Introduction to
Optimization Method

Dis·$count$

School of Management
University of Science and Technology of China

Sept 5, 2020

Table of contents

1 Dynamic Programming

2 Integer & Linear Programming

3 General Optimization Methods for MILP
- Exact Methods
- Approximate Methods

4 Combinatorial Optimization

5 General Optimization problem

Dynamic Programming

0-1 Knapsack Problem

The most common problem being solved is the 0-1 knapsack problem, which restricts the number x_i of copies of each kind of item to zero or one. Given a set of n items numbered from 1 up to n, each with a weight w_i and a value v_i, along with a maximum weight capacity W,

$$\max \sum_{i=1}^{n} v_i x_i$$

$$\text{s.t.} \sum_{i=1}^{n} w_i x_i \leq W \text{ and } x_i \in \{0, 1\}.$$

0-1 Knapsack Problem

Assume $w_1, w_2, \ldots, w_n, W$ are strictly positive integers. Define $m[i, w]$ to be the maximum value that can be attained with weight less than or equal to w using items up to i. We can define $m[i, w]$ recursively as follows:

$$m[0, w] = 0$$
$$m[i, w] = m[i-1, w] \text{ if } w_i > w$$
$$m[i, w] = \max(m[i-1, w], m[i-1, w-w_i] + v_i) \text{ if } w_i \leq w.$$

0-1 Knapsack Problem

Assume $w_1, w_2, \ldots, w_n, W$ are strictly positive integers. Define $m[i, w]$ to be the maximum value that can be attained with weight less than or equal to w using items up to i. We can define $m[i, w]$ recursively as follows:

$$m[0,\, w] = 0$$
$$m[i,\, w] = m[i-1,\, w] \text{ if } w_i > w$$
$$m[i,\, w] = \max(m[i-1,\, w],\, m[i-1, w-w_i] + v_i) \text{ if } w_i \leq w.$$

0-1 Knapsack Problem

The solution can then be found by calculating $m[n, W]$. To do this efficiently, we can use a table to store previous computations. This solution will therefore run in $O(nW)$ time and $O(nW)$ space.

Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1$ $v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2$ $v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5$ $v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6$ $v_4 = 22$	0	1	6	7	7	18	22	24	28	29	29	40
$w_5 = 7$ $v_5 = 28$	0											

Optimal substructure

- Fibonacci sequence

$$fib(n) = fib(n-1) + fib(n-2)$$

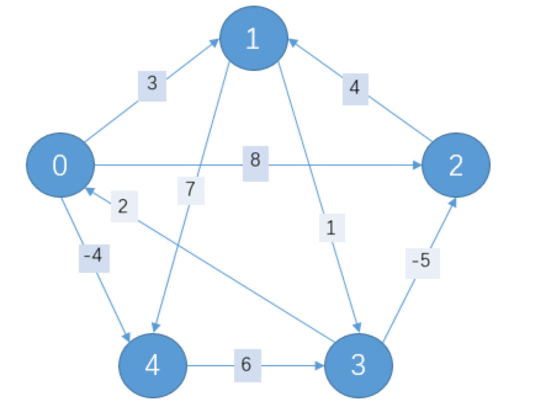
- Dijkstra's algorithm for the shortest path problem

$$d[y] = \min_{x}\{d[y], d[x] + w(x, y)\}$$

How to define the status and stage of problems is essential.

Shortest path problem

Given a directed graph (V, A) with source node s, target node t, and cost w_{ij} for each edge (i, j) in A, consider the program with variables x_{ij}.



Shortest path problem

- Integer programming formulation:

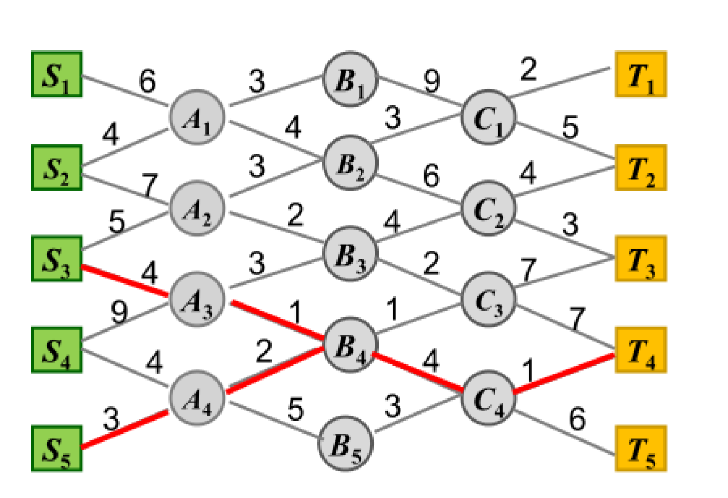
$$\min \sum_{(i,j)\in A} w_{ij}x_{ij}$$

$$\text{s.t.} \sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 1, & \text{if } i = s; \\ -1, & \text{if } i = t; \\ 0, & \text{otherwise.} \end{cases}$$

$$x \in \{0,1\} \text{ and for all } i.$$

Shortest path problem

- Find the shortest path from s to t.



