ECE 457A Course Notes
Cooperative and Adaptive Algorithms

Michael Socha

4A Software Engineering
University of Waterloo
Spring 2018

Contents

1 Course Overview

1.1 Logistics

- **Professor:** Allaa Hilal

1.2 Overview of Topics

This is a course that approaches artificial intelligence by examining various algorithms that adapt to their environment and cooperate with one another. Topics include:

- Algorithms that exhibit intelligent behavior
- Concepts of adaptation and cooperation between algorithms
- Meta-heuristics, evolutionary computing methods, swarm intelligence, ant-colony algorithms, and particle swarm methods
- Apply the above ideas to solve various (often ill-structured) continuous and discrete problems

2 Introduction

2.1 Overview of Artificial Intelligence

Intelligence is the ability to acquire and apply knowledge and skills. Artificial intelligence (AI) is the science of creating intelligent machines, including intelligent computer programs. Sample applications include visual perception, speech recognition and various forms of decision-making

2.1.1 Artificial Intelligence vs Machine Learning vs Deep Learning vs Data Science

The above terms are often thrown around almost interchangealbly, but they refer to different things. Deep learning is a type of machine learning, which is a type of AI. Data science is a separate field that has some overlap with these three other concepts.

2.1.2 Inspiration from Nature

Some of the inspirations for intelligent systems come from the natural world. Sample inspirations include ant path-finding, bird flocking, and fish schooling.

2.2 Thinking Rationally vs Behaving Rationally

In rational thinking, logical systems are used to achieve goals via inferencing. It can be hard to represent informal knowledge, and not all problems can be solved through such methods (e.g. problems with uncertainty).

In rational behavior, a perceives its environment and acts to achieve goals according to some set of beliefs. This is a more general approach than inferencing, and actions taken to achieve a goal may not necessarily be optimal or "correct", but may be an acceptable solution anyways.

2.3 Swarm Intelligence

Swarm intelligence is an AI technique that builds systems based on the collective behavior of decentralized, self-organized units. There are no centralized control structures, with agents only interacting with each other and the environment.

2.4 Intelligent Agents

An agent is something that senses its environment and acts on the collected information. A rational agent acts in a way that is expected to maximize performance on the basis of perceptual history and built-in knowledge. Types of agents include:

- **Simple Reflex Agents:** Follow a lookup-table approach; needs fully observable environment
- **Model-Based Reflex Agents:** Add state information to handle partially observable environments
- **Goal-Based Agents:** Add concept of goals to help choose actions
- **Utility-Based Agents:** Add utility to decide "good" or "'bad" when faced with conflicting goals
- **Learning Agents:** Add ability to learn from experience to improve performance

2.5 Environments

The environments in which agents operate influence the design of the agents. Types of environments include:

- **Fully vs Partially Observable:** In fully observable environments, sensors can detect all aspects relevant to choice of action. This is not the case in partially observable environments, which may exist because of missing information or inaccurate sensors.
- **Deterministic vs Stochastic:** Deterministic environments are only influenced by their current state and the next action executed by the agent; otherwise, the environment is stochastic.
- **Episodic vs Sequential:** In episodic environments, the choice of action in each episode does not depend on previous episodes, unlike in sequential environments, where an agent needs to "think ahead".
- **Static vs Dynamic:** Environments where the state may change while the agent is deliberating are considered dynamic; other environments are considered static.
- **Discrete vs Continuous:** A discrete/continuous distinction can be applied to various aspects of an environment, including the way time flow is handled and to the actions of the agent.
- **Single vs Multi Agent**

2.6 Adaptive and Cooperative Algorithms

Adaptive algorithms are able to change their behavior as they are run. Cooperative algorithms algorithms work together to solve a joint problem, communicating either directly or indirectly with one another.

3 Search Problem Formulation

The first step in solving a search problem lies in formulation. Sources of potential complexity in formalizing and solving real-world search problems include multimodality, lack of a mathematical model, non-differentiability, and a combinatoric or distributed nature.

3.1 Well-Structured vs Ill-Structured Problems

In well-structured problems, the existing and end state are well-defined, and so are the actions that can be taken to change state. In ill-structured problems, the existing state is well-defined, but the end state and actions to change state are typically undefined.

3.2 Optimization Problems

Optimization problems concern finding the "best" solution from a set of solutions subject to a set of constraints. Optimization techniques are search methods where the goal is to find a solution to an optimization problem. Optimization techniques can be divided into exact algorithms, where an optimal solution is found but often at a very high computational cost, and approximate solutions, where a near-optimal solution is found at a far lower computational cost.

3.2.1 Approximation Methods

Due to their relatively low computational costs, approximation methods are often preferred for solving problems of a combinatorial nature where heuristics can be applied to find near-optimal solutions. Approaches to approximation can be divided into constructive methods, where a solution is built up one component at a time, and local search methods, where actions are iteratively applied to some initial solution to attempt to improve it.

3.3 Goal and Problem Formulation

In goal formulation, decisions are made on which aspects of a problem are important and which can be ignored. Problem formulation concerns determining how to manipulate these important aspects. Goal formulation followed by problem formulation yields the following information:

- **State space:** Complete or partial configuration of a problem
- **Initial state:** State in which search begins
- **Goal state:** State in which search ends
- **Action set:** Set of possible transitions between states

- **Cost:** Can be used to compare various solutions

Generally speaking, a closed-world assumption can be followed, meaning that each state is a complete description of the world. Environments in closed-world problems are typically fully observable, deterministic, sequential, static, and discrete.

