKatie, R27182818, David Eppstein, NetXpert, ClueBot, Andy pyro, MapXpert, Sun Creator, Addbot, Fgnievinski, Erel Segal, DrilBot, Nobleaxis, Helpful Pixie Bot, Andreschulz and Anonymous: 13
- **Bees algorithm** *Source:* <http://en.wikipedia.org/wiki/Bees%20algorithm?oldid=596896061> *Contributors:* Michael Hardy, Bkell, Giftlite, BozMo, Andreas Kaufmann, Bender235, Jonathan Drain, Diego Moya, Pontus, Ruud Koot, Rjwilmsi, Tedder, Rjlabs, Epipelagic, SmackBot, Jasonb05, DO11.10, Beetstra, Harej bot, Cydebot, JAnDbot, MGarcia, Armehrabian, Cpl Syx, Neeku, K.menin, TJRC, ImageRemovalBot, Cardiffbaybee, XLinkBot, Addbot, MrOllie, Yobot, MuffledThud, Buddy23Lee, Jonkerz, DotKuro, Daryakov, Krystofer, AvicBot, Ego White Tray, Honeybee-sci, MerlIwBot, BG19bot, SPhotographer, Marco castellani 1965 and Anonymous: 24
- **Benson's algorithm** *Source:* <http://en.wikipedia.org/wiki/Benson'{}s%20algorithm?oldid=628188083> *Contributors:* Jitse Niesen, Zfeinst and Anonymous: 1
- **Berndt–Hall–Hall–Hausman algorithm** *Source:* <http://en.wikipedia.org/wiki/Berndt%E2%80%93Hall%E2%80%93Hall%E2%80%93Hausman%20algorithm?oldid=625247931> *Contributors:* Michael Hardy, Skysmith, Rich Farmbrough, Bender235, Dirknachbar, Retired username, Werdna, Sadads, Berland, Nutcracker, Cydebot, Postcard Cathy, WRK, Melcombe, Razorflame, Qwfp, AdamXtreme18, Yobot, AnomieBOT, Erik9bot, Difu Wu, Xg.su and Anonymous: 4
- **Big M method** *Source:* <http://en.wikipedia.org/wiki/Big%20M%20method?oldid=633032335> *Contributors:* Michael Hardy, Fintor, Rich Farmbrough, Lockley, Stormbay, SmackBot, Cydebot, K.menin, Toddst1, Denisarona, HairyD, Justin W Smith, Qwfp, Addbot, KamikazeBot, Zfeinst, Helpful Pixie Bot and Anonymous: 7
- **Bin packing problem** *Source:* <http://en.wikipedia.org/wiki/Bin%20packing%20problem?oldid=644668528> *Contributors:* Damian Yerrick, Michael Hardy, Docu, Dcoetze, Dysprosia, Greenrd, Hyacinth, Grendelkhan, Fibonacci, Donarreiskoffer, Giftlite, EmilJ, Rbirmann, K3rb, Tournesol, Wgsimon, Rjwilmsi, FlaBot, Finell, SmackBot, Oli Filth, EncMstr, Tompsc, Ligulembot, Crenshaw, Ocatecir, Pjrm, Colinj, CRGreathouse, Amalas, Nczempin, Cydebot, Geopoul, CharlotteWebb, David Eppstein, R'n'B, Tarotcards, Ken g6, IBitOBear, Borat fan, Daviddoria, StevenBell, Jan Winnicki, Chromaticity, DeXXus, Cngoulimis, Jayelson, Ost316, Gjacquenot, Addbot, Favonian, Yobot, Wiki5d, Marioxcc, Citation bot, Buenasdiaz, Twri, Doremo, Karakfa, Gshaham, Jiri.demel, RjwilmsiBot, DenisNazarov, Ein student, Ksoltys, MerlIwBot, Helpful Pixie Bot, Mmonaci, BG19bot, Tzhai, FooTheBar, Mark viking, Grad2grad, Almeria.raul, Monkbot, Berechnung and Anonymous: 63
- **Bland's rule** *Source:* <http://en.wikipedia.org/wiki/Bland'{}s%20rule?oldid=623178780> *Contributors:* Michael Hardy, Orangemike, C S, Naasking, Sneftel, Cydebot, Headbomb, Apavlo, Tinucherian, David Eppstein, Nsk92, Leofric1, Cold Phoenix, Dawynn, Omnipaedista, Kiefer.Wolfowitz, RjwilmsiBot, AManWithNoPlan, Zfeinst, ClueBot NG, Monkbot, Tyrneaeplith and Anonymous: 10
- **BOBYQA** *Source:* <http://en.wikipedia.org/wiki/BOBYQA?oldid=655208745> *Contributors:* Cnilep and Zhangzk
- **Branch and price** *Source:* <http://en.wikipedia.org/wiki/Branch%20and%20price?oldid=638789017> *Contributors:* Michael Hardy, Rjwilmsi, Cydebot, Magioladitis, David Eppstein, Heheman3000, Kiefer.Wolfowitz, Trappist the monk, Crosstemplejay, Bmears11, Monkbot and Anonymous: 2
- **CMA-ES** *Source:* <http://en.wikipedia.org/wiki/CMA-ES?oldid=639946420> *Contributors:* Edward, Jitse Niesen, Phil Boswell, Mandarax, Rjwilmsi, FlaBot, Thiseye, Malcolma, Ses, Elonka, EmreDuran, CBM, Cydebot, Magioladitis, Obscurans, K.menin, Jamesontai, Frank.Schulz, Dhatfield, Melcombe, Tangonacht, Sentewolf, Mild Bill Hiccup, Addbot, Yobot, Citation bot, GrouchoBot, FrescoBot, Citation bot 1, Optimering, SporkBot, Tomschaul, ClueBot NG, TheProfessor, Wikipedia wii man, Zhangzk, Mouse7mouse9, YAkimoto, Monkbot and Anonymous: 73
- **COBYLA** *Source:* <http://en.wikipedia.org/wiki/COBYLA?oldid=655207521> *Contributors:* Michael Hardy, Jitse Niesen, Qwertus, Cpbl, David Eppstein, AnomieBOT, Zhangzk and Anders9ustafsson
- **Coffman–Graham algorithm** *Source:* <http://en.wikipedia.org/wiki/Coffman%E2%80%93Graham%20algorithm?oldid=607758272> *Contributors:* Rjwilmsi, Cydebot, David Eppstein, Sun Creator, Yobot, DutchCanadian, ClueBot NG, Helpful Pixie Bot and Anonymous: 1
- **Column generation** *Source:* <http://en.wikipedia.org/wiki/Column%20generation?oldid=624813613> *Contributors:* The Anome, Michael Hardy, Hike395, Almi, Alotau, Reuben, Diego Moya, Oleg Alexandrov, Jeff3000, Mathbot, SarahRoot, Eskimbot, Kjetil1001, CBM, Cydebot, AntiVandalBot, K.menin, Mr nipples, Nith Sahor, AnomieBOT, Erik9bot, Kiefer.Wolfowitz, KLBot2, Qetuth and Anonymous: 7
- **Constructive cooperative coevolution** *Source:* <http://en.wikipedia.org/wiki/Constructive%20cooperative%20coevolution?oldid=647564396> *Contributors:* Yobot, BattyBot, StarryGrandma, EmileGlorieux and SkateTier
- **Crew scheduling** *Source:* <http://en.wikipedia.org/wiki/Crew%20scheduling?oldid=583580457> *Contributors:* Michael Hardy, Tabletop, Vegaswikian, Ahunt, CmdrObot, Cydebot, JamesBWatson, Wikip rhyre, Yf metro, Falcon8765, Kpaynter, Badgernet, ZooFari, Jncraton, MrOllie, Ivasju, LilHelpa, Alvin Seville, Jesse V., DexDor, John of Reading, Ghealy59, Deepakmrl, TruckCard, JJULEN92, Jørdan, Shire Reeve, Hallerin, Jalaxi, MiNicole and Anonymous: 19
- **Cross-entropy method** *Source:* <http://en.wikipedia.org/wiki/Cross-entropy%20method?oldid=641162149> *Contributors:* Michael Hardy, Jitse Niesen, Mebden, Jasonb05, PierRRoMaN, David Cooke, Cydebot, KrakatoaKatie, Kroese, Sapphic, Addbot, MrOllie, Luckas-bot, Twri, FrescoBot, Shuroo, Gabap, Pm.prashant and Anonymous: 7
- **Cuckoo search** *Source:* <http://en.wikipedia.org/wiki/Cuckoo%20search?oldid=651803820> *Contributors:* Edward, Michael Hardy, Phil Boswell, Andreas Kaufmann, Bender235, Diego Moya, Peerst, Ruud Koot, Tabletop, Macaddct1984, Rjwilmsi, TeaDrinker, Wavelength, Pburka, Serg3d2, Ealdent, Myasuda, Cydebot, Headbomb, Utopiah, Magioladitis, K.menin, Saxyclarinette, Addbot, Favonian, Tassedethe,

Yobot, AnomieBOT, Aschlosberg, Metafun, Psogeek, Fæ, Cuckoolover, Levyflight, Petriot333, Skeduling, Matlabshow, Frietjes, BG19bot, Notsilly comments, Birdieee, Gamingconf, Softwaree, TimeMuffin, Pennycuckoo, Daylearner, Islandic, Poincaree, Birdiee, Unlimiteed, Salvaluee, Knaapp, LéarnLévy, Szczalg, Krowlod, Léckß, Veniabeu, Compsim, عشوائيّة, Αλγόριθμος, ଶାଖକ୍ରିତ୍, Knowlge, Ginsuloft, Monkbot, Kotteswaran, Notsoimp2012 and Anonymous: 15