Shortest path problem

$$Stage1 \quad F(C_l) = \min_m \{C_l T_m\}$$

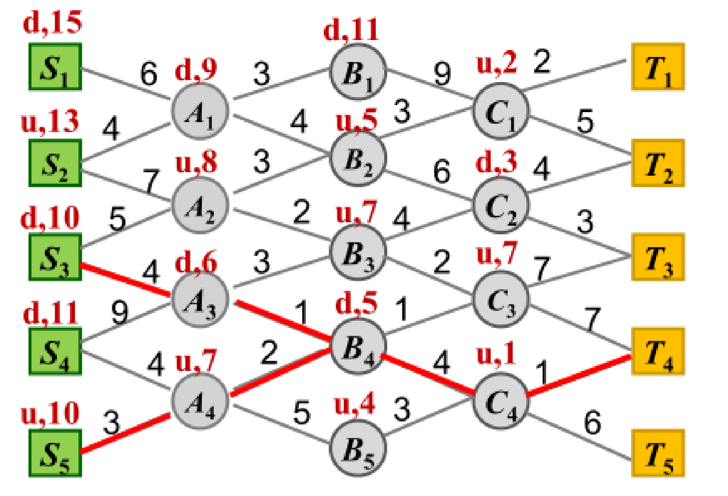
$$Stage2 \quad F(B_k) = \min_l \{B_k C_l + F(C_l)\}$$

$$Stage3 \quad F(A_j) = \min_k \{A_j B_k + F(B_k)\}$$

$$Stage4 \quad F(S_i) = \min_j \{S_i A_j + F(A_j)\}$$

Shortest path problem

- Find the shortest path from s to t.



Summary

- Optimal substructure.
- DP vs Recursion.
- Multi-stage.

Integer & Linear Programming

Integer Programming

- Shortest path problem

$$\min \sum_{(i,j)\in A} w_{ij}x_{ij}$$

$$\text{s.t.} \sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 1, & \text{if } i = s; \\ -1, & \text{if } i = t; \\ 0, & \text{otherwise.} \end{cases}$$

$$x \in \{0, 1\} \text{ and for all } i.$$

- Maximum flow problem
- Assignment problem

Integer Programming

- Shortest path problem

$$\min \sum_{(i,j)\in A} w_{ij}x_{ij}$$

$$\text{s.t.} \sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 1, & \text{if } i = s; \\ -1, & \text{if } i = t; \\ 0, & \text{otherwise.} \end{cases}$$

$$x \in \{0,1\} \text{ and for all } i.$$

- Maximum flow problem
- Assignment problem

Integer Programming

- Shortest path problem

$$\min \sum_{(i,j)\in A} w_{ij} x_{ij}$$

$$\text{s.t.} \sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 1, & \text{if } i = s; \\ -1, & \text{if } i = t; \\ 0, & \text{otherwise.} \end{cases}$$

$$x \in \{0, 1\} \text{ and for all } i.$$

- Maximum flow problem
- Assignment problem

Totally Unimodular Matrix

- Every entry in A is 0, +1, or -1;
- Every column of A contains at most two non-zero (i.e., +1 or -1) entries;
- If two non-zero entries in a column of A have the same sign, then the row of one is in B, and the other in C;
- If two non-zero entries in a column of A have opposite signs, then the rows of both are in B, or both in C.

TU Matrix

- Totally unimodular matrices are extremely important in combinatorial optimization since they give a quick way to verify that a linear program is integral (has an integral optimum, when any optimum exists).
- Specifically, if A is TU and b is integral, then linear programs of forms like $\{\min cx \mid Ax \geq b, x \geq 0\}$ or $\{\max cx \mid Ax \leq b\}$ have integral optimum, for any c. Hence if A is totally unimodular and b is integral, every extreme point of the feasible region (e.g. $\{x \mid Ax \geq b\}$) is integral and thus the feasible region is an integral polyhedron.