3.4 Properties of Search Algorithms

The key properties of search algorithms are:

- **Completeness:** Whether algorithm is guaranteed to find goal node, provided one exists
- **Optimality:** Whether algorithm is guaranteed to find the best goal node (i.e. one with cheapest cost)
- **Time complexity:** How many nodes generated
- **Space complexity:** Maximum number of nodes stored in memory

4 Graph Search Algorithms

4.1 Search Tree Terminology

- **Node:** Represents a state in a search problem
- **Edge:** Represents a transition between states
- **Branching factor (b):** Maximum number of child nodes extending from a parent node
- **Maximum depth (m):** Number of edges in the shortest path from the root node to the furthest node. Variable d is used to represent the depth of the solution.

4.2 Generic Search

Given a graph, generic search algorithms repeatedly pick a node and expand a search to its children. Different algorithms differ in how they select the next node to consider. Many algorithms maintain some queue-like structure for storing nodes on a fringe to be expanded.

4.3 Uninformed vs Informed Search

Uninformed search algorithms do not have any idea of the direction of the goal node, and simply expand nodes until they reach the goal. Informed search algorithms apply some domain knowledge to determine the general direction of goal nodes. These informed algorithms are also known as "heuristic search" algorithms.

4.4 Breadth-first Search

Breadth-first search (BFS) operates by expanding the shallowest unexpanded nodes, storing the fringe to be expanded in a FIFO queue.

4.4.1 Properties

- Complete
- Optimal if cost is equal to depth, otherwise may be non-optimal
- Time complexity $O(b^{d+1})$
- Space complexity $O(b^{d+1})$

4.5 Uniform Cost Search

Uniform cost search operates similarly to BFS, but instead of expanding the shallowest unexpanded node, expands the lowest cost unexpanded node. Note that BFS is a variant of uniform cost search where all edges have uniform cost.

4.5.1 Properties

Let C be the cost of shortest path to the goal node, and other actions have a minimum cost of ϵ.

- Complete
- Optimal
- Time complexity $O(b^{\frac{C}{\epsilon}+1})$
- Space complexity $O(b^{\frac{C}{\epsilon}+1})$

4.6 Depth-first Search

Depth-first search (DFS) operates by expanding the deepest unexpanded nodes, storing nodes to be expanded in a LIFO stack.

4.6.1 Properties

- May be incomplete in infinite-depth search spaces, or in spaces with loops
- Non-optimal
- Time complexity $O(b^m)$, note that m may be much larger than d.
- Space complexity $O(bm)$

4.7 Depth-limited Search

Depth-limited search operates like DFS but imposes a maximum depth on the search.

4.7.1 Properties

Let l be the maximum depth of the search.

- Complete if there is a solution with depth $d \leq m$.
- Non-optimal

- Time complexity $O(b^l)$
- Space complexity $O(bl)$

4.8 Iterative Deepening Search

Iterative deepening search operates by repeating a depth-limited search with a larger and larger search depth l until a solution if found.

4.8.1 Properties

- Complete if b is finite.
- Guaranteed to return shallowest goal (similar to BFS), so optimal if cost is equal to depth
- Time complexity $O(b^d)$, note that m may be much larger than d.
- Space complexity $O(bd)$

5 Informed Search Strategies

Informed search algorithms apply domain knowledge in a problem to search the "most promising" branches first. This can lead to finding solutions more quickly, or finding solutions with lower costs than uninformed search algorithms.

5.1 Heuristics

The domain knowledge used in the problem is applied in the form of a heuristic function, which gives an estimate to the distance from the goal. Heuristics are often applied as a "commonsense" technique without many theoretical guarantees.

A heuristic function $h(n)$ can be used to estimate the "goodness" of node n. In general:

- $\forall n, h(n) \geq 0$
- $h(n) = 0$ means n is a goal node.
- $h(n) = \infty$ means n is a dead end that does not lead to a goal.

A heuristic function is considered admissible (or optimistic) if it never overestimates the cost of reaching the goal. Admissible heuristics can often be derived by relaxing the rules to a problem.

5.2 Strong vs Weak Methods

Strong methods are designed a address a particular type of problem, while weak methods are general approaches that can be applied to many types of problems. Because of the limited way in which domain-specific information is used to solve a problem with heuristic search, it is known as a weak method, since a strong method would apply more powerful domain-specific heuristics. Other examples of weak methods include:

- **Means-end analysis** - A strategy where a representation is formed for the current and goal state, and actions are analyzed that shrink the difference between the two.
- **Space splitting** - A strategy where possible solutions to a problem are listed, and then classes of these solutions are ruled out to shrink the search space.
- **Subgoaling** - A strategy where a large problem is split into independent smaller ones.

5.3 Best-first Search

Best-first search operates similarly to uniform cost search, but places nodes in a priority queue such that nodes in the queue are ordered by their values of $h(n)$.

5.3.1 Properties

- Not complete - can get stuck in loops
- Non-optimal
- Time complexity $O(b^m)$, but good heuristic can yield major improvements
- Space complexity $O(b^m)$

5.4 Beam Search

Beam search is an optimization of best-first search that reduces its memory requirement. Instead of expanding all nodes on a fringe, beam search only expands the β (beam width) with the lowest values of $h(n)$.