- **Derivation of the conjugate gradient method** *Source:* <http://en.wikipedia.org/wiki/Derivation%20of%20the%20conjugate%20gradient%20method?oldid=590981270> *Contributors:* Chris Howard, Cydebot, K.menin, Addbot, Luckas-bot, CES1596, Kxx, Going-Batty, Helpful Pixie Bot, Monkbot and Anonymous: 3
- **Derivative-free optimization** *Source:* <http://en.wikipedia.org/wiki/Derivative-free%20optimization?oldid=650586257> *Contributors:* Asimkumar2222, Vinzklorthos, Michael.Clerx, Cnilep, Yobot, AnomieBOT, Zhangzk, Indiuser and Anonymous: 2
- **Destination dispatch** *Source:* <http://en.wikipedia.org/wiki/Destination%20dispatch?oldid=631918504> *Contributors:* Michael Hardy, Ospalh, SmackBot, Cydebot, RichardVeryard, Albany NY, Largoplazo, Surfthetsu, Yobot, AnomieBOT, Virtualdispatcher, PeterEastern, Tamariki, I am One of Many, Stevejboisvert, Michael Lilja and Anonymous: 6
- **Differential evolution** *Source:* <http://en.wikipedia.org/wiki/Differential%20evolution?oldid=654802241> *Contributors:* Michael Hardy, Andreas Kaufmann, Discospinster, Rich Farmbrough, Diego Moya, Oleg Alexandrov, Alkarex, Robert K S, Ruud Koot, Rjwilmsi, Mathbot, NawlinWiki, SmackBot, RDBury, MidgleyDJ, Hongooi, Jasonb05, Guroadrunner, Mishrasknehu, Cydebot, KrakatoaKatie, Dvunkanon, Liquid-aim-bot, Ph.eyes, Calltech, R'nB, J.A. Vital, Athaenara, K.menin, STBotD, Jamesontai, TXiKiBoT, Kjells, SieBot, Wmpearl, D14C050, Esa-petri, Fell.inchoate, Sun Creator, SchreiberBike, XLinkBot, Addbot, MrOllie, Chipchap, Luckas-bot, Jorge.maturana, Am-inrahimian, Scribbleink, Optimering, SporkBot, EdoBot, BG19bot, Lilingxi, Sharkyangliu916, Monkbot, Jacob.Wilfridovich and Anonymous: 34
- **Divide and conquer algorithms** *Source:* <http://en.wikipedia.org/wiki/Divide%20and%20conquer%20algorithms?oldid=642697167> *Contributors:* XJaM, Pnm, Kku, TakuyaMurata, Looxix, Stevenj, Poor Yorick, Dcoetze, Daniel Quinlan, Furrykef, AndrewKepert, Head, Fredrik, Adam78, Giftlite, Jorge Stolfi, LiDaobing, Phe, Mycompsimm, El C, Bobo192, R. S. Shaw, Neg, Nsaa, Mdd, Varuna, ThePedanticPrick, MIT Trekkie, Finem, Linas, Madmardigan53, Ruud Koot, WadeSimMiser, Qwertys, Bhadani, VKokielov, DevastatorIIC, Tardis, Taichi, Chobot, Roboto de Ajvol, YurikBot, Amshali, Irrevenant, Dtrebbien, Goffrie, BOT-Superzerocool, Josh3580, Garion96, Thijswijs, SmackBot, Unyoyega, Atomota, Bluebot, Nbarth, Mlpkr, Vina-iwbot, SashatoBot, G-Bot, BrownHairedGirl, Jake-helliwell, Tawkerbot2, Destructor, Nczempin, Cydebot, Jfcorbett, IrishPete, JAnDbot, anacondabot, Magioladitis, David Eppstein, Pomte, J.delanoy, LokiClock, Philip Trueman, Rei-bot, LanceBarber, Wolfrock, SieBot, Antzervos, OKBot, Excirial, Quercus basaseachicensis, Bob man801, Thingg, Pichpitch, DragonFury, Addbot, Mortense, CL, Jarble, Luckas-bot, Yobot, AnomieBOT, Ailun, Citation bot, LilHelpa, Xqbot, Ekaratsuba, Almabot, Algoritmist, Prunesqualer, UTQ Shadow, Citation bot 1, Boxplot, Trappist the monk, ArbitUsername, Bluefist, PowerPaul86, EmausBot, Tolly4bolly, Bomazi, ChuipastonBot, Petrb, ClueBot NG, Widr, Riteshrit20, Nullzero, Bloghog23, Monkbot and Anonymous: 85
- **Dykstra's projection algorithm** *Source:* <http://en.wikipedia.org/wiki/Dykstra%27s%20projection%20algorithm?oldid=609671939> *Contributors:* Michael Hardy, Rjwilmsi, Lavaka, Rocchini, Svick, Jim1138, Shire Reeve, ChrisGualtieri and Anonymous: 2
- **Eagle strategy** *Source:* <http://en.wikipedia.org/wiki/Eagle%20strategy?oldid=632665183> *Contributors:* Bearcat, Ruud Koot, Lockley, Toddst1, SchreiberBike, Good Olfactory, Yobot, Cavarrone, Starcheerspeaknewslostwars, Идтиорел, Gyrov, عقاب, ئەلەن, Velgengni, إسـتراتيـجيـة and Anonymous: 2
- **Evolutionary programming** *Source:* <http://en.wikipedia.org/wiki/Evolutionary%20programming?oldid=639946759> *Contributors:* Karada, Jitse Niesen, Samsara, Alan Lifting, Psb777, Melaan, Gadig, YurikBot, Dql, SmackBot, Mira, Bluebot, Tobym, Pgr94, Cydebot, Thijs!bot, Sm8900, Soho123, Tsaitgaist, Algorithms, CharlesGillingham, ClueBot, Addbot, Luckas-bot, Rubinbot, Sergey539, Customline, Hirsutism, Pooven, Sahimrobot, TheProfessor and Anonymous: 12
- **Expectation–maximization algorithm** *Source:* <http://en.wikipedia.org/wiki/Expectation%E2%80%93maximization%20algorithm?oldid=653898112> *Contributors:* Rodrigob, Michael Hardy, Karada, Jrauser, BAxelrod, Hike395, Phil Boswell, Owenman, Robbyjo, Ben-wing, Wile E. Heresiarch, Giftlite, Paisa, Vadmium, Onco p53, MarkSweep, Piotrus, Cataphract, Rama, MisterSheik, Alex Kosorukoff, O18, John Vandenberg, Jjmerelo, 3mta3, Terrycojones, B k, Eric Kvaalen, Cburnett, Finfobia, Jheald, Forderud, Sergey Dmitriev, Igny, Bkkbrad, Bluemoose, Bytyner, Qwertys, Rjwilmsi, KYPark, Salix alba, Hild, Mathbot, Glopk, Kri, BradBeattie, YurikBot, Nils Grimsmo, Schmock, Régis B., Klutzy, Hakeem.gadi, Maechler, Ladypine, M.A.Dabbah, SmackBot, Meld, Nbarth, Tekhnofiend, Iwaterpolo, Bil-grau, Joeyo, Raptur, Derek farn, Jrouquie, Dicklyon, Alex Selby, Saviourmachine, Lavaka, Requestion, Cydebot, A876, Kallerdis, Libro0, Blaisorblade, Skittleys, Andyrew609, Talgalili, Tiedyeina, Rusmike, Headbomb, RobHar, AnAj, Zzpmarco, Dekimasu, JamesB-Watson, Richard Bartholomew, Livingthingdan, Nkwatra, User A1, Edratzer, Osquar F, Numbo3, Salih, GongYi, Douglas-Lanman, Bi-gredbrain, Market Efficiency, Lamro, Daviddoria, Pine900, Tambal, Mosaliganti1.1, Melcombe, Sitush, Pratx, Alexbot, Hbeigi, Jakarr, Jwmarck, XLinkBot, Jamshidian, Addbot, Sunjuren, Fgnievinski, LaaknorBot, Aanthon1243, Peni, Luckas-bot, Yobot, LeonardoWeiss, AnomieBOT, Citation bot, TechBot, Chuanren, FrescoBot, Nageh, Erhanbas, Nocheenlatierra, Qiemen, Kiefer.Wolfowitz, Jmc200, Stpasha, Jszyomon, GeypycGn, Trappist the monk, Thái Nhi, Ismailari, Dropscienctobombs, RjwilmsiBot, Slon02, EmausBot, Mikealander, John of Reading, III, Chire, Statna, ClueBot NG, Rezabot, Meea, Qwertys9967, Helpful Pixie Bot, Rxnt, Bibcode Bot, BG19bot, Chafe66, Whym, Lvlinis, BattyBot, Yasuo2, Illia Connell, JYBot, Blegat, Yogtad, Tentinator, Marko0991, Ginsuloft, Wccsnow, Ron-niemaor, Monkbot, Nbole, Faror91, DilumA, Rider ranger47, Velvel2 and Anonymous: 148
- **Extremal optimization** *Source:* <http://en.wikipedia.org/wiki/Extremal%20optimization?oldid=534197044> *Contributors:* Michael Hardy, Karada, Johndarrington, Oli Filth, Hongooi, Jasonb05, Cydebot, Christian75, KrakatoaKatie, K.menin, Sliders06, MrOllie and Anonymous: 5
- **Fernandez's method** *Source:* <http://en.wikipedia.org/wiki/Fernandez%20method?oldid=654694857> *Contributors:* Bearcat, Discospinster, Vegaswikan, NawlinWiki, Biscuittin, StAnselm, C. lorenz, Yobot, AnomieBOT, BattyBot and RockerZ007
- **Firefly algorithm** *Source:* <http://en.wikipedia.org/wiki/Firefly%20algorithm?oldid=651803413> *Contributors:* Michael Hardy, Andreas Kaufmann, Bender235, Diego Moya, Ruud Koot, Rjwilmsi, DavidConrad, Epipelagic, Cedar101, Nbarth, Glenbarnett, Myasuda, Cydebot, Headbomb, Utopiah, Destynova, EagleFan, K.menin, WereSpielChequers, SoheiMajin Gotenks, Shem1805, Dekart, Tassedethe, Legobot, Yobot, AnomieBOT, FrescoBot, Algorithmgeek, Scifun, Lovefirefly, Vectorqu, Loopso, Helpful Pixie Bot, Analyser2010, BG19bot, Flash-ingbug, RoysocA, Michaelmalak, Simarkov, Clusteering, Fireflasher, Atomistikrun, Wolvson, Radioattenna, Saeid1360, Swarmteam, PrinEe, Nobleimager, Snow Blizzard, Igorso, Bazinge, Qq101, Létat, 沈祖堯, Eugenecheung, Brilantaj, Algoritmus, Jiejie9988, Eduardofpl, Schwatzwutz, Ουδέτερος, WikiforEA, بروتین, Notsoimp2012 and Anonymous: 29

- **Fourier–Motzkin elimination** *Source:* <http://en.wikipedia.org/wiki/Fourier%20%80%93Motzkin%20elimination?oldid=634271040> *Contributors:* Darkwind, Jitse Niesen, David.Monnaux, Phil Boswell, Phatsphere, Saforrest, Macrakis, Bender235, Joriki, Rjwilmsi, Rwalker, CmdrObot, Cydebot, JamesBWatson, David Eppstein, Xyz9000, K.menin, AndreasJSbot, Philmac, Nielsthl, Flyer22, Simon04, Acabashi, Addbot, Baffle gab1978, Yobot, AnomieBOT, Materialscientist, Citation bot, Twiri, EmausBOT, Slawekb, Michaelnikolaou, ClueBot NG, Helpful Pixie Bot, AustinBuchanan, Sumguy5k, Horus 65, DarwinianLoser, Alderzdev, NMICK64, Awais-gillani, Threefournine, Satyajeetpradhanorissa, HaxxorZ596, MNatural, MAXooodKhan, Vrejbusiness, Aidan j Lawson, Kejri, Kirantrans, Almallory3, Kushagra khare12, ThatplaceinMali, Mikeandmike18, Tinavicks and Anonymous: 12
- **Generalized iterative scaling** *Source:* <http://en.wikipedia.org/wiki/Generalized%20iterative%20scaling?oldid=638345596> *Contributors:* Michael Hardy, Qwertys, Rjwilmsi, Wavelength, David Eppstein and Mitch1610
- **Genetic algorithm** *Source:* <http://en.wikipedia.org/wiki/Genetic%20algorithm?oldid=655247696> *Contributors:* Magnus Manske, The Epoft, Taw, Piotr Gasiorowski, Vignaux, William Avery, Waveguy, Edward, Simonham, Michael Hardy, Soegoe, Kwertii, Felsenst, Lexor, David Martland, AdamRaizen, Shyamal, Kku, Bobby D. Bryant, Ahoerstemeier, Ronz, Ijon, Qed, Marco Krohn, Kyokpae, Nikai, Evercat, BAxelrod, Smack, Hike395, Disdoro, Timwi, Jitse Niesen, Greenrd, Steinsky, Furrykef, Grendelhan, AnthonyQBachelor, Alikeens, Robbot, Chocolateboy, Arkuat, Chopchopwhitey, Stewartadcock, Texture, Hippietrail, Mark Krueger, Unyounyo, Centrx, Giftlite, Sankar netsoft, Jyril, Lee J Haywood, Stefano KALB, Curps, Duncharris, Utcursch, KaHa242, Antandrus, Oneiros, Aabs, Asbestos, Hellisp, Goobergunch, SamuelScarano, Andreas Kaufmann, RevRagnarok, Jwdietrich2, AAAAA, Rfl, Vincom2, 2fargon, Yinon, Cap'n Refsmmat, DerrickCheng, Alex Kosorukoff, Spoon!, ..Avjol.., Malafaya, Giraffedata, Cpcjr, Mdd, Phyzome, Larham, Alansohn, Silver hr, Crispin Cooper, BryanD, Hu, Brinkost, CloudNine, Artur adib, Zawersh, Oleg Alexandrov, Postrach, Stuartyeates, Ruud Koot, Jeff3000, GregorB, Dionyziz, Male1979, VsevolodSipakov, Jwoodger, Qwertys, Terryn3, Avinesh (usurped), Tailboom22, CoderGnome, Kane5187, Rjwilmsi, Loudenvier, MikeMayer, Arbor, Mpo, Vietbio, Mathbot, Ewlyahooocom, Diza, Kri, Chobot, Bgwhite, Manu3d, Dúnadan, YurikBot, Wavelength, RussBot, Freiberg, Bhny, Chris Capoccia, Stephenb, Gaius Cornelius, Bovineone, Madcoverboy, LM-Schmitt, RKUrsem, Toncek, Mikeblas, Brat32, Bota47, Pegwash, Tribaal, Twelvethirteen, Open2universe, Lt-wiki-bot, Arthur Rubin, Janto, GraemeL, CWenger, QmunkE, Wjousts, Allens, A.Nath, Xiaojeng, Edin1, Pecoraj, SmackBot, Tarret, InverseHypercube, Brick Thrower, Eskimbot, Mikolaj Koziarkiewicz, Sundaryourfriend, Oli Filth, Conway71, MalafayaBot, Nosophorus, DHIN-bot, Mohan1986, Ludvig von Hamburger, Simpsons contributor, Jasonb05, Gragus, Chlebot, MattOates, Parent5446, Spacecaldwell, Robma, Radagast83, "alyosha", Acdx, Bidabadi, Kuzaar, Thomas weise, Antonioli, Jcmiras, Beetstra, Negruvio, DabMachine, Pelotas, Poweron, Tawkerbot2, George100, Purplesword, Pgr94, Bumbulski, Grein, Simeon, Gpel461, VladB, Cydebot, Carl Turner, ST47, Tawkerbot4, Roberta F., Scarpy, Omicronpersei8, Klausikm, Lawrenceb, Techna1, Bockbockchicken, Pruetboonma, Massimo Macconi, Tapan bagchi, KrakatoaKatie, AntiVandalBot, Baguio, TimVickers, Temporary-login, Kdakin, JANdbot, MER-C, CosineKitty, Raduberinde, SDas, Xn4, PeterStJohn, Eleschinski2000, Destynova, SSZ, David Eppstein, User A1, Jetxee, Diroth, Francob, A. S. Aulakh, Projectstann, Sm8900, Glrx, Tulkolahten, J.delanoy, Stochastics, TempestCA, Silas S. Brown, K.menin, Edrucker, Plasticup, Feyeligh, DavidCBryant, Riccardopoli, Ratfox, Useight, CardinalDan, Marksale, VolkovBot, JohnBlackburne, AlnoktaBOT, Kyle the bot, TXiKiBot, Esotericengineer, Tameeria, Rei-bot, TyrantX, Kjells, Keburjor, Jasper53, Broadbot, Andy Dingley, Kindyroot, AlterMind, Cnilep, Thric3, SieBot, Bjtaylor01, Zwgeem, ToePeu.bot, Kaell, Yuanwang200409, Antzervos, Raulcleary, Algorithms, CharlesGillingham, Anchor Link Bot, DixonD, Novablogger, Guang2500, CShistory, AussieScribe, Cngoulimis, Djhache, Euhapt1, SlackerMom, ClueBot, Toshke, Justin W Smith, Ahyeek, Swarmcode, Adrianwn, Razimanty, Mild Bill Hiccup, Bradka, Dylan620, Otolemur crassicaudatus, Jkolom, DragonBot, SteelSoul, Calimo, Leonard^Bloom, Chaosdruid, Unixcrab, Frongle, Josilber, LieAfterLie, Johnuniq, DumZiBoT, XLinkBot, Avoided, Addbot, DOI bot, Download, ChenzwBot, SamatBot, Tide rolls, Chipchap, Angrysockhop, Legobot, Luckas-bot, Yobot, Twexcom, Armchair info guy, AnomieBOT, Paskornc, Jim1138, No1sundevil, Citation bot, ArthurBot, Breeder8128, Xqbot, TheAMmollusc, Jeffrey Mall, DSisyphBot, GrouchoBot, Tarantulae, BulldogBeing, FrescoBot, Mark Renier, Mohdavary, Citation bot 1, Kelaodisi, Biker Biker, RedBot, Kon michael, Orenburg1, TobeBot, Catator, Jonkerz, Rdelcueto, YouAndMeBabyAintNothingButCamels, RjwilmsiBot, Justinaction, VernoWhitney, BertSeghers, Lbartnik, Kencflew, Dzkd, Sigurdurbj, Yuejiao Gong, Sunandwind, Gretchen Hea, Wikipelli, Daryakov, Optimering, Nentrex, H3llBot, SporkBot, AManWithNoPlan, Kcwong5, RockMagnetist, Tezzet, Sednodna, ClueBot NG, Mathstat, Mctechucztecatl, Helpful Pixie Bot, Jr271, J.Dong820, Luge74, BG19bot, Padapim.G, Metricopolus, Anubhab91, Compfreak7, Darkeffy, BattyBot, Ferrarisailor, ChrisGualtieri, Mediran, AusCanBri, Oritnk, Mogism, Kub1x, Mrzazz001, Thomas Jeal, Mark viking, Selvi muthukumar, Epicgenius, Bhrnjica, Jramsden271, Brzydalski, David.conradie, Kks11deq, Monkbot, Btomoiaiga, Sofia Koutsouveli, Moshaydi, DAColey, Olosko, Boky90, SharifCS, Muafa, Cruella wiki and Anonymous: 459
- **Genetic algorithms in economics** *Source:* <http://en.wikipedia.org/wiki/Genetic%20algorithms%20in%20economics?oldid=545879797> *Contributors:* Michael Hardy, Greenrd, Neoncow, John Quiggin, Ylem, SmackBot, Parent5446, CRGreathouse, Cydebot, Tsabbadini, Destynova, TempestCA, Ric168, DavidCBryant, CharlesGillingham, Melcombe, Frongle, Jarble, AnomieBOT, Visoot, YouAndMeBabyAintNothingButCamels, RockMagnetist and Anonymous: 2
- **Glowworm swarm optimization** *Source:* <http://en.wikipedia.org/wiki/Glowworm%20swarm%20optimization?oldid=549559877> *Contributors:* Michael Hardy, Andreas Kaufmann, Diego Moya, Ruud Koot, Rjwilmsi, Epipelagic, SMasters, Cydebot, Utopiah, EagleFan, David Eppstein, K.menin, Thisisborin9, Biscuittin, Addbot, Materialscientist, Jonkerz, Josephgec, Pepanek Nezdara, DoctorKubla, Wverv, Kaipakrishna and Anonymous: 4
- **Gradient method** *Source:* <http://en.wikipedia.org/wiki/Gradient%20method?oldid=545095574> *Contributors:* Altenmann, Cydebot, Ad-dbot, CES1596 and Jonkerz
- **Graduated optimization** *Source:* <http://en.wikipedia.org/wiki/Graduated%20optimization?oldid=630208838> *Contributors:* Michael Hardy, Chris Capoccia, Cydebot, Headlessplatter, Yobot, AnomieBOT, Citation bot, Erik9bot, Laughsinthestocks, BG19bot, Mark viking and Anonymous: 4
- **Great Deluge algorithm** *Source:* <http://en.wikipedia.org/wiki/Great%20Deluge%20algorithm?oldid=544561284> *Contributors:* Xiaodai, Pavel Vozenilek, Oleg Alexandrov, Cydebot, KrakatoaKatie, K.menin, PixelBot, Addbot, FrescoBot and Anonymous: 3
- **Guided Local Search** *Source:* <http://en.wikipedia.org/wiki/Guided%20Local%20Search?oldid=618012686> *Contributors:* Edward, Michael Hardy, Jitse Niesen, Altenmann, Rich Farmbrough, Elkman, Jasonb05, Cydebot, Robina Fox, K.menin, Epktsang, Monobi, Is-maelLuceno, MrOllie, AnomieBOT, Socialopt and Anonymous: 7
- **Harmony search** *Source:* <http://en.wikipedia.org/wiki/Harmony%20search?oldid=626046167> *Contributors:* Twanyl, Diego Moya, Oleg Alexandrov, Ruud Koot, Rjwilmsi, SmackBot, Jasonb05, Ehheh, Cydebot, Alaiobot, Kroese, Magioladitis, K.menin, KylieTastic, VolkovBot, UNDGRND, Fesanghary, Trumpetpunk42, Zwgeem, KathrynLybarger, Celique, ClueBot, Sun Creator, SchreiberBike, Ghutemann, Mexy ok, Addbot, MrOllie, AnomieBOT, FrescoBot, Jacofourie, Aacoo, JosephCatrambone, Evonash, Daryakov, Aaissa70, Jr271, BG19bot, Pacopocopico, Yunoromero, Henry Weyland, Ganjkola and Anonymous: 36