Another Perspective

Recall the simplex method for linear programming.

$$Bx = b$$

Cramer's rule:

$$x^* = (B^{-1}b, 0)$$

How to obtain the inverse of B?

$$B^{-1} = B^*/\mathsf{det}(B)$$

Simplex Method

- Feasible region(Convex polytope)
- Basic feasible solution(Extreme point)
- Basic variables(Identity matrix)
- Entering variable selection
- Leaving variable selection
- Pivot operation
- Reduced costs

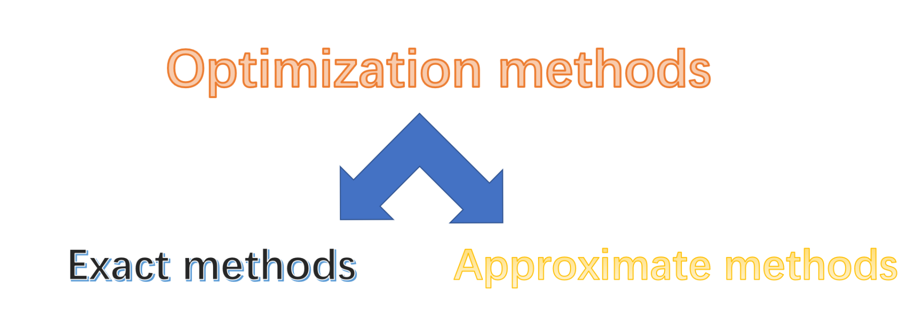
Another Perspective

The simplex method is an iteration process.

$$x' = x - \theta\lambda$$

- λ: Gradient direction(As small as possible)
  Entering variable selection
- θ: Step length(As long as possible)
  Leaving variable selection

General Optimization Methods for MILP



General Optimization Methods for MILP
Exact Methods

Exact Methods

- Branch and X

1. Branch and bound
2. Branch and cut
3. Branch and price

Exact Methods

- Branch and X
  1. Branch and bound
  2. Branch and cut
  3. Branch and price

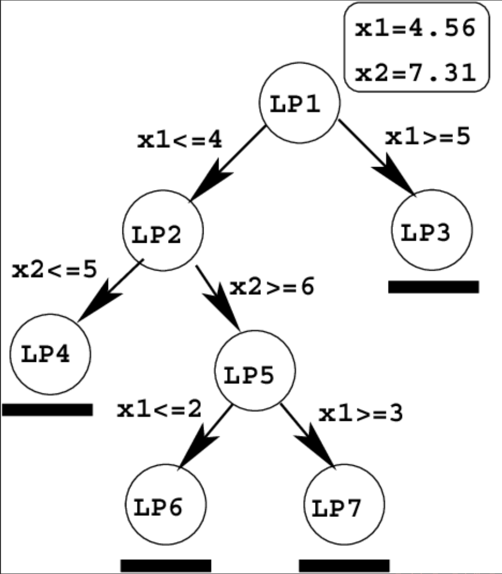
Exact Methods

- Branch and X
    1. Branch and bound
    2. Branch and cut
    3. Branch and price

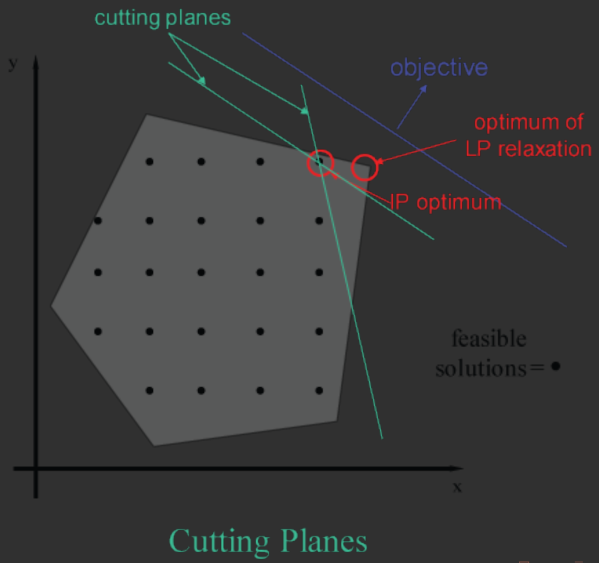
Exact Methods

- Branch and X
  1. Branch and bound
  2. Branch and cut
  3. Branch and price

Branch and bound