5.4.1 Properties

- Not complete
- Non-optimal
- Time complexity $O(\beta b)$
- Space complexity $O(\beta b)$
- Not admissible

5.5 A* Search

A function $g(n) = g(n) + h(n)$ can be used to give the total estimated cost of a solution, where $g(n)$ is the cost from the start to n, and $h(n)$ is the heuristic function for node n. However, an algorithm based on this is not complete if $h(n)$ can be infinite, and is not admissible.

A* search is a variation on the above algorithm with the constraint $h(n) \leq h^*(n)$, where $h^*(n)$ is the actual minimal path cost from n to a goal. This constraint makes A* search complete whenever the branching factor is finite and there is a fixed positive cost, and also makes it admissible.

Some properties of A* search include:

- **Perfect heuristic:** If $h(n) = h^*(n)$, then only nodes on the optimal solution are expanded.
- **Null heuristic:** If $h(n) = 0$, then A* acts like uniform cost search
- **Better heuristic:** If $h_1(n) < h_2(n) < h^*(n)$, then h_2 is a better heuristic than h_1.

5.6 Hill Climbing Search

Hill climbing search attempts to improve the efficiency of depth-first search. After starting with an arbitrary solution, hill climbing search attempts to improve the solution by changing a single element at a time. This is done by sorting the successors of a node according to their heuristic values, and then adding them to the list to be expanded. These changes are made until no further improvements can be found.

The rule applied in hill climbing search is that if there is a successor s for node n such that:

- $h(s) < h(n)$ and
- $h(s) \leq h(t)$ for all successors t of n.

then advance from n to s, otherwise halt at n.

Hill climbing search functions like a beam search with $\beta = 1$.

6 Game Playing as Search

Games involve playing against an opponent, where search problems involve finding a good move, waiting for an opponent's response, and then repeating. Time is typically limited in each search.

6.1 Types of Games

- **Perfect information:** Each player has complete information on the opponent's state and available choices (e.g chess).
- **Imperfect information:** Each player does not have complete information on the opponent's state and available choices (e.g. poker).

6.2 Max Min Strategy

In situations with perfect information and two players, a game tree can be expanded that describes all possible moves of the player and opponent in the game.

The minimax strategy is commonly used in zero sum games where the goal is to minimize the maximum losses that can occur. Given a game tree, an optimal strategy (assuming both players play optimally from each position) can be derived by examining the minimax strategy of each node. Each level in a game tree is assigned alternating MAX (player) and MIN (opponent) values, where MAX is the maximum value of its successors and MIN is the minimum value of its successors.

6.2.1 Limiting Depth

In practice, game trees can only be fully expanded for simple games. For more complicated games, a limited depth of the game tree should be explored. In such circumstances, an evaluation function $f(n)$ is used to measure the "goodness" of a game state.

6.2.2 Properties

- Complete if tree is finite
- Optimal (when playing against optimal opponent)
- Time complexity $O(b^d)$
- Space complexity $O(bd)$ (depth-first exploration)

6.3 α-β Pruning

Alpha-beta pruning seeks to reduce the number of nodes that need to be generated and evaluated. The key idea behind this algorithm is to avoid processing subtrees that cannot have an effect on the result. Two new parameters are introduced, namely α, which is the best value of MAX found so far, and β, which if the best value of MIN seen so far. Alpha is used in MIN nodes, and is assigned in MAX nodes, and vice versa for beta. The two introduced bounds are:

- **Alpha cutoff:** When value of min position is less than or equal to the alpha-value of its parent, stop generating further successors.
- **Beta cutoff:** When value of max position is greater than the beta-value of its parent, stop generating further successors.

The output of alpha-beta pruning is exactly the same as the output of the minimax algorithm; alpha-beta pruning is purely an efficiency consideration. The time complexity of the minimax algorithm is $O(b^d)$, while the time complexity of alpha-beta pruning is between $O(b^{\frac{d}{2}})$ and $O(b^d)$ depending on the properties of the search tree.

7 Metaheuristics

Metaheuristics are high-level heuristics designed to select other heuristics to solve a problem. Search algorithms making use of metaheuristics are approximate and usually non-deterministic, and vary from simple local search to complex learning processes.

7.1 Common Properties

- Mechanisms to avoid getting trapped in confined areas of search space.
- Not being problem-specific, though may make use of domain-specific knowledge from heuristics controlled by upper-level strategy.
- Search experience (i.e. some representation of search history) to guide the search.
- Hybrid search models where the search identifies neighborhoods where a goal may lie, and then the search is intensified in that area.

7.2 Population-based Methods

Population-based methods are metaheuristic approaches that apply multiple agents to a search space and can handle multiple simultaneous solutions.

7.3 Trajectory Methods

Trajectory methods are metaheuristic variants of local search that apply memory structure to avoid getting stuck at local minima, and implement an explorative strategy that tries to avoid revisiting nodes.

8 Tabu Search

Tabu Search (TS) is a widely used, general trajectory-based metaheuristic strategy for controlling inner heuristics. TS can be seen as a combination of a local search strategy with a search experience model that is used to escape local minima and implement an explorative strategy.

8.1 Local Search

Local search strategies start with an initial feasible solution, after which a series of local modifications are applied. If a modification generates a better solution, then the process is repeated from that node. Two major challenges faced by local search algorithms include:

- Can be costly to consider all possible local modifications.
- Can get stuck in local optimum.

8.2 Basic Ideas

TS attempts to overcome the limitations of local search alone. A key idea behind this is to penalize moves that take a solution back to a previously visited (tabu) state. This can result in non-improving solutions being accepted in order to escape from local optima and eventually find a better solution.