- **Imperialist competitive algorithm** *Source:* <http://en.wikipedia.org/wiki/Imperialist%20competitive%20algorithm?oldid=626319456> *Contributors:* Rjwilmsi, Cydebot, Headbomb, Magioladitis, Moonriddengirl, AnomieBOT, FrescoBot, Duoduoduo, RjwilmsiBot, EmausBot, Mmdba, Lateg, Icasite, Vahidkay, Jalalmousavirad and Anonymous: 4
- **Intelligent Water Drops algorithm** *Source:* <http://en.wikipedia.org/wiki/Intelligent%20Water%20Drops%20algorithm?oldid=611047390> *Contributors:* Michael Hardy, Rjwilmsi, Yobot, Xqbot, FrescoBot, EmausBot, ClueBot NG, Hoodaan, Gronk Oz and Anonymous: 3
- **Interior point method** *Source:* <http://en.wikipedia.org/wiki/Interior%20point%20method?oldid=649818817> *Contributors:* Mark Foskey, Charles Matthews, Jitse Niesen, Alotau, TedPavlic, Oleg Alexandrov, Marcol, YurikBot, Whaa?, SmackBot, Serg3d2, Tgdwyer, Rijkbenik, Lavaka, Cydebot, Headbomb, Avaya1, Johnbibby, K.menin, Llorenzi, VolkovBot, Trevorgoodchild, Anna Lincoln, Reinderien, Atdotde, Addbot, AnomieBOT, Citation bot, CXCV, Alexeicolin, Kiefer.Wolfowitz, Duoduoduo, RjwilmsiBot, Bonnans, AManWithNoPlan, Bugmenot10, Helpful Pixie Bot, Lolamind, Hannes sommer, Monkbot, Fergus the widget, Uriah Huang and Anonymous: 27
- **Interval contractor** *Source:* <http://en.wikipedia.org/wiki/Interval%20contractor?oldid=555046427> *Contributors:* Michael Hardy, Bearcat, Luc Jaulin and Anonymous: 1
- **IOSO** *Source:* <http://en.wikipedia.org/wiki/IOSO?oldid=627206556> *Contributors:* Michael Hardy, SmackBot, Cydebot, EagleFan, Vanished User 8902317830, DanielPharos, Xevilgeniusx, Klochkov.ivan, John of Reading, MathMaven, Snotbot, BG19bot and Anonymous: 4
- **IPOPT** *Source:* <http://en.wikipedia.org/wiki/IPOPT?oldid=629068106> *Contributors:* The Anome, Idpipe, Ajgorhoe, Alison9, Oleg Alexandrov, Bobstay, DevastatorIIC, RussBot, EncMstr, Colonies Chris, JonHarder, Claird, Cydebot, Fbahr, A quant, Gwern, Akasha27, Ankital, Dmitrey, Apmonitor, R.lopez.negrete, MrOllie, Delaszk, Tassedethe, Ettrig, Luckas-bot, FrescoBot, Kiefer.Wolfowitz, EmausBot, Cramilo, Tyrneaepith and Anonymous: 21
- **Iterated local search** *Source:* <http://en.wikipedia.org/wiki/Iterated%20local%20search?oldid=635237827> *Contributors:* Edward, Rjwilmsi, Cydebot, Malcolmxl5, Drpicke, Yobot, FrescoBot, Robiminer, Matthieu Vergne, Monkbot and Anonymous: 3
- **Job shop scheduling** *Source:* <http://en.wikipedia.org/wiki/Job%20shop%20scheduling?oldid=643261773> *Contributors:* Michael Hardy, Dcoetze, Greenrd, Phil Boswell, Sunray, Shahab, Diego Moya, Qwertus, Rjwilmsi, Matt Deres, Kernanb, RussBot, Sasuke Sarutobi, Bovineone, Epipelagic, AlexTG, Nelson50, SmackBot, Gilliam, Franc, CRGreathouse, Cydebot, Headbomb, Dougher, David Eppstein, Calltech, MichaelClair, Crouz, Sallicio, DOI bot, Yobot, Kilom691, AnomieBOT, Citation bot, Jordiferrer, Thore Husfeldt, Omnipaedista, Mcmlxxxii, Citation bot 1, Trappist the monk, YouAndMeBabyAintNothingButCamels, RjwilmsiBot, Hba2, Alpha Quadrant (alt), AvicAWB, ClueBot NG, BG19bot, Qx2020, Monkbot, U2fanboi, Moorshed k, Piwaldo and Anonymous: 26
- **Kantorovich theorem** *Source:* <http://en.wikipedia.org/wiki/Kantorovich%20theorem?oldid=615306815> *Contributors:* Michael Hardy, Charles Matthews, Markhurd, Tobias Bergemann, Giftlite, Bender235, LutzL, RDBury, Cronholm144, Cydebot, R'n'B, Phantom xxiii, Addbot, Yobot, Kiefer.Wolfowitz, DARTH SIDIOUS 2, Perpeduumimmobile and Anonymous: 3
- **Killer heuristic** *Source:* <http://en.wikipedia.org/wiki/Killer%20heuristic?oldid=620005096> *Contributors:* Shimmin, Giftlite, Wmahan, Tabletop, GregorB, Ground Zero, SColombo, SmackBot, DCDuring, Loabok, IanOsgood, Bouke, JaysonSunshine and Anonymous: 10
- **LINCOA** *Source:* <http://en.wikipedia.org/wiki/LINCOA?oldid=655208857> *Contributors:* Qwertus, Chilep, Yobot, BG19bot, Zhangzk and Anders9ustafsson
- **Local convergence** *Source:* <http://en.wikipedia.org/wiki/Local%20convergence?oldid=526081343> *Contributors:* CBM, Cydebot, Jmath666, Yobot, Kiefer.Wolfowitz and Anonymous: 1
- **Luus–Jaakola** *Source:* <http://en.wikipedia.org/wiki/Luus%E2%80%93Jaakola?oldid=573267340> *Contributors:* Michael Hardy, Cydebot, Headbomb, MrOllie, Yobot, Kiefer.Wolfowitz, Optimering and SporkBot
- **Matrix chain multiplication** *Source:* <http://en.wikipedia.org/wiki/Matrix%20chain%20multiplication?oldid=654175224> *Contributors:* Michael Hardy, Dominus, Charles Matthews, Dcoetze, Jogloran, Giftlite, Real NC, Aranel, Oleg Alexandrov, Qwertus, BenPowerski, Rjwilmsi, Cedar101, SmackBot, Chris the speller, Kostmo, MichaelBillington, Chrylis, Vina-iwbot, Bleargh, Screw3d, CmdrObot, Cydebot, Ike-bana, Asmeurer, Email4mobile, David Eppstein, Gwern, M simin, CrossWinds, Ged.R, Nicgarner, Caltas, Sioffe, Jonlandrum, ClueBot, Cp111, Max613, Kletos, Addbot, Yoenit, Doniago, مارک دوک, Dodek, Yobot, Gms, Citation bot, PabloCastellano, MastiBot, R.J.C. van Haaften, Sihag.deepak, Polariseke, RjwilmsiBot, StitchProgramming, LokeshRavindranathan, Hiteshgupta88, Monkbot, Leegrc, R.J.C.vanHaaften and Anonymous: 50
- **Maximum subarray problem** *Source:* <http://en.wikipedia.org/wiki/Maximum%20subarray%20problem?oldid=645109582> *Contributors:* Andreas Kaufmann, Ascetic, Qwertus, Darguz Parsilvan, Wavelength, Ilmari Karonen, Heavyrain2408, SmackBot, Derek farn, Scientizle, Tobe2199, Markandey, Cydebot, Leolaursen, Acertain, Fabrictramp, A3nm, David Eppstein, Pilotbob, Davekaminski, Guray9000, Baiyubin, Somasundarambk, Addbot, Amirobot, AnomieBOT, Materialscientist, Xqbot, Rohit Coolkarni, Dsulli99, Arademaker, Jothikumar Rathinamoorthy, Tellarunbose, Elromulous, Nischayn22, ChrisGualtieri, افریزه رکرسو, Rorkeso, Luthefirst, DrMorcos, Akira Li 12345, Amitpurwar, Mjnovice, Eennaa, Powder 77, Anthonychhan, Alex Muscar, Softweave, Rohithkanna1987 and Anonymous: 51
- **MCS algorithm** *Source:* <http://en.wikipedia.org/wiki/MCS%20algorithm?oldid=614151185> *Contributors:* Andrewman327, Oleg Alexandrov, Finbarr Saunders, X1cygnus, SmackBot, Cydebot, KrakatoaKatie, Meredyth, Gilderien, Qetuth and Anonymous: 4
- **Mehrotra predictor–corrector method** *Source:* <http://en.wikipedia.org/wiki/Mehrotra%20predictor%20%20corrector%20method?oldid=616809178> *Contributors:* Charles Matthews, Jitse Niesen, Oleg Alexandrov, Marcol, Mathbot, Tony1, Tribaal, Lunch, Hua001, Cydebot, Magioladitis, DOI bot, Yobot, Citation bot 1, Monkbot and Anonymous: 2
- **Meta-optimization** *Source:* <http://en.wikipedia.org/wiki/Meta-optimization?oldid=653877386> *Contributors:* Michael Hardy, Ruud Koot, Rjwilmsi, Petkr, Cydebot, Christian75, This, that and the other, Addbot, MrOllie, Yobot, Citation bot, Citation bot 1, Kiefer.Wolfowitz, Optimering, Will Beback Auto, BG19bot, José Javier Señaris, Monkbot and Anonymous: 3
- **Minimax** *Source:* <http://en.wikipedia.org/wiki/Minimax?oldid=653532085> *Contributors:* Tobias Hoevekamp, Zundark, The Anome, Andre Engels, Arvindn, ChangChienFu, Imran, Robert Dober, Michael Hardy, Kku, MatrixFrog, Dcoetze, Jitse Niesen, Furrykef, MH, R3m0t, Henrygb, Wereon, Robinh, Giftlite, DavidCary, Mat-C, Sampo, Remy B, Foobar, Karl-Henner, Sam Hocevar, Rich Farmbrough, ZeroOne, El C, Grick, Triepg, Cretog8, Touriste, Scott sauyet, Sean Kelly, PsiXi, Cburnett, LOL, David Haslam, Moneky, MattGiua, Smmurphy, RalfKoch, CharlesC, Qwertus, OliAtlasson, NeonMerlin, Pete.Hurd, WriterHound, UkPaolo, YurikBot, Borgx, Trovatore, Dlugosz, Zvika, SmackBot, Honza Záruba, Eskimbot, Syr0, Tghe-retford, Nbarth, DHN-bot, Pegua, Glengordon01, Ohconfucius, Will