8.3 Use of Memory

TS applies "short-term" memory structures to store the recency of occurrence of a previous solution to. This is often known as a Tabu list, where the Tabu list length (or Tabu tenure) refers to the number of iterations for which moves are kept. Tabu lists rarely hold complete solutions due to their large memory requirements, and often instead store some related data (e.g. moves used to generate past solutions). When selecting a new neighborhood to explore, neighborhoods outside of the Tabu list are preferred.

TS can also apply "long-term" memory structures to store the frequency of solution components, which can also be used to diversify the search by exploring unvisited areas of the solution space.

8.4 Termination Conditions

Sample TS termination conditions include:

- No feasible solution in neighborhood of current solution.

- Maximum number of iterations reached.
- Number of iterations since last improvement has crossed an upper bound.
- Some sort of evidence shows that an optimal solution has been obtained.

8.5 Candidates and Aspiration

A candidate list stores the potential solutions in a neighborhood to be examined. This is done by isolating regions of a neighborhood with desirable features, aiming to find a solution more efficiently than by just searching all possible solutions in a neighborhood. At times, it can be desirable to include a move in a candidate list even if it is tabu in order to prevent stagnation. Approaches used to cancel tabus are referred to as aspiration criteria.

8.6 Selecting Tabu Restrictions

Examples of restrictions for picking the next move include:

- Not picking the same exchange of positions as in a tabu move.
- Not picking a move that results in a positions that previously appeared in a tabu move.

8.7 Selecting Tabu Tenure

Tabu tenure T can be selected in the following ways:

- Statically assigning T to be a constant (as a general rule, $\sqrt{n}$ where n is the problem size.
- Dynamically letting T vary between a T_{min} and T_{max}. Dynamic Tabu tenures tend to be better at limiting cycling, though some searches may contain cycles longer than the Tabu tenure.

8.8 Selecting Aspiration Criteria

Aspiration criteria can be selected based on the following strategies:

- By default, where a tabu move becomes admissible if it yields a better solution than any found so far.
- By objective, where a tabu move becomes admissible if it yield a solution better than an aspiration value.
- By search direction, where a tabu move becomes admissible if the direction of the search remains constant (e.g. non-improving remains non-improving).

8.9 Intensification

Intensification refers to the process of exploiting a small portion of the search space (e.g. penalizing solutions far from current solution). Intensification can be used to locally optimize a best known solution while trying to preserve the general components of that solution (based on short-term memory).

8.10 Diversification

Diversification refers to the process of forcing the search into unexplored areas (e.g. penalizing solutions close to current solution). While Tabu lists can aid in diversification, they often act very locally, and long-term memory known as the "frequency memory" is used to store the number of iterations in which certain components always appear in solutions. Diversification strategies can include:

- **Restart diversification**, where components rarely appearing in solutions are forced into new solutions.
- **Continuous diversification**, where the evaluation of possible moves is biased by a term related to component frequency.

8.11 Adaptation

Adaptation refers to a series of techniques for varying the Tabu tenure. As a general rule, if the Tabu tenure is too small, then cycles in the search are likely, while if it is too big, then many moves could be prevented at each iteration.

Several difference techniques can be applied to vary to Tabu tenure. One such technique involves randomly selecting a new tenure from a pre-computed range every predetermined number of iterations. Another approach involves setting to tenure to 1 if a best-so-far solution is found, lowering it in an improving phase, and increasing it in a worsening phase. This can be coupled by randomly changing the min and max values of the tenures for each fixed number of moves.

8.12 Cooperation

Cooperation can be applied by multiple Tabu searches running concurrently. One such approach involves coordinated searches exchanging information every fixed number of iterations (synchronous communication), where an incoming solution to a search agent can be handled by replacing its own best-so-far (forced diversification), or by replacing its own best-so-far solution only if the incoming solution is better (conditional import). Alternatively, asynchronous communication can be applied where search agents relay their best-so-far results to a central memory. If this best-so-far result if better than the global one, then it replaces

the global one. Otherwise, search agents continue using their worse one for a certain number of moves, after which they fetch the solution stored in the central memory. Some variants store multiple solutions in central memory, where a random one is assigned to agents requesting a solution.

General results from cooperation experiments found that:

- Increasing the number of search agents improves the solution up to a certain point.
- Increasing the number of synchronization messages increases the computation time due to message passing overhead.
- Conditional imports are almost always preferable to forced diversification.

9 Simulated Annealing

9.1 Physical Annealing Analogy

Physical annealing involves heating a substance (e.g. a metal) and then letting it cool to increase its ductility and reduce hardness. The goal is to make the molecules in a cooled substances arrange themselves in a low-energy structure, and the properties of this structure are influenced by the temperatures reached and the rate of cooling. A sequence of cooling times and temperatures is referred to as an annealing or cooling schedule.

Simulated annealing is a search algorithm that mimics the physical annealing process. First introduced in 1953, simulated annealing leads to a random walk between solutions that, under certain conditions, is guaranteed to sample from the probability distributions of an ideal equilibrium state. Further contributions were made in 1983 and 1985 that applied simulated annealing to finding approximations of global optima in large search spaces.

9.2 Basics of Simulated Annealing

Simulated annealing search algorithms start with some arbitrary initial solution, "temperature", and temperature reduction function. Then until some sort of stopping condition is satisfied, the algorithm repeatedly picks new potential solutions from a neighborhood $N(s)$. If the solution has a lower cost than the previously found one, then it is accepted. Otherwise, the solution is accepted only if a random variable x in the range $(0, 1)$ is lesser than $e^{-\frac{\Delta c}{t}}$, where Δc is the increase in cost and t is the temperature. After each step, the temperature reduction function is applied.

9.3 Strategy

The general strategy of simulated annealing is to encourage exploring more of the search space at the beginning when the temperature is high. Later on, once the algorithm converges on an approximation of the global optimum, accepting worse solutions to facilitate exploration becomes less probable. Lower temperatures as well as higher changes in cost lower the probability of a particular change being accepted as a new solution.

9.4 Annealing Schedule

An annealing schedule provides a mapping from time to temperature, and is determined by a search's initial temperature, final temperature, rules for decrementing temperatures, and the number of search iterations at each temperature.

9.4.1 Initial Temperature

The initial temperature should be high enough to allow exploration to any part of the search space. However, making the initial temperature too hot will make the search behave almost completely randomly. The maximum change of a cost function should be considered when setting the initial temperature, which as a general rule, should accept around 60% of worse solutions.

9.4.2 Final Temperature

A final temperature should be quite low but does not necessarily have to reach 0. A search using simulated annealing can be stopped once no better moves are being found and no worse moves are being accepted.

9.4.3 Temperature Decrement Rules

Two basic techniques for decrementing temperature are though a linear ($t = t - \alpha$) and a geometric ($t = t \cdot \alpha$) function. For geometric decrement, the recommended value of α is between 0.8 and 0.99. Alternatively, an technique that facilitates slow decrease ($t = \frac{t}{1-\beta t}$) can be applied.

9.4.4 Temperature Iterations

Ideally, enough iterations should be performed for a search to become stable at each tested temperature. Large population sizes are a common reason for increasing this number of iterations, while slowly decreasing temperatures are a common reason for decreasing the number of iterations. A constant value of iterations is commonly used, though it is common to increase the number of iterations as the search increases to fully explore local optima.

9.5 Convergence of Simulated Annealing

Simulated annealing is guaranteed to eventually converge to a solution at a constant temperature, assuming some sequence of moves leads to the goal state. This becomes much more nuanced when temperature is not constant, where convergence can still be guaranteed but only under conditions that result in very slow temperature reduction and an exponential increase in the number of iterations at each temperature. Hence, this convergence theory is not commonly used in practice, though it is worth noting that simulated annealing is better backed theoretically than many other heuristic methods.

9.6 Adaptation

Adaptation in simulated annealing refers to adapting the critical parameters of the algorithm, including the initial temperature, cooling schedule, and number of iterations.

9.6.1 Initial Temperature

Finding the right temperature is very problem-specific, and different search algorithms can be applied in finding this temperature.

9.6.2 Cooling Schedule

Some cooling schedules require that only the cooling rate α is specified, and the remainder of parameters are automatically determined using a linear random combination of previously accepted states and parameters to estimate new steps and parameters.

9.6.3 Probability of Acceptance

Some attempted adaptations concerning the probability of acceptance include using a lookup table for the relevant calculations (to decrease computation time) or using a different, non-exponential probability formula.

9.6.4 Cost Function

Cost functions that return similar values for many different states tend to not lead to effective search. As an alternative, a cost function can have a penalty term associated with certain types of states, and the weighting of these penalty terms can vary dynamically.

9.7 Cooperation

Cooperative simulated annealing involves multiple concurrent runs of a simulated annealing search algorithm on a search space. Potential solutions are produced by somehow combining the value of a run with the value of a random previous run of the algorithm. This exchange of information from other solutions is known as cooperative transition, and is a concept borrowed from genetic algorithms.

9.8 Advantages and Disadvantages

Some advantages of simulated annealing include that it provides good solutions to a wide range of problems, and is fairly simple to understand and apply. Some disadvantages include

that simulated annealing often takes many steps to reach a good enough solution, and has many tunable parameters.

10 Evolutionary Algorithms

Evolutionary computing forms a category of population-based meta-heuristic methods whose behavior is inspired by biological evolution. These algorithms feature a population of individuals competing for limited resources, with the "fitter" individuals being used as seeds to form future generations. Over time, this population rises in overall fitness due to the principles of natural selection. Evolutionary algorithms are stochastic, with variation operators (crossover and mutation) propagating changes to new generations.

10.1 Active Information

Unless a search algorithm takes advantage of problem-specific information about a search target of search space, its average performance is that of random search. Three measures of information that can increase the effectiveness of a search can be categorized as:

- **Endogeneous information**, which measures the difficulty of finding a target through random search.
- **Exogeneous information**, which measures the difficulty of finding a target once problem-specific information is applied.
- **Active information**, which is the difference between exogenous and endogeneous information (i.e. measures the contribution of problem-specific information in solving a problem).

11 Genetic Algorithms

Genetic algorithms are a class of evolutionary algorithms that operate by maintaining a population of candidate solutions and iteratively applying a set of stochastic operators, namely selection, reproduction, and mutation. These algorithms are inspired by Darwin's theory of natural selection, with the population eventually moving towards fitter solutions.

11.1 Simple Genetic Algorithms

The idea of genetic algorithms was first introduced by John Henry Hollard in 1975. The first described genetic algorithms are now known as simple genetic algorithms (SGAs), also known as classical or canonical SGAs.