Beback, Nmnogueira, Dante Shamest, Brainix, A. Pichler, Karenjc, Cydebot, Erasmussen, Edupedro, AllUltima, Escarbot, Beta16, TuvicBot, BKfi, OckRaz, PhilKnight, Garygagliardi, David Eppstein, SlamDiego, Icaoberg, Policron, Kriskra, Philip Trueman, TXiKiBoT, Rei-bot, Telespiza, Geometry guy, Wikiisawesome, Monty845, Ctxppc, CharlesGillingham, Anchor Link Bot, Bpeps, ClueBot, Artichoker, Vacio, Emmanuel5h, No such user, Alexbot, Xijiah, Brews ohare, SchreiberBike, Qwfp, Sniedo, XLinkBot, Spitfire, Thinboy00P, Addbot, נטנער, Maschelos, Gametheorist77, Tassedethe, Hunyadym, RobertHannah89, Luckas-bot, Yobot, Ptbotgourou, AnomieBOT, Citation bot, ArthurBot, Xqbot, Anne Bauval, Resident Mario, RibotBOT, Citation bot 1, Shuroo, Kiefer.Wolfowitz, Riitoken, Maximin, MastiBot, Surement, Lotje, EmausBot, RenamedUser01302013, Terry0201, Andresambrois, Tijfo098, Nesasio, ClueBot NG, Dr. Persi, Frijtjes, Hahahafr, JanSegre, Hoyda1, Sialtschuler, Mogism, Abderrahman AIT ALI, GamePlayerAI, Tim36272, Zdravozdravo6 and Anonymous: 155

- **MM algorithm** *Source:* <http://en.wikipedia.org/wiki/MM%20algorithm?oldid=635322522> *Contributors:* Billlion, Rjwilmsi, JHunterJ, Yobot, AnomieBOT, Zixu1986, Chris857, Fzwaest, CarrieVS, Auwzb, Cerisara, DilumA and Anonymous: 8
- **Multi-swarm optimization** *Source:* <http://en.wikipedia.org/wiki/Multi-swarm%20optimization?oldid=637238124> *Contributors:* Michael Hardy, Nbarth, Cydebot, Sychen, Sun Creator, Addbot, Yobot, Bolufe, ZéroBot, TheHappiestCritic, Filing Flunk and Anonymous: 4
- **Natural evolution strategy** *Source:* <http://en.wikipedia.org/wiki/Natural%20evolution%20strategy?oldid=650882817> *Contributors:* Michael Hardy, Cydebot, CorenSearchBot, Adrignola, Tomschaul, TheProfessor, The Average Wikipedian and Anonymous: 3
- **Negamax** *Source:* <http://en.wikipedia.org/wiki/Negamax?oldid=650166677> *Contributors:* Edward, Bart Massey, Rl, Furrykef, RichiH, Nomeata, Utcursch, Kogorman, Cretog8, GregorB, Qwertys, Krapnik, Aasmith, SmackBot, Cydebot, Thij's!bot, Jdm64, Heineman, Inhumandecency, JSB73, Addbot, Maschelos, Mik01aj, AnomieBOT, SassoBot, Erik9bot, Wyverald, Devper94, ZéroBot, Tecknoize, ChuispastonBot, Hahahafr, BG19bot, SmagR59R60R61, GamePlayerAI and Anonymous: 20
- **Newton's method** *Source:* <http://en.wikipedia.org/wiki/Newton%20method?oldid=651158768> *Contributors:* AxelBoldt, Lee Daniel Crocker, Zundark, Miguel, Roadrunner, Formulax, Hirzel, Pichai Asokan, Patrick, JohnOwens, Michael Hardy, Pit, Dominus, Deljr, Loisel, Minesweeper, Ejrh, Looxix, Cyp, Poor Yorick, Pizza Puzzle, Hike395, Dcoetze, Jitse Niesen, Kb, Saltine, AaronSw, Robbot, Jaredwf, Fredrik, Wikibot, Giftlite, Rs2, BenFrantzDale, Neile, MarkSweep, PDH, Torokun, Sam Hocevar, Kutulu, Fintor, Frau Holle, TheObtuseAngleOfDoom, Paul August, Pt, Aude, Iamunknow, Blotwell, Nk, Naham hanuka, LutzL, Borisblue, Jeltz, Laug, Olegalexandrov, Oleg Alexandrov, Tbsmith, Joriki, Shreevatsa, LOL, Decrease789, Jimbryho, Robert K S, Birge, GregorB, Casey Abell, Eyu100, Mathbot, Shultz, Kri, Glenn L, Wikipedia is Nazism, Chobot, YurikBot, Wavelength, Laurentius, Swerty, JabberWok, KSmrq, Exir Kamalabadi, Tomisti, Alias Flood, Marquez, SmackBot, RDBury, Selfworm, Adam majewski, Saravask, Tom Lougheed, InverseHypercube, Jagged 85, Dulcamara, Commander Keane bot, Slaniel, Skizzik, Chris the speller, Berland, Rrburke, Earlh, ConMan, Jon Awbrey, Henning Makhholm, Bdiscoe, Wvbaily, Coredesat, Jim.belk, Magnait, Gco, JRSpriggs, CRGreathouse, Jackzhp, David Cooke, Holycow958, Eric Le Bigot, Cydebot, Quibik, Christian75, Billtubbs, Talgalili, Thij's!bot, Epbr123, Nonagonal Spider, Headbomb, Martin Hedegaard, BigJohnHenry, Ben pcc, Seaphoto, CPMartin, JAnDbot, Coffee2theorems, VoABot II, JamesBWatson, Baccyak4H, Avicennasis, David Eppstein, User A1, GuidoGer, Arithmonic, Glrx, Pbroks13, Kawautar, Rankarana, Nedunuri, K.menin, Gombang, Chiswick Chap, Goingstuckey, Policron, Juliancolton, Homo logos, JohnBlackburne, Philip Trueman, TXiKiBoT, Anonymous Dissident, Broadbot, Aaron Rotenberg, Draconx, Pitel, Katzmik, Psymun747, SieBot, Gex999, Dawn Bard, Bentogoa, Flyer22, MinorContributor, Jasondet, Smarchesini, Redmarkviolinist, Dreamofthedolphin, Cyfal, PlantTrees, ClueBot, Metaprimer, Wysprgr2005, JP.Martin-Flatin, Mild Bill Hiccup, CounterVandalismBot, Tesspub, Chrisgolden, Anne furanku, Dekisugi, Xooll, Muro Bot, Jpginn, RMFan1, Galoisgroupie, Addbot, Some jerk on the Internet, Eweinber, Fluffernutter, Ckamas, Protonk, EconoPhysicist, LinkFA-Bot, Uscitizenjason, AgadaUrbanit, Numbo3-bot, Tide rolls, CountryBot, Xieyhui, Luckas-bot, Yobot, Estudiarne, AnomieBOT, 1exec1, Illegal604, Apau98, Пика Пика, Materialscientist, Zhurov, ArthurBot, PavelSolin, Capricorn42, Titolatif, CBoeckle, Nyirenda, Dlacesz, Point-set topologist, CnkALTDS, Shadowjams, FrescoBot, Pepper, DNA Games, Citation bot 1, Tkuvho, Gaba p, Pinethicket, Eyrrys, Kiefer.Wolfowitz, White Shadows, Vrenator, Clark-Sims, Duoduo duo, KMic, Suffusion of Yellow, H.ehsaan, Jfmantis, Hyarmendacil, 123Mike456Winston789, EmausBot, TheJesterLaugh, KHamsun, Mmeijeri, Shuipzv3, D.Lazard, Eniagram, U+003F, Bomazi, Chris857, Howard nyc, Kyle.drerup, Elvek, Mikhail Ryazanov, ClueBot NG, KlappCK, Aero-Plex, Chogg, Helpful Pixie Bot, EmadIV, Drift chambers, Toshiki, Scuchina, AWTom, RiabzevMichael, Electricmuffin11, Khazar2, Qxukhgiels, Dexbot, M.shahriarinia, I am One of Many, Lakshmi7977, Manoelramon, Ginsuloft, Mohit.del94, Blitzall, Rob Haelterman, Loraof and Anonymous: 275
- **NEWUOA** *Source:* <http://en.wikipedia.org/wiki/NEWUOA?oldid=655208550> *Contributors:* Qwertys, Rjwilmsi, BiH, Hebrides, Magioladitis, Cnilep and Zhangzk
- **Nonlinear programming** *Source:* <http://en.wikipedia.org/wiki/Nonlinear%20programming?oldid=655173647> *Contributors:* Michael Hardy, Stevenj, Hike395, Charles Matthews, Jitse Niesen, Jaredwf, Giftlite, Bfinn, Leonard G., Ajgorhoe, Mike40033, Frau Holle, Monkeyman, Mdd, Olegalexandrov, Oleg Alexandrov, Miaow Miaow, Myleslong, Krishnavedala, YurikBot, FrenchIsAwesome, David R. Ingham, Bota47, SmackBot, Brunner7, Tgdwyer, Bluebot, MartinPoulter, EncMstr, Broom eater, G.de.Lange, Hu12, RekishiEJ, Dto, Cydebot, DumbBOT, Headbomb, EdJohnston, KrakatoaKatie, AntiVandalBot, .anacondabot, Sabamo, User A1, Sbvb, VolkovBot, Xeltifon, BarryList, Psvarbanov, Jamelan, Garde, Metiscus, PimBeers, Niceguyedc, Dmitrey, Addbot, MrOllie, LaaknorBot, EconoPhysicist, Alexander.mitsos, Snaily, Luckas-bot, Yobot, McSush, LilHelpa, Isheden, Mcmlxxxi, FrescoBot, Kiefer.Wolfowitz, Callanecc, EmausBot, WikitanvirBot, Nacopt, ZéroBot, Vgmdg, Makecat, Jean-Charles.Gilbert, Rezabot, PiotrGregorczyk, Mehr86 and Anonymous: 52
- **Ordered subset expectation maximization** *Source:* <http://en.wikipedia.org/wiki/Ordered%20subset%20expectation%20maximization?oldid=494678207> *Contributors:* The Anome, SimonP, Michael Hardy, Karada, Charles Matthews, Eubot, SmackBot, Bluebot, G716, Dinamisbo, Bwpach, Mafiq, Cydebot, Melcombe, Pratx, Kiefer.Wolfowitz and Anonymous: 8
- **Parallel metaheuristic** *Source:* <http://en.wikipedia.org/wiki/Parallel%20metaheuristic?oldid=595285742> *Contributors:* Michael Hardy, EncMstr, Gregbard, Cydebot, Magioladitis, Falcon8765, Mild Bill Hiccup, Sun Creator, Paradiseo, AnomieBOT, John of Reading, Enrique.alba1, Almeria.raul, Jptustin and Anonymous: 1
- **Particle swarm optimization** *Source:* <http://en.wikipedia.org/wiki/Particle%20swarm%20optimization?oldid=653877383> *Contributors:* Michael Hardy, Lexor, Ronz, Mxn, Hike395, Jitse Niesen, Unknown, Robbot, Amgine, Lysy, Giftlite, Sepreece, BenFrantzDale, Dratman, Tagishsimon, Horndude77, Ronaldo, Rich Farmbrough, Swiftly, Photonique, Diego Moya, Oleg Alexandrov, Ruud Koot, Waldir, CoderGnome, Rjwilmsi, Datakid, Hgkamath, Wavelength, NawlinWiki, Epipelagic, Slicing, Dbratton, Neomagus00, Gwe0351, SmackBot, Ma8thew, Mcld, Oli Filth, DHN-bot, Prometheus, Cybercobra, Blake-, Ehheh, Dicklyon, Mishrasknehu, Cydebot, DustinFreeman, BetacommandBot, KrakatoaKatie, Whenning, Storkk, Armehrabian, Mange01, Jiuguang Wang, NerdyNSK, K.menin, My wing hk, Seb az86556, Tjh22, Lourakis, Seamustara, CharlesGillingham, Denisarona, Mild Bill Hiccup, Blanchardb, Betamoo, Saeed.Veradi, Sliders06, Mexy ok, Addbot, Foma84, MrOllie, Chipchap, Ender.ozcan, Luckas-bot, Yobot, Sriramvijay124, Kingpin13, Citation bot, Wgao03,