SGAs start by initializing a population with random candidate solutions. Then, parents are selected, after which a new generation is formed through the genetic operations of recombination and mutation. This generation replaces the parent generation, and the breeding cycle repeats until some termination criteria is reached. Example termination criteria include the search going passed a certain number of iterations, no improvement in the solution for a certain number of iterations, or the fitness of the solution passing a certain threshold.

11.1.1 Representation

SGAs represent candidate solutions as binary strings, which are often referred to as chromosomes of genotypes. Each binary digit is considered to be a separate gene, and the ordering of the genes can have important implications. This, the mapping from a problem's parameter domain to this binary representation is one of the most important factors for the performance of an SGA algorithm.

Binary representations of a problem often apply gray coding, which means that small differences in the underlying solution result in small changes in the binary representation. Some implementations use representations other than binary strings, including integers or floating-point numbers.

11.1.2 Selection

Individuals are selected to be parents in the future generation with a probability proportional to their fitness (i.e. proportional selection). Based on this principle, N parents can be selected from a population of N solutions. The key idea behind this selection is to facilitate an increase in the fitness of future generations while also maintaining an element of randomness.

A major limitation of fitness-proportional selection (FPS) is that it does not provide guarantees on the distribution of selected parents, since each selection is done independently. An alternative is stochastic universal sampling, which can be imagined as spinning a roulette wheel with n evenly spaced arms to find n values.

Some other common problems with FPS algorithms are premature convergence (e.g. one highly fit member dominating a population), and a lack of selection pressure at the end of runs with similar fitness. Some attempts to overcome these problems include selection proportional to rank rather than absolute fitness, and tournament selection where random subsets of individuals are found, and the fittest members of these subsets are selected.

11.1.3 Crossover

Crossover/recombination is applied between two parents with a probability of P_c, which is typically in the range (0.6, 0.9). If no crossover takes place, the two parents are copied to two offspring unmodified. The following types of crossovers can be applied:

- **1-point crossover**, where a random point is chosen on the two parents, and the two children are formed by exchanging the tails this point divides.
- **n-point crossover**, which is a generalization of 1-point crossovers where n points are chosen on the two parents.
- **Uniform crossover**, where each gene has an independent $\frac{1}{2}$ chance of undergoing recombination, which makes inheritance independent of position.

11.1.4 Mutation

After recombination, each gene can be altered with a probability of p_m. p_m is typically between $\frac{1}{popsize}$ and $\frac{1}{chromosomelength}$. Crossovers tends to result in large changes in a population, while mutation results in small ones. Thus, crossovers are considered to be explorative, making a big jump to a possibly unexplored area between the two parent solutions, while mutations are exploitative, introducing small amounts of new information to further explore near an existing solution.

11.1.5 Population Models

SGA typically applies a generational GA model (GGA), where individuals survive for exactly one generation before they are replaced by offspring. An alternative is a steady-state GA model (SSGA), where part of a population is replaced by offspring. The proportion of the population replaced between successive generations is known as a generational gap (1 for GGA, and typically $\frac{1}{popsize}$ for SSGA).

The process of selecting individuals from parents and offspring to make up the next generation is known as survivor selection. Common approaches for survivor selection include age-based (e.g. delete oldest) and fitness-based (e.g. keep best, delete worst) selection.

11.2 Real-Valued Genetic Algorithms

It is impractical to map many problems with real-values parameters to binary representations. Although real-valued GAs are similar to the SGAs discussed above they can differ as described below:

11.2.1 Crossover

It is possible to still apply discrete crossover techniques as done for binary chromosomes. However, real-valued chromosomes open up the possibility for the following intermediate crossovers:

- **Single arithmetic crossover**, where for a single parent gene pair x and y, one child's gene becomes $\alpha x + (1 - \alpha)y$, and the reverse for the other child.
- **Simple arithmetic crossover**, where for each parent gene pair x and y, after a certain gene, one child's gene becomes $\alpha x + (1 - \alpha)y$, and the reverse for the other child.
- **Whole arithmetic crossover**, where for each parent gene pair x and y, one child's gene becomes $\alpha x + (1 - \alpha)y$, and the reverse for the other child.

11.2.2 Mutation

With real-valued chromosomes, mutations can be applied by assigning a uniform random value between some lower and upper bound to a gene. Some variants on this technique also add some noise (e.g. from a Gaussian distribution) to this number.

11.3 Permutation Genetic Algorithms

Permutation problems are problems that deal with finding a sequence in which elements are arranged. There are two main classes of such problems:

- Problems that deal with adjacency of elements (e.g. Travelling Salesman Problem).
- Problems that deal with the overall order of elements (e.g. Job Shop Scheduling).

11.3.1 Crossover

Previously defined crossover operations often result in inadmissible solutions (e.g. crossing [1,2,3] with [3,2,1] can result in duplicates of one element in an arrangement).

Below are some crossover operations than can be applied to adjacency-based problems:

- **Partially Mapped Crossover (PMX)**, which follows the approach below to generate a child from parents P1 and P2:
  1. Copy a random segment from P1.
  2. Starting from the first crossover point, look for elements in P2 that have not been copied.
  3. For each of these elements i, look in the offspring to see which element j has been copied in its place.
  4. Place i in the position occupied by j in P2. If the place occupied by j in P2 has already been filled in by k put i in the position occupied by k in P2.
  5. The rest of the offspring can be filled in from P2.
- **Edge crossovers**

Below are some crossover operations than can be applied to order-based problems:

- **Order 1 Crossover**, which follows the approach below to generate children:
  1. Choose arbitrary part from P1.
  2. Copy this part to first child.
  3. Starting from the right of the cut point of the copied part, copy the elements in the order of P2 that are not yet in the child, wrapping around if needed.
- **Cycle Crossover**, which follows the approach below to generate children:
  1. Form a cycle of alleles from P1 by:
     (a) Start with the first allele of P1.
     (b) Go to the position in P1 that has the value of the corresponding allele in P2.
     (c) Add this allele to the cycle.
     (d) Repeat this cycle formation until the first allele of P1 is arrived at.
  2. Put alleles of the cycle in the in the positions from P1.
  3. Repeat steps above for second parent.