YakbutterT, McoupaL, SassoBot, MuffledThud, Joquin008, Sanremofilo, Jder, FrescoBot, Khafanus, Sharkyangliu, AdrianoCunha, Zhanapollo, Saveur, Bolufe, Bshahul44, Gfoidl, RjwilmsiBot, Ripchip Bot, Becritical, EmausBot, Dzkd, Murilo.pontes, Yuejiao Gong, Huabdo, Daryakov, Optimering, ZéroBot, PS., George I. Evers, SporkBot, MCLerc, Wingman417, EnzzeF, Bdonckel, ChuispastonBot, Jalsck, ClueBot NG, Swarming, Anne Koziolek, Helpful Pixie Bot, Younessabdussalam, BG19bot, Rijinatwki, ChrisGaultieri, Σμήνος, Garytvocal, Mark viking, Sharkyangliu916, Rerumpf, Monkbot, Rezabny and Anonymous: 150 سرچ

- **Pattern search (optimization)** *Source:* [http://en.wikipedia.org/wiki/Pattern%20search%20\(optimization\)?oldid=634028792](http://en.wikipedia.org/wiki/Pattern%20search%20(optimization)?oldid=634028792) *Contributors:* Michael Hardy, Jonsafari, Solidpeg, RussBot, KerryVeenstra, Cydebot, Gjacquenot, Addbot, MrOllie, Luckas-bot, Isheden, Kiefer.Wolfowitz, RjwilmsiBot, John of Reading, Optimering, SporkBot and Anonymous: 3
- **PSeven** *Source:* <http://en.wikipedia.org/wiki/PSeven?oldid=654717459> *Contributors:* Ukekpat, X201, ImageRemovalBot, YSSYguy, Yobot and Nata Kozlova
- **Quantum annealing** *Source:* <http://en.wikipedia.org/wiki/Quantum%20annealing?oldid=652783784> *Contributors:* Dcoetzee, Jorge Stolfi, Aibyou chan, Ben Standeven, Mpeg4codec, Jérôme, Ringbang, Oleg Alexandrov, Adking80, Bgwhite, RussBot, Gaius Cornelius, Petri Krohn, Fuhghettaboutit, ChrisCork, CRGreathouse, Cydebot, Arnab das, Erik53081, KrakatoaKatie, Widefox, Aswarp, K.menin, Sun Creator, Addbot, Bruno.apolloni, Prim Ethics, Yobot, 4th-otaku, JKoff, MsPorterAtFHS, Wikielwikingo, EmausBot, P. Dickins, Chris857, BG19bot, QuantumPhysicsPhD, Weiberg1, JudieT, Anrusnusna, Rwarren123 and Anonymous: 36
- **Quasi-Newton method** *Source:* <http://en.wikipedia.org/wiki/Quasi-Newton%20method?oldid=642945736> *Contributors:* Michael Hardy, Jitse Niesen, Benwing, DonvinzK, BenFrantzDale, MisterSheik, O18, Oleg Alexandrov, Male1979, Qwertys, Marcol, Grafen, Zwobot, Bilgrau, Lavaka, Cydebot, Alaibot, AbnerCYH, JPRBW, Abnercyh, R'n'B, Mathemaduenn, Thomas.kn, Smarchesini, Jdaloner, Shai mach, Rubybrian, Addbot, LaaknorBot, Yobot, Zakke, Happyrabbit, Isheden, ChristianGruen, Drybid, Kiefer.Wolfowitz, ClueBot NG, Starshipenterprise, Dayson39, Jimw338, ChrisGaultieri, Aschmied, Rob Haelterman, Velvel2 and Anonymous: 32
- **Random optimization** *Source:* <http://en.wikipedia.org/wiki/Random%20optimization?oldid=605932982> *Contributors:* Zundark, Michael Hardy, David Martland, Kku, Breakpoint, Diberri, Beland, Rjwilmsi, Tribaal, DonkeyKong64, CRGreathouse, Cydebot, KrakatoaKatie, MystBot, Legobot, Yobot, J04n, FrescoBot, RobinK, RjwilmsiBot, Optimering, SporkBot, ChrisGaultieri and Anonymous: 4
- **Random search** *Source:* <http://en.wikipedia.org/wiki/Random%20search?oldid=628847771> *Contributors:* Michael Hardy, Rjwilmsi, Octahedron80, Cydebot, A3nm, Addbot, 4th-otaku, RjwilmsiBot, Optimering, SporkBot, Sphereon, Shaggypete19, VeryCrocker and Anonymous: 5
- **Rosenbrock methods** *Source:* <http://en.wikipedia.org/wiki/Rosenbrock%20methods?oldid=578754403> *Contributors:* Michael Hardy, BenFrantzDale, SmackBot, Karthik.raman, Ser Amantio di Nicolao, Cydebot, K.menin, MatthewVanitas, MKzim, SciCompTeacher, ChrisGaultieri, LaguerreLegendre and Anonymous: 1
- **Search-based software engineering** *Source:* <http://en.wikipedia.org/wiki/Search-based%20software%20engineering?oldid=649428575> *Contributors:* Edward, Hairy Dude, Jpbown, Prodego, Hongooi, Ohconfucius, Cydebot, Alaibot, Lfstevens, Elinruby, GoheX, Charleca, Samansouri, NVar, Niceguyedc, Arjayay, Dthomsen8, Addbot, Yobot, AnomieBOT, FrescoBot, Marw08, Mo aimn, Chire, Philmcminn, ClueBot NG, Primergrey, Enrique.alba1, BattyBot, ChrisGaultieri, Mogism, John.robert.woodward, Robert4565, ShaukatAli1982, Npcomp, Monkbot, Rjkel, Pbg6yh9s, Anothernoggintheng, Emil.rubinic and Anonymous: 18
- **Sequence-dependent setup** *Source:* <http://en.wikipedia.org/wiki/Sequence-dependent%20setup?oldid=465609718> *Contributors:* Michael Hardy, Kernanb, MalcolmA, Cydebot, Erik9bot, YouAndMeBabyAintNothingButCamels and Anonymous: 4
- **Sequential minimal optimization** *Source:* <http://en.wikipedia.org/wiki/Sequential%20minimal%20optimization?oldid=621332973> *Contributors:* Michael Hardy, Phil Boswell, Qwertys, David Pal, SmackBot, Cydebot, Marcuscalabresus, Jiuguang Wang, K.menin, Semifinalist, Casablanca2000in, Addbot, X7q, Ch3m, Siyuwj, Koertefa and Anonymous: 5
- **Shuffled frog leaping algorithm** *Source:* <http://en.wikipedia.org/wiki/Shuffled%20frog%20leaping%20algorithm?oldid=645880246> *Contributors:* Michael Hardy, Mohsinbokhari 2, BG19bot and User-000
- **Simultaneous perturbation stochastic approximation** *Source:* <http://en.wikipedia.org/wiki/Simultaneous%20perturbation%20stochastic%20approximation?oldid=613436675> *Contributors:* Michael Hardy, GregorB, Rjwilmsi, SmackBot, Cydebot, Stochastics, Hellno2, Phil Bridger, Melcombe, Eeekster, LilHelpa, The Banner, Alvin Seville, FrescoBot, Kiefer.Wolfowitz, M for Molecule, DexDor, Zfeinst, Salman3037, BG19bot, DoctorKubla, Matrig, Noyal Sharook and Anonymous: 9
- **Social cognitive optimization** *Source:* <http://en.wikipedia.org/wiki/Social%20cognitive%20optimization?oldid=645042926> *Contributors:* Mild Bill Hiccup, Yobot, Xfxie and Anonymous: 2
- **Space allocation problem** *Source:* <http://en.wikipedia.org/wiki/Space%20allocation%20problem?oldid=597098809> *Contributors:* Wavelength, Joao.pimentel.ferreira, Makecat, Batard0 and BattyBot
- **Space mapping** *Source:* <http://en.wikipedia.org/wiki/Space%20mapping?oldid=622064677> *Contributors:* MadmanBot, Neøn, Mr-NiceGuy1113, Fox2k11, Mark viking, Johnbandler, Radh 60, Shasha.cheng and Anonymous: 2
- **Special ordered set** *Source:* <http://en.wikipedia.org/wiki/Special%20ordered%20set?oldid=647068427> *Contributors:* Oleg Alexandrov, Salix alba, Cydebot, PER9000, Edadk, Mild Bill Hiccup, Yobot, Eugene-elgato, FrescoBot, Asommerh, John of Reading, Bugmenot10, Ukjt, ChrisGaultieri, Arcandam and Anonymous: 5
- **Stochastic hill climbing** *Source:* <http://en.wikipedia.org/wiki/Stochastic%20hill%20climbing?oldid=544133159> *Contributors:* Oleg Alexandrov, SmackBot, Took, Cydebot, Addbot, Luckas-bot, Erik9bot, EmausBot and Anonymous: 2
- **Swarm intelligence** *Source:* <http://en.wikipedia.org/wiki/Swarm%20intelligence?oldid=655336830> *Contributors:* Michael Hardy, Ixfd64, Ronz, Andrevan, Hydnio, Warofdreams, Phil Boswell, Robot, Altenmann, Giftlite, Gtrmp, Jyril, Everyking, Khalid hassani, Simoneau, Urhixidur, Ronaldo, Discospinster, Bender235, Photonique, Ire and curses, Mdd, K3rb, Oleg Alexandrov, Myleslong, Ruud Koot, Waldir, Stefanomione, Ashmoo, Rjwilmsi, Quiddity, Rewinn, Zeeshan.Usmani, Mathbot, TheMidnighters, Sheltim, Chobot, Bgwhite, Shervinashar, Wavelength, Jacen137, GeeJo, M0nstr42, Kjl, Deodar, Rjlabs, Epipelagic, Diavosh, Rwalker, Black Falcon, Pawylee, Knowlengr, Nikkimaria, Arthur Rubin, Warreed, SmackBot, Moxon, Mneser, Chris the speller, Apankrat, Xino2005, Nbarth, Joerite, Rene Mas, Dreadstar, Yserbius, Physis, Werdan7, Roregan, Beefyt, BranStark, Michaelbusch, PapayaSF, IDSIAupdate, CmdrObot, N2e, .Koen, Myasuda, Abdullahazzam, Cydebot, Kanags, Odie5533, Johnfn, Archange56, Markybish, Keraunos, Marek69, KrakatoaKatie, Seaphoto, Ouc, Gadlen, JAnDbot, SiobhanHansa, Magioladitis, Nyq, Quailwood, Mdorigo, MGarcia, Ehrami, Armehrabian, Pudor, Fbln, PatPeter, Dragsheets, StaticDust, Blooy, Etet245, Manticore, Jiuguang Wang, Maurice Carbonaro, NerdyNSK, K.menin, Tarotcards, DadaNeem,

Jamesontai, J ham3, Ajithabraham, LuckyInWaco, Prometheus1000, BotKung, Van Parunak, Zwgeem, Iciac, Christophe Dioux, Sockettome, Kumioko (renamed), Tesi1700, Jbw2, Squid603, Binksternet, The Thing That Should Not Be, Gottfried1646, Calimo, Bahriyebasturk, Krinndnz, Cardiffbaybee, Dobri2, XLinkBot, DrOxacropheles, Addbot, DynamicUno, Jbapowell, Fyrael, CarsracBot, Abelaragorn, Lightbot, Balabiot, Legobot, Luckas-bot, Yobot, Fraggle81, Aelyan, 4th-otaku, AnomieBOT, Trevithj, Materialscientist, DynamoD-egsy, Gacpro, Alpha for knowledge, Xqbot, TheDegreeCollector, Tcooling, Miym, RibotBOT, MuffledThud, Citation bot 1, Skyrise, Turian, Bolufe, Megaman7de, FoxBot, Sergey539, RjwilmsiBot, Scifun, Fipblizip, Kasradaneshvar, Helwr, John of Reading, Josephgec, Slightsmile, Serketan, Daryakov, AvicBot, Famontreal, Ultimamultima, Іванко1, Akira.exe, JoelShprentz, ClueBot NG, Siamaktalat, Helpful Pixie Bot, Bibcode Bot, BG19bot, Swarmcolor, Intelcrowd, Mozartee, Swarmingeek, Alimirjalili, Rahulmothiya, Mazhar ansari ardeh, ChrisGualtieri, GoShow, Khazar2, Mohmaj, Ahalavi, ፩፪, Mr.Goblins, Erkanbesdok, Turn on a Dime, Kaipakrishna, Крабвеци, Rileyberi, Buzzants, S5a4rmit, Hoodaan, Swarmshyte, Blackbombchu, Julian Bronte Glenn, Iceswi, Anrusnus, Martsami, Pinky ohne Brain, Optnova, Uwerwt, Marco castellani 1965, Kosoptie, Shahradsmd and Anonymous: 299