11.3.2 Mutation

Some previously defined mutation operators may lead to inadmissible solutions for permutation problems. Below are some mutations that can be applied:

- **Insert mutations**, which pick two elements, and move one to immediately follow the other, shifting any elements if necessary (i.e. as in insertion sort)
- **Swap mutations**, which pick two elements and swap their order.

- **Inversion mutations**, which pick two genes and reverse the subsequence between them.
- **Scramble mutations**, which pick two genes and find a random permutation of the genes between them.

11.4 Adaptation

11.4.1 Parameter Tuning

Much of the work in applying a genetic algorithm to solve a problem often lies in parameter tuning, which refers to finding suitable values for different algorithm parameters. However, parameter tuning is often a lengthy process, since parameters are often not independent from one another, and different parameter values may be optimal at different stages of the search process.

11.4.2 Parameter Control

Various parameter control techniques can be applied to set the values of GA algorithm parameters during runtime. Although not necessarily optimal, these techniques tend to be far more efficient than parameter tuning and can be applied to a wide range of problems. The main types of parameter control techniques are:

- **Deterministic:** Parameters are controlled as a function of the generation number; search progress is not taken into account.
- **Adaptive:** Parameters are controlled using the current state of the search combined with some heuristics.
- **Self-adaptive:** Parameters are incorporated into chromosomes and are controlled through similar processes as the problem population (i.e. selection, crossover and mutation).

11.5 Parallel Genetic Algorithms

11.5.1 Master-Slave Approach

In master-slave approaches, selection and mating are performed by a single master processor. However, fitness evaluation is distributed among several slave processors. Master-slave GAs are also known as global parallel GAs.

11.5.2 Fine-Grained Genetic Algorithms

In fine-grained GAs, selection and mating occurs in local (not necessarily disjunct) neighborhoods. This technique can be effective for handling computations on massively parallel computers.

11.5.3 Coarse-Grained Genetic Algorithms

Coarse-grained genetic algorithms feature multiple populations evolving in parallel. Selection and mating is limited to individuals within the same population, and different populations may periodically exchange individuals. These are also known as distributed or multi-deme GAs, and depend on many factors including:

- **Topology**, which controls which populations are connected to one another. Individuals may only migrate between connected populations.
- **Migration policy**, which determines how migrating individuals are sent and received. Sample policies include exchanging a random individual in one population for a random individual in another population, or exchanging the best individual in one generation for the worst individual in another.
- **Migration frequency**, which controls when communication between populations occurs. This communication may be synchronous, where migrations occur every predetermined number of generations, or asynchronous, where migrations are triggered by a certain event.
- **Migration rate**, which refers to the number of individuals migrating from one population to another at every communication step. A low number of migrating individuals may lead to the populations acting almost completely independently, while a high number of migrating individuals may lead to fast convergence to sub-optimal solutions.

11.5.4 Cooperative Genetic Algorithms

Cooperative GAs tend to contain multiple populations where the fitness of an individual in one population depends on the fitness of individuals in other population. A commonly applied strategy is to have different populations optimize different variables of a problem, where the fitness of an individual is determined by its value and the value of the best individuals in other populations. This strategy tends to perform best when the different variables are independent of one another.

11.6 Advantages and Disadvantages

A few advantages of GAs include that they are highly multimodal, can be applied to discrete and continuous problems, can support high-dimensional problems, and support non-linear

dependencies between parameters. A few disadvantages of GAs include premature conversion, poor choice of genetic operators, and biased or incomplete representations.

12 Swarm Intelligence

Swarm intelligence (SI) refers to the collective behavior of decentralized, self-organized systems. As a computational phenomena, it was largely inspired by the collective behavior of certain social animals, such as ant colonies and bird flocking.

12.1 Interactions

A swarm refers to a group of agents that communicate with each other by acting on their local environment. Complex problem-solving behavior may emerge not as a result of any particular individual, but rather as a result of their interactions. Interactions between individuals may be direct (e.g. by physical contact) or indirect (e.g. via local changes to an environment to be picked up later by that individual or by other individuals, often referred to as stigmergy).

Interactions among individuals resulting in emergent behavior have been observed in various non-biological systems as well. Examples include stock markets, traffic patterns, and the spatial structure of galaxies.

12.2 Properites

Below are common properties of systems based on swarm intelligence:

- **Flexibility:** System performance is adaptive to internal or external changes.
- **Robustness:** System can perform even if some individuals fail.
- **Decentralization:** Control is distributed among individuals rather than allocated to some master.
- **Self-organization:** Global behaviors emerge as a result of local interactions.

12.3 Models of behavior

Common behavioral models include:

- **Swarm:** Group with little parallel alignment.
- **Torus:** Group in which individuals rotate around an empty core in one direction.
- **Dynamic parallel group:** Group where individuals are polarized and move as a coherent group, but can still move throughout the group such that the group density and form fluctuates.
- **Highly parallel group:** Group similar to dynamical parallel group but with minimal fluctuations.