- **TOLMIN (optimization software)** Source: [http://en.wikipedia.org/wiki/TOLMIN%20\(optimization%20software\)?oldid=655216687](http://en.wikipedia.org/wiki/TOLMIN%20(optimization%20software)?oldid=655216687) Contributors: Zhangzk
- **Tree rearrangement** Source: <http://en.wikipedia.org/wiki/Tree%20rearrangement?oldid=593906233> Contributors: Centrx, Aranae, Hon-gooi, Mgiganteus1, Smith609, Cydebot, Opabinia regalis, PigFlu Oink, DrilBot and Anonymous: 3
- **UOBYQA** Source: <http://en.wikipedia.org/wiki/UOBYQA?oldid=655243022> Contributors: Zhangzk
- **Very large-scale neighborhood search** Source: <http://en.wikipedia.org/wiki/Very%20large-scale%20neighborhood%20search?oldid=641382369> Contributors: Nowozin, Chris the speller, SalopianJames, Cydebot, Timchippingonderrick, Jhay116, David Eppstein, Addbot, Luckas-bot, Yobot, Erik9bot, Pyschobbens, DoctorKubla and Anonymous: 2
- **Zionts–Wallenius method** Source: <http://en.wikipedia.org/wiki/Zionts%20E2%80%93Wallenius%20method?oldid=554566860> Contributors: Michael Hardy, Andreas Kaufmann, Slipperyweasel, SmackBot, LeoNomis, SwitcherCat, Cydebot, ThisIsAce, Magioladitis, JackSchmidt, Good Olfactory, Ironholds, Yobot, Arparp, AnomieBOT, Gibbjja, BattyBot and DoctorKubla

148.3.2 Images

- **File:AB_pruning.svg** Source: http://upload.wikimedia.org/wikipedia/commons/9/91/AB_pruning.svg License: CC-BY-SA-3.0 Contributors: Transferred from en.wikipedia Original artist: Original uploader was Jez9999 at en.wikipedia
- **File:Adaptive_Coordinate_Descent_illustration.png** Source: http://upload.wikimedia.org/wikipedia/commons/2/2e/Adaptive_Coordinate_Descent_illustration.png License: CC BY-SA 3.0 Contributors: Own work Original artist: Evolutionarycomputation
- **File:Ambox_important.svg** Source: http://upload.wikimedia.org/wikipedia/commons/b/b4/Ambox_important.svg License: Public domain Contributors: Own work, based off of Image:Ambox scales.svg Original artist: Dsmurat (talk · contribs)
- **File:Ambox_wikify.svg** Source: http://upload.wikimedia.org/wikipedia/commons/e/e1/Ambox_wikify.svg License: Public domain Contributors: Own work Original artist: penubag
- **File:Animation2.gif** Source: <http://upload.wikimedia.org/wikipedia/commons/c/c0/Animation2.gif> License: CC-BY-SA-3.0 Contributors: Own work Original artist: MG (talk · contribs)
- **File:Approximation_using_MACROS_GT_Approx.png** Source: http://upload.wikimedia.org/wikipedia/en/c/c5/Approximation_using_MACROS_GT_Approx.png License: CC-BY-SA-3.0 Contributors: was sent to me personally
Original artist:
DATADVANCE llc
- **File:Arithmetic_symbols.svg** Source: http://upload.wikimedia.org/wikipedia/commons/a/a3/Arithmetic_symbols.svg License: Public domain Contributors: Own work Original artist: This vector image was created with Inkscape by Elembis, and then manually replaced.
- **File:Auklet_flock_Shumagins_1986.jpg** Source: http://upload.wikimedia.org/wikipedia/commons/5/5e/Auklet_flock_Shumagins_1986.jpg License: Public domain Contributors: images.fws.gov ([1]) Original artist: D. Dibenski
- **File:Banana-SteepDesc.gif** Source: <http://upload.wikimedia.org/wikipedia/commons/6/60/Banana-SteepDesc.gif> License: CC-BY-SA-3.0 Contributors: Wikipedia en (page on gradient descent: <http://en.wikipedia.org/wiki/Image:Banana-SteepDesc.gif>) Original artist: P.A. Simionescu
- **File:Bellcurve.svg** Source: <http://upload.wikimedia.org/wikipedia/commons/d/df/Bellcurve.svg> License: Copyrighted free use Contributors: ? Original artist: ?
- **File:Bellman-Ford_worst-case_example.svg** Source: http://upload.wikimedia.org/wikipedia/commons/8/85/Bellman-Ford_worst-case_example.svg License: CC0 Contributors: Own work Original artist: User:Dcoetze
- **File:Binary_tree.svg** Source: http://upload.wikimedia.org/wikipedia/commons/f/f7/Binary_tree.svg License: Public domain Contributors: Own work Original artist: Derrick Coetze
- **File:Boruvka'{}s_algorithm_(Sollin'{}s_algorithm)_Anim.gif** Source: http://upload.wikimedia.org/wikipedia/commons/2/2e/Boruvka%27s_algorithm_%28Sollin%27s_algorithm%29_Anim.gif License: CC BY-SA 3.0 Contributors: Own work Original artist: Alieseraj
- **File:Borùvka_Algorithm_1.svg** Source: http://upload.wikimedia.org/wikipedia/commons/9/9c/Bor%C5%AFvka_Algorithm_1.svg License: CC BY-SA 3.0 Contributors: Own work, based on File:Prim Algorithm 0.svg Original artist: User:Dcoetze, User:Maksim, User:Alexander Drichel
- **File:Borùvka_Algorithm_2.svg** Source: http://upload.wikimedia.org/wikipedia/commons/5/5c/Bor%C5%AFvka_Algorithm_2.svg License: CC BY-SA 3.0 Contributors: Own work, based on File:Prim Algorithm 0.svg Original artist: User:Dcoetze, User:Maksim, User:Alexander Drichel
- **File:Borùvka_Algorithm_3.svg** Source: http://upload.wikimedia.org/wikipedia/commons/8/86/Bor%C5%AFvka_Algorithm_3.svg License: CC BY-SA 3.0 Contributors: Own work, based on File:Prim Algorithm 0.svg Original artist: User:Dcoetze, User:Maksim, User:Alexander Drichel

- **File:Branch_and_price_diagram.png** *Source:* http://upload.wikimedia.org/wikipedia/en/2/27/Branch_and_price_diagram.png *License:* CC-BY-SA-3.0 *Contributors:*
Own work
Original artist:
Bmears11
- **File:Catalan-Hexagons-example.svg** *Source:* <http://upload.wikimedia.org/wikipedia/commons/a/a8/Catalan-Hexagons-example.svg> *License:* Public domain *Contributors:* <http://en.wikipedia.org/wiki/File:Catalan-Hexagons-example.svg> *Original artist:* Dmharvey
- **File:Column_impost.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/5/57/Column_impost.svg *License:* CC-BY-SA-3.0 *Contributors:* Own work, [Image:Komposita1.png](#) *Original artist:* Ssolbergj
- **File:Commons-logo.svg** *Source:* <http://upload.wikimedia.org/wikipedia/en/4/4a/Commons-logo.svg> *License:* ? *Contributors:* ? *Original artist:* ?
- **File:Concept_of_directional_optimization_in_CMA-ES_algorithm.png** *Source:* http://upload.wikimedia.org/wikipedia/en/d/d8/Concept_of_directional_optimization_in_CMA-ES_algorithm.png *License:* PD *Contributors:*
Own work
Original artist:
Sentewolf (talk) (Uploads)
- **File:DE_Meta-Fitness_Landscape_(12_benchmark_problems).JPG** *Source:* http://upload.wikimedia.org/wikipedia/commons/7/73/DE_Meta-Fitness_Landscape_%2812_benchmark_problems%29.JPG *License:* Public domain *Contributors:* Own work *Original artist:* Pedersen, M.E.H., Tuning & Simplifying Heuristical Optimization, PhD Thesis, 2010, University of Southampton, School of Engineering Sciences, Computational Engineering and Design Group.
- **File:DE_Meta-Fitness_Landscape_(Sphere_and_Rosenbrock).JPG** *Source:* http://upload.wikimedia.org/wikipedia/commons/e/e5/DE_Meta-Fitness_Landscape_%28Sphere_and_Rosenbrock%29.JPG *License:* Public domain *Contributors:* Own work *Original artist:* Pedersen, M.E.H., Tuning & Simplifying Heuristical Optimization, PhD Thesis, 2010, University of Southampton, School of Engineering Sciences, Computational Engineering and Design Group.
- **File:DE_Meta-Optimization_Progress_(12_benchmark_problems).JPG** *Source:* http://upload.wikimedia.org/wikipedia/commons/4/4d/DE_Meta-Optimization_Progress_%2812_benchmark_problems%29.JPG *License:* Public domain *Contributors:* Own work *Original artist:* Pedersen, M.E.H., Tuning & Simplifying Heuristical Optimization, PhD Thesis, 2010, University of Southampton, School of Engineering Sciences, Computational Engineering and Design Group.
- **File:DWave_128chip.jpg** *Source:* http://upload.wikimedia.org/wikipedia/commons/1/17/DWave_128chip.jpg *License:* CC BY 3.0 *Contributors:* D-Wave Systems, Inc. *Original artist:* D-Wave Systems, Inc.
- **File:Design_of_Experiment.png** *Source:* http://upload.wikimedia.org/wikipedia/en/9/95/Design_of_Experiment.png *License:* CC-BY-SA-3.0 *Contributors:*
was sent to me personally
Original artist:
DATADVANCE llc
- **File:Dijkstra_Animation.gif** *Source:* http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif *License:* Public domain *Contributors:* Work by uploader. *Original artist:* Ibmua
- **File:Dijkstras_progress_animation.gif** *Source:* http://upload.wikimedia.org/wikipedia/commons/2/23/Dijkstras_progress_animation.gif *License:* CC BY 3.0 *Contributors:* Own work *Original artist:* Subh83
- **File:Dinic_algorithm_G1.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/3/37/Dinic_algorithm_G1.svg *License:* Public domain *Contributors:* Transferred from en.wikipedia
Original artist: Chin Ho Lee. Original uploader was Tchhasaposse at en.wikipedia
- **File:Dinic_algorithm_G2.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/5/56/Dinic_algorithm_G2.svg *License:* Public domain *Contributors:* Transferred from en.wikipedia
Original artist: Chin Ho Lee. Original uploader was Tchhasaposse at en.wikipedia
- **File:Dinic_algorithm_G3.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/7/71/Dinic_algorithm_G3.svg *License:* Public domain *Contributors:* Transferred from en.wikipedia
Original artist: Chin Ho Lee. Original uploader was Tchhasaposse at en.wikipedia
- **File:Dinic_algorithm_GL1.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/8/80/Dinic_algorithm_GL1.svg *License:* Public domain *Contributors:* Transferred from en.wikipedia
Original artist: Chin Ho Lee. Original uploader was Tchhasaposse at en.wikipedia
- **File:Dinic_algorithm_GL2.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/9/97/Dinic_algorithm_GL2.svg *License:* Public domain *Contributors:* Transferred from en.wikipedia
Original artist: Chin Ho Lee. Original uploader was Tchhasaposse at en.wikipedia
- **File:Dinic_algorithm_GL3.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/9/95/Dinic_algorithm_GL3.svg *License:* Public domain *Contributors:* Transferred from en.wikipedia
Original artist: Chin Ho Lee. Original uploader was Tchhasaposse at en.wikipedia
- **File:Dinic_algorithm_Gf1.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/f/fd/Dinic_algorithm_Gf1.svg *License:* Public domain *Contributors:* Transferred from en.wikipedia
Original artist: Chin Ho Lee. Original uploader was Tchhasaposse at en.wikipedia
- **File:Dinic_algorithm_Gf2.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/4/43/Dinic_algorithm_Gf2.svg *License:* Public domain *Contributors:* Transferred from en.wikipedia
Original artist: Chin Ho Lee. Original uploader was Tchhasaposse at en.wikipedia