12.4 Swarm Intelligence Problem Solving

Applying SI techniques to solving a problem involves modeling an agent/individual, modeling the interaction process, and possibly adding adaptive or cooperative properties. General principles to keep in mind when applying SI techniques include:

- **Proximity:** The swarm should be able to carry out simple space and time computations.
- **Quality:** The swarm should be able to respond to quality factors in its environment.
- **Diverse response:** The swarm should not commit to excessively narrow channels of exploration.
- **Stability:** The swarm should not rapidly alter its behavior in response to all environmental changes.
- **Adaptability:** The swarm should be willing to change its behavior when it is worth the computational price.

13 Ant Colony Optimization

13.1 Background

Ant colony optimization (ACO) is a search technique based on the swarm intelligence of ants. Ants interact with one another through stigmergy, meaning that individual ants can make changes to their environment to be picked up by other ants.

Ants do this by forming trails with substances known as pheromone. Ants tend to follow paths with higher pheromone concentrations. Pheromone also evaporates over time, so more recent pheromone deposits have a greater influence than older ones.

Ants can use this mechanism to find the shortest path to a destination. If multiple paths are available, ants will pick their initial paths randomly. Ants that take the shorter path will return faster, so their trail will have more unevaporated pheromone. This makes other ants more likely to pick this trail, and eventually, they converge on this shortest path.

13.2 Basic Algorithm

Let $G = (N, E)$ denote a graph where N is a set of nodes and E is a set of edges. Each edge (i, j) has a length denoted as $d_{i,j}$, and a value $\tau_{i,j}$ that denotes the amount of pheromone.

13.2.1 Initialization

Initially, a small amount of pheromone is placed on all edges, and a group of m ants is placed at a source node.

13.2.2 Transition Rule

At each node i, with adjacent nodes N_i, an ant can move to an adjacent node j with the following probability:

$$\frac{\frac{\tau_{i,j}^{\alpha}}{d_{i,j}^{\beta}}}{\sum_{n \in N_i} \frac{\tau_{i,n}^{\alpha}}{d_{i,n}^{\beta}}}$$

α and β balance the local and global search ability respectively. More generally, a problem heuristic function can be used instead of the inverse of distance.

13.2.3 Pheromone Evaporation and Update

In each step the pheromone trail is evaporated (i.e. $\tau = \tau \cdot (1 - \rho)$). Ants also increase the amount of pheromone on the trail they choose (i.e. $\tau = \tau + \Delta\tau$), a process known as

step-by-step pheromone update. $\Delta\tau$ can be chosen in various ways, including:

- **Ant density model**, where a constant value is added.
- **Ant quality model**, where a constant is divided by the edge length.
- **Online delayed model**, where an ant first builds a solution, then traces its path backwards and adds pheromone based on the solution quality. This is applied in extensions of ACO known an ant system algorithms (AS).

13.2.4 Termination Criteria

The above step of evaporating and then adding pheromones is repeated until termination conditions are met. Common termination conditions include:

- Maximum number of iterations reached.
- Good enough solution reached.
- Stagnation occurs (i.e. almost all ants follow same path).

13.2.5 Tunable Parameters

Tunable parameters for ACO include:

- Number of ants
- Max number of iterations
- Initial pheromone
- Pheromone delay parameters (i.e. ρ)

13.3 Extensions

13.3.1 Ant Colony System Algorithm

The ant colony system (ACS) algorithm features the following extensions on ACO:

- The transition rules sometimes just chooses the best path (i.e. acts greedily) instead of applying probabilistic selection.
- Pheromone update is only based on the best solution (i.e. highest increase in τ either globally or in iteration).

13.3.2 Max-Min Ant System Algorithm

This max-min ant system algorithm (MMAS) is an extension that restricts pheromone values within a range. This extension mitigates stagnation, since the max and min pheromone values can be adjusted to favor exploration over exploitation during early phases of the search.

13.4 Characteristics of Problems

ACO is often a good fit for combinatorial optimization problems. ACO is typically applied to discrete problems, though there are variants for handling continuous optimization problems.

13.5 Adaptation

13.5.1 ACSGA-TSP

The idea behind the ACSGA-TSP adaptation approach is to have a genetic algorithm running on top of ACS to attempt to optimize its parameter values. Each ant has certain parameters (e.g. ρ, β) encoded into a chromosome. New generations are formed by crossing over the best-performing ants.

13.5.2 Near Parameter Free ACS

The idea behind near parameter free ACS is to apply an ant approach to optimize ant parameters.

13.6 Cooperation

Heterogeneous cooperation approaches involve ants in different colonies having different behavior. This can be used, for instance, to optimize for different criteria of a solution.

Homogeneous cooperation approaches involve ants in different colonies having similar behavior. Exchange of information can take place in a similar way to that of genetic algorithms. Homogeneous cooperation implementations can be course-grained, where each process holds a single ant, or fine-grained, where each process holds a colony. An effective version of homogeneous cooperation is a circular exchange of locally best solutions.

13.7 Advantages and Disadvantages

A few advantages of ACO are:

- Memory of entire colony retained (not just previous generation).
- Poor solutions rarely converged on due to many combinations of path selection.
- Effective handling of dynamic environments.

A few advantages are:

- Theoretical analysis is limited.
- Many parameters to tune.
- Convergence may take long.