- **File:Dinic_algorithm_Gf3.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/2/20/Dinic_algorithm_Gf3.svg *License:* Public domain *Contributors:* Transferred from en.wikipedia
Original artist: Chin Ho Lee. Original uploader was Tcshasapose at en.wikipedia
- **File:Direct_search_BROYDEN.gif** *Source:* http://upload.wikimedia.org/wikipedia/commons/1/14/Direct_search_BROYDEN.gif *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Guillaume Jacquetot
- **File:Dykstra_algorithm.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/0/02/Dykstra_algorithm.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Claudio Rocchini
- **File:EM_Clustering_of_Old_Faithful_data.gif** *Source:* http://upload.wikimedia.org/wikipedia/commons/6/69/EM_Clustering_of_Old_Faithful_data.gif *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Chire
- **File>Edit-clear.svg** *Source:* <http://upload.wikimedia.org/wikipedia/en/f/f2/Edit-clear.svg> *License:* Public domain *Contributors:* The Tango! Desktop Project. *Original artist:*
The people from the Tango! project. And according to the meta-data in the file, specifically: “Andreas Nilsson, and Jakub Steiner (although minimally).”
- **File:Edmonds-Karp_flow_example_0.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/3/3e/Edmonds-Karp_flow_example_0.svg *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en:User:Cburnett
- **File:Edmonds-Karp_flow_example_1.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/6/6d/Edmonds-Karp_flow_example_1.svg *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en:User:Cburnett
- **File:Edmonds-Karp_flow_example_2.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/c/c1/Edmonds-Karp_flow_example_2.svg *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en:User:Cburnett
- **File:Edmonds-Karp_flow_example_3.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/a/a5/Edmonds-Karp_flow_example_3.svg *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en:User:Cburnett
- **File:Edmonds-Karp_flow_example_4.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/b/bd/Edmonds-Karp_flow_example_4.svg *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en:User:Cburnett
- **File:Ellipsoid_1.png** *Source:* http://upload.wikimedia.org/wikipedia/en/6/6a/Ellipsoid_1.png *License:* PD *Contributors:* ? *Original artist:* ?
- **File:Ellipsoid_2.png** *Source:* http://upload.wikimedia.org/wikipedia/en/f/ff/Ellipsoid_2.png *License:* PD *Contributors:* ? *Original artist:* ?
- **File:Ellipsoid_3.png** *Source:* http://upload.wikimedia.org/wikipedia/en/c/c1/Ellipsoid_3.png *License:* PD *Contributors:* ? *Original artist:* ?
- **File:Ellipsoid_4.png** *Source:* http://upload.wikimedia.org/wikipedia/en/f/fd/Ellipsoid_4.png *License:* PD *Contributors:* ? *Original artist:* ?
- **File:Ellipsoid_5.png** *Source:* http://upload.wikimedia.org/wikipedia/en/d/d0/Ellipsoid_5.png *License:* PD *Contributors:* ? *Original artist:* ?
- **File:Ellipsoid_6.png** *Source:* http://upload.wikimedia.org/wikipedia/en/5/5a/Ellipsoid_6.png *License:* PD *Contributors:* ? *Original artist:* ?
- **File:Em_old_faithful.gif** *Source:* http://upload.wikimedia.org/wikipedia/commons/a/a7/Em_old_faithful.gif *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* 3mta3 (talk) 16:55, 23 March 2009 (UTC)
- **File:Factory.svg** *Source:* <http://upload.wikimedia.org/wikipedia/commons/a/a2/Factory.svg> *License:* Public domain *Contributors:* Self-made, taken from Image:1 9 2 9.svg *Original artist:* Howard Cheng
- **File:Fibonacci_dynamic_programming.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/0/06/Fibonacci_dynamic_programming.svg *License:* Public domain *Contributors:* en:Image:Fibonacci dynamic programming.png *Original artist:* en:User:Dcoetzee, traced by User:Stannered
- **File:Fisher_iris_versicolor_sepalwidth.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/4/40/Fisher_iris_versicolor_sepalwidth.svg *License:* CC BY-SA 3.0 *Contributors:* en:Image:Fisher iris versicolor sepalwidth.png *Original artist:* en:User:Qwfp (original); Pbroks13 (talk) (redraw)
- **File:Floyd-Warshall_example.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/2/2e/Floyd-Warshall_example.svg *License:* CCO *Contributors:* Own work *Original artist:* User:Dcoetzee
- **File:Folder_Hexagonal_Icon.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/4/48/Folder_Hexagonal_Icon.svg *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?
- **File:Ford-Fulkerson_example_0.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/f/f1/Ford-Fulkerson_example_0.svg *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en:User:Cburnett
- **File:Ford-Fulkerson_example_1.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/9/93/Ford-Fulkerson_example_1.svg *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en:User:Cburnett
- **File:Ford-Fulkerson_example_2.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/1/13/Ford-Fulkerson_example_2.svg *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en:User:Cburnett
- **File:Ford-Fulkerson_example_final.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/f/f2/Ford-Fulkerson_example_final.svg *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en:User:Cburnett
- **File:Ford-Fulkerson_forever.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/6/6e/Ford-Fulkerson_forever.svg *License:* CC BY 3.0 *Contributors:* Own work, generated by dot from Graphviz with the following code: *Original artist:* Svick
- **File:Frank-Wolfe_Algorithm.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/e/e5/Frank-Wolfe_Algorithm.png *License:* CC BY 4.0 *Contributors:* Made by Stephanie Stutz for public domain, labels added by Martin Jaggi *Original artist:* Stephanie Stutz

- **File:Gauss_Newton_illustration.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/5/5e/Gauss_Newton_illustration.png *License:* Public domain *Contributors:* self-made with Matlab *Original artist:* Oleg Alexandrov
- **File:Gold,_silver,_and_bronze_rectangles_vertical.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/c/c1/Gold%2C_silver%2C_and_bronze_rectangles_vertical.png *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Hyacinth
- **File:GoldenSectionSearch.png** *Source:* <http://upload.wikimedia.org/wikipedia/commons/5/52/GoldenSectionSearch.png> *License:* CC-BY-SA-3.0 *Contributors:* ? *Original artist:* ?
- **File:Gradient_Decent_Example_Nonlinear_Equations.gif** *Source:* http://upload.wikimedia.org/wikipedia/commons/0/01/Gradient_Decent_Example_Nonlinear_Equations.gif *License:* Public domain *Contributors:* Own work *Original artist:* Peryeat
- **File:Gradient_ascent_(contour).png** *Source:* http://upload.wikimedia.org/wikipedia/commons/d/db/Gradient_ascent_%28contour%29.png *License:* Public domain *Contributors:* Created with Maple 10, using the code below: *Original artist:* user:Joris Gillis
- **File:Gradient_ascent_(surface).png** *Source:* http://upload.wikimedia.org/wikipedia/commons/6/68/Gradient_ascent_%28surface%29.png *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Gradient_descent.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/f/ff/Gradient_descent.svg *License:* Public domain *Contributors:* This file was derived from: Gradient descent.png: *Original artist:* Gradient_descent.png: The original uploader was Olegalexandrov at English Wikipedia
- **File:Graduated_optimization.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/b/be/Graduated_optimization.svg *License:* PD *Contributors:*
I (Headlessplatter (talk)) created this work entirely by myself. *Original artist:*
Headlessplatter (talk)
- **File:Greedy-search-path-example.gif** *Source:* <http://upload.wikimedia.org/wikipedia/commons/8/8c/Greedy-search-path-example.gif> *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Swfung8
- **File:Greedy_Glouton.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/0/06/Greedy_Glouton.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Tos
- **File:Greedy_algorithm_36_cents.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/d/da/Greedy_algorithm_36_cents.svg *License:* Public domain *Contributors:* Own work *Original artist:* Nandhp
- **File:Green_bug_and_broom.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/8/83/Green_bug_and_broom.svg *License:* LGPL *Contributors:* File:Broom icon.svg, file:Green_bug.svg *Original artist:* Poznaniak, pozostali autorzy w plikach źródłowych
- **File:Hill_Climbing_with_Simulated_Annealing.gif** *Source:* http://upload.wikimedia.org/wikipedia/commons/d/d5/Hill_Climbing_with_Simulated_Annealing.gif *License:* CC0 *Contributors:* Own work *Original artist:* Kingpin13
- **File:Hill_climb.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/0/05/Hill_climb.png *License:* Public domain *Contributors:* Own work // Transferred from en.wikipedia; transferred to Commons by User:Sreejithk2000 using CommonsHelper. *Original artist:* Original uploader was Headlessplatter at en.wikipedia
- **File:Honey_Bee_takes_Nectar.JPG** *Source:* http://upload.wikimedia.org/wikipedia/commons/a/a6/Honey_Bee_takes_Nectar.JPG *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Sajjad Fazel
- **File:IP_polytope_with_LP_relaxation.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/f/fd/IP_polytope_with_LP_relaxation.png *License:* CC BY-SA 2.5 *Contributors:* Self-made using xfig and fig2dev (see <http://www.xfig.org/>) under Linux. The .fig-files can be obtained from me upon request. *Original artist:* Sdo
- **File:Imperialist-competitive-algorithm-flowchart.jpg** *Source:* <http://upload.wikimedia.org/wikipedia/commons/c/cd/Imperialist-competitive-algorithm-flowchart.jpg> *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Icasite
- **File:Internet_map_1024.jpg** *Source:* http://upload.wikimedia.org/wikipedia/commons/d/d2/Internet_map_1024.jpg *License:* CC BY 2.5 *Contributors:* Originally from the English Wikipedia; description page is/was here. *Original artist:* The Opte Project
- **File:Iterated_local_search.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/3/34/Iterated_local_search.png *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Roberto Battiti
- **File:Johnson'{}s_algorithm.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/4/4f/Johnson%27s_algorithm.svg *License:* Public domain *Contributors:* Own work *Original artist:* David Eppstein
- **File:Karmarkar.png** *Source:* <http://upload.wikimedia.org/wikipedia/en/8/8c/Karmarkar.png> *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Kruskal_Algorithm_1.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/b/b4/Kruskal_Algorithm_1.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Kruskal_Algorithm_2.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/f/ff/Kruskal_Algorithm_2.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Kruskal_Algorithm_3.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/5/59/Kruskal_Algorithm_3.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Kruskal_Algorithm_4.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/2/2e/Kruskal_Algorithm_4.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Kruskal_Algorithm_5.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/f/f4/Kruskal_Algorithm_5.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Kruskal_Algorithm_6.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/8/87/Kruskal_Algorithm_6.svg *License:* Public domain *Contributors:* ? *Original artist:* ?

- **File:LampFlowchart.svg** *Source:* <http://upload.wikimedia.org/wikipedia/commons/9/91/LampFlowchart.svg> *License:* CC-BY-SA-3.0
Contributors: vector version of Image:LampFlowchart.png *Original artist:* svg by Booyabazooka
- **File:Lebesgue_Icon.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/c/c9/Lebesgue_Icon.svg *License:* Public domain *Contributors:* w:Image:Lebesgue_Icon.svg *Original artist:* w:User:James pic
- **File:Lev-Mar-best-fit.png** *Source:* <http://upload.wikimedia.org/wikipedia/en/0/02/Lev-Mar-best-fit.png> *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?
- **File:Lev-Mar-better-fit.png** *Source:* <http://upload.wikimedia.org/wikipedia/en/3/34/Lev-Mar-better-fit.png> *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?
- **File:Lev-Mar-poor-fit.png** *Source:* <http://upload.wikimedia.org/wikipedia/en/0/05/Lev-Mar-poor-fit.png> *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?
- **File:Local_maximum.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/7/7e/Local_maximum.png *License:* Public domain *Contributors:* Transferred from en.wikipedia; transferred to Commons by User:Shashenka using CommonsHelper. *Original artist:* Original uploader was Headlessplatter at en.wikipedia
- **File:MST_kruskal_en.gif** *Source:* http://upload.wikimedia.org/wikipedia/commons/5/5c/MST_kruskal_en.gif *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Schullz
- **File:Matrix_chain_multiplication_polygon_example.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/e/ea/Matrix_chain_multiplication_polygon_example.svg *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* R.J.C. van Haften
- **File:Max-flow_min-cut_example.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/0/0e/Max-flow_min-cut_example.svg *License:* PD *Contributors:* self-created by Graphviz.
Original artist:
Chin Ho Lee
- **File:MaximumParaboloid.png** *Source:* <http://upload.wikimedia.org/wikipedia/commons/6/62/MaximumParaboloid.png> *License:* CC-BY-SA-3.0 *Contributors:* Transferred from en.wikipedia; transferred to Commons by User:Enen using CommonsHelper. *Original artist:* Original uploader was Sam Derbyshire at en.wikipedia
- **File:Meta-Optimization_Concept.JPG** *Source:* http://upload.wikimedia.org/wikipedia/commons/b/b0/Meta-Optimization_Concept.JPG *License:* Public domain *Contributors:* Own work *Original artist:* Pedersen, M.E.H., Tuning & Simplifying Heuristical Optimization, PhD Thesis, 2010, University of Southampton, School of Engineering Sciences, Computational Engineering and Design Group.
- **File:Minimax.svg** *Source:* <http://upload.wikimedia.org/wikipedia/commons/6/6f/Minimax.svg> *License:* CC BY-SA 2.5 *Contributors:* <http://en.wikipedia.org/wiki/Image:Minimax.svg>, created in Inkscape by author *Original artist:* Nuno Nogueira (Nmogueria)
- **File:Minmaxab.gif** *Source:* <http://upload.wikimedia.org/wikipedia/en/7/79/Minmaxab.gif> *License:* CC-BY-SA-3.0 *Contributors:* I created this work entirely by myself!
Original artist:
Maschelos (talk)
- **File:Mmalgorithm.jpg** *Source:* <http://upload.wikimedia.org/wikipedia/commons/c/cc/Mmalgorithm.jpg> *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Peng Wang
- **File:Negamax_AlphaBeta.gif** *Source:* http://upload.wikimedia.org/wikipedia/en/f/ff/Negamax_AlphaBeta.gif *License:* CC-BY-SA-3.0 *Contributors:* I (Maschelos (talk)) created this work entirely by myself. *Original artist:*
Maschelos (talk)
- **File:Nelder_Mead1.gif** *Source:* http://upload.wikimedia.org/wikipedia/commons/0/09/Nelder_Mead1.gif *License:* CC-BY-SA-3.0 *Contributors:* Transferred from en.wikipedia *Original artist:* Original uploader was Simiprof at en.wikipedia
- **File:Nelder_Mead2.gif** *Source:* http://upload.wikimedia.org/wikipedia/commons/9/96/Nelder_Mead2.gif *License:* CC-BY-SA-3.0 *Contributors:* Transferred from en.wikipedia *Original artist:* Original uploader was Simiprof at en.wikipedia
- **File:NewtonIteration_Ani.gif** *Source:* http://upload.wikimedia.org/wikipedia/commons/e/e0/NewtonIteration_Ani.gif *License:* CC-BY-SA-3.0 *Contributors:* de:Image:NewtonIteration Ani.gif *Original artist:* Ralf Pfeifer
- **File:Newton_optimization_vs_grad_descent.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/d/da/Newton_optimization_vs_grad_descent.svg *License:* Public domain *Contributors:* self-made with en:Matlab. Tweaked in en:Inkscape *Original artist:* Oleg Alexandrov
- **File:NewtonsMethodConvergenceFailure.svg** *Source:* <http://upload.wikimedia.org/wikipedia/commons/f/f1/NewtonsMethodConvergenceFailure.svg> *License:* Public domain *Contributors:* Self-made, created using FooPlot and modified in Inkscape. *Original artist:* Aaron Rotenberg
- **File:Newtroot_1_0_0_0_0_m1.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/b/bf/Newtroot_1_0_0_0_0_m1.png *License:* GPL *Contributors:* ? *Original artist:* ?
- **File:Nonlinear_programming.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/1/10/Nonlinear_programming.svg *License:* CC0 *Contributors:* Own work *Original artist:* Krishnavedala
- **File:Nonlinear_programming_3D.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/4/4c/Nonlinear_programming_3D.svg *License:* Public domain *Contributors:*
- Nonlinear_programming_3D_jaredwf.png *Original artist:*
- derivative work: McSush (talk)

- **File:Nuvola_apps_edu_mathematics_blue-p.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/3/3e/Nuvola_apps_edu_mathematics_blue-p.svg *License:* GPL *Contributors:* Derivative work from Image:Nuvola apps edu mathematics.png and Image:Nuvola apps edu mathematics-p.svg *Original artist:* David Vignoni (original icon); Flamura (SVG conversion); bayo (color)
- **File:PSO_Meta-Fitness_Landscape_(12_benchmark_problems).JPG** *Source:* http://upload.wikimedia.org/wikipedia/commons/a/ac/PSO_Meta-Fitness_Landscape_%2812_benchmark_problems%29.JPG *License:* Public domain *Contributors:* Own work *Original artist:* Pedersen, M.E.H., Tuning & Simplifying Heuristical Optimization, PhD Thesis, 2010, University of Southampton, School of Engineering Sciences, Computational Engineering and Design Group.
- **File:PSeven_logo.png** *Source:* http://upload.wikimedia.org/wikipedia/en/b/bc/PSeven_logo.png *License:* Fair use *Contributors:* I work for the owner *Original artist:* ?
- **File:Parallel_models.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/d/d1/Parallel_models.png *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Enrique.alba1
- **File:Plain_Negamax.gif** *Source:* http://upload.wikimedia.org/wikipedia/en/1/16/Plain_Negamax.gif *License:* CC-BY-SA-3.0 *Contributors:*
I (Maschelos (talk)) created this work entirely by myself. *Original artist:* Maschelos (talk)
- **File:Plminmax.gif** *Source:* <http://upload.wikimedia.org/wikipedia/commons/e/e1/Plminmax.gif> *License:* CC BY-SA 3.0 *Contributors:* Own work (Original text: *I created this work entirely by myself.*) *Original artist:* Maschelos (talk)
- **File:Portal-puzzle.svg** *Source:* <http://upload.wikimedia.org/wikipedia/en/f/fd/Portal-puzzle.svg> *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Push-Relabel_Algorithm_Example_-_Final_Network_Graph.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/8/8b/Push-Relabel_Algorithm_Example_-_Final_Network_Graph.svg *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Kamac124
- **File:Push-Relabel_Algorithm_Example_-_Step_1.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/0/0f/Push-Relabel_Algorithm_Example_-_Step_1.svg *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Kamac124
- **File:Push-Relabel_Algorithm_Example_-_Step_2.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/9/9f/Push-Relabel_Algorithm_Example_-_Step_2.svg *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Kamac124
- **File:Push-Relabel_Algorithm_Example_-_Step_3.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/1/11/Push-Relabel_Algorithm_Example_-_Step_3.svg *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Kamac124
- **File:Push-Relabel_Algorithm_Example_-_Step_4.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/7/7a/Push-Relabel_Algorithm_Example_-_Step_4.svg *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Kamac124
- **File:Push-Relabel_Algorithm_Example_-_Step_5.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/d/d5/Push-Relabel_Algorithm_Example_-_Step_5.svg *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Kamac124
- **File:Push-Relabel_Algorithm_Example_-_Step_6.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/b/bb/Push-Relabel_Algorithm_Example_-_Step_6.svg *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Kamac124
- **File:Push-Relabel_Algorithm_Example_-_Step_7.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/7/7e/Push-Relabel_Algorithm_Example_-_Step_7.svg *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Kamac124
- **File:Push-Relabel_Algorithm_Example_-_Step_8.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/2/25/Push-Relabel_Algorithm_Example_-_Step_8.svg *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Kamac124
- **File:Push-Relabel_Algorithm_Example_-_Step_9.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/8/82/Push-Relabel_Algorithm_Example_-_Step_9.svg *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Kamac124
- **File:Push_Relabel_Algorithm_Example_-_Initial_Graph.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/f/fc/Push_Relabel_Algorithm_Example_-_Initial_Graph.svg *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Kamac124
- **File:Quant-annl.jpg** *Source:* <http://upload.wikimedia.org/wikipedia/commons/1/12/Quant-annl.jpg> *License:* Public domain *Contributors:* Transferred from en.wikipedia; transfer was stated to be made by User:Ebe123. *Original artist:* author: Arnab Das (Theoretical Condensed Matter Physics Division and Centre for Applied Mathematics and Computational Science, Saha Institute of Nuclear Physics, Kolkata, India). Original uploader was Arnab das at en.wikipedia
- **File:Question_book-new.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/9/99/Question_book-new.svg *License:* Cc-by-sa-3.0 *Contributors:*
Created from scratch in Adobe Illustrator. Based on Image:Question book.png created by User:Equazcion *Original artist:* Tkgd2007
- **File:Random_sampling_unimodal_function_(1).JPG** *Source:* http://upload.wikimedia.org/wikipedia/commons/a/a5/Random_sampling_unimodal_function_%281%29.JPG *License:* Public domain *Contributors:* Own work *Original artist:* Pedersen, M.E.H., Tuning & Simplifying Heuristical Optimization, PhD Thesis, 2010, University of Southampton, School of Engineering Sciences, Computational Engineering and Design Group.
- **File:Random_sampling_unimodal_function_(2).JPG** *Source:* http://upload.wikimedia.org/wikipedia/commons/0/01/Random_sampling_unimodal_function_%282%29.JPG *License:* Public domain *Contributors:* Own work *Original artist:* Pedersen, M.E.H., Tuning & Simplifying Heuristical Optimization, PhD Thesis, 2010, University of Southampton, School of Engineering Sciences, Computational Engineering and Design Group.
- **File:Ridge.png** *Source:* <http://upload.wikimedia.org/wikipedia/en/3/37/Ridge.png> *License:* CC0 *Contributors:* ? *Original artist:* ?
- **File:Rosenbrock2D.png** *Source:* <http://upload.wikimedia.org/wikipedia/commons/5/58/Rosenbrock2D.png> *License:* Public domain *Contributors:* my research work *Original artist:* Ilya Loshchilov
- **File:Shortest_path_optimal_substructure.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/0/03/Shortest_path_optimal_substructure.svg *License:* CC0 *Contributors:* http://en.wikipedia.org/wiki/File:Shortest_path_optimal_substructure.png *Original artist:* User:Dcoetze

- **File:Simplex-description-en.svg** *Source:* <http://upload.wikimedia.org/wikipedia/commons/e/ef/Simplex-description-en.svg> *License:* CC BY-SA 3.0 *Contributors:* https://commons.wikimedia.org/wiki/File:Simplex_description.png *Original artist:* Victor Treushchenko
- **File:Simplex-method-3-dimensions.png** *Source:* <http://upload.wikimedia.org/wikipedia/commons/d/d4/Simplex-method-3-dimensions.png> *License:* CC-BY-SA-3.0 *Contributors:* Created using gimp based on Image:Elongated_pentagonal_orthocupolarotunda.png. *Original artist:* User:Sdo
- **File:Simplex_description.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/7/78/Simplex_description.png *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:SimulatedAnnealingFast.jpg** *Source:* <http://upload.wikimedia.org/wikipedia/commons/8/82/SimulatedAnnealingFast.jpg> *License:* CC-BY-SA-3.0 *Contributors:* Transferred from en.wikipedia; transferred to Commons by User:Sfan00_IMG using CommonsHelper. *Original artist:* Original uploader was Cyp at en.wikipedia
- **File:SimulatedAnnealingSlow.jpg** *Source:* <http://upload.wikimedia.org/wikipedia/commons/8/8d/SimulatedAnnealingSlow.jpg> *License:* CC-BY-SA-3.0 *Contributors:* Transferred from en.wikipedia; transferred to Commons by User:Sfan00_IMG using CommonsHelper. *Original artist:* Original uploader was Cyp at en.wikipedia
- **File:Software_spinner.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/8/82/Software_spinner.png *License:* CC-BY-SA-3.0 *Contributors:* Transferred from en.wikipedia; transfer was stated to be made by User:Rockfang. *Original artist:* Original uploader was CharlesC at en.wikipedia
- **File:St_5-xband-antenna.jpg** *Source:* http://upload.wikimedia.org/wikipedia/commons/f/ff/St_5-xband-antenna.jpg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Text_document_with_red_question_mark.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/a/a4/Text_document_with_red_question_mark.svg *License:* Public domain *Contributors:* Created by bdesham with Inkscape; based upon Text-x-generic.svg from the Tango project. *Original artist:* Benjamin D. Esham (bdesham)
- **File:Tower_of_Hanoi.jpg** *Source:* http://upload.wikimedia.org/wikipedia/commons/0/07/Tower_of_Hanoi.jpeg *License:* CC-BY-SA-3.0 *Contributors:* ? *Original artist:* ?
- **File:Tower_of_Hanoi_4.gif** *Source:* http://upload.wikimedia.org/wikipedia/commons/6/60/Tower_of_Hanoi_4.gif *License:* CC BY-SA 2.5 *Contributors:* Own work *Original artist:* André Karwath aka Aka
- **File:Translation_to_english_arrow.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/8/8a/Translation_to_english_arrow.svg *License:* CC-BY-SA-3.0 *Contributors:* Transferred from en.wikipedia; transferred to Commons by User:Faigl.ladislav using CommonsHelper. *Original artist:* tkgd2007. Original uploader was Tkgd2007 at en.wikipedia
- **File:Unbalanced_scales.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/f/fe/Unbalanced_scales.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Unitcube.svg** *Source:* <http://upload.wikimedia.org/wikipedia/commons/8/89/Unitcube.svg> *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Visualization_tools_in_pSeven.png** *Source:* http://upload.wikimedia.org/wikipedia/en/8/88/Visualization_tools_in_pSeven.png *License:* CC-BY-SA-3.0 *Contributors:*
was sent to me personally
Original artist:
DATADVANCE llc
- **File:Wiki_contractor_after.jpg** *Source:* http://upload.wikimedia.org/wikipedia/commons/b/b6/Wiki_contractor_after.jpg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Luc Jaulin
- **File:Wiki_contractor_before.jpg** *Source:* http://upload.wikimedia.org/wikipedia/commons/4/4c/Wiki_contractor_before.jpg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Luc Jaulin
- **File:Wiki_letter_w.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/6/6c/Wiki_letter_w.svg *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?
- **File:Wiki_letter_w_cropped.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/1/1c/Wiki_letter_w_cropped.svg *License:* CC-BY-SA-3.0 *Contributors:*
- **Wiki_letter_w.svg** *Original artist:* Wiki_letter_w.svg: Jarkko Piironen
- **File:Wikibooks-logo-en-noslogan.svg** *Source:* <http://upload.wikimedia.org/wikipedia/commons/d/df/Wikibooks-logo-en-noslogan.svg> *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* User:Bastique, User:Ramac et al.
- **File:Workflow_as_a_ready_tool.png** *Source:* http://upload.wikimedia.org/wikipedia/en/e/ea/Workflow_as_a_ready_tool.png *License:* CC-BY-SA-3.0 *Contributors:*
available online
Original artist:
DATADVANCE llc

148.3.3 Content license

- Creative Commons Attribution-Share Alike 3.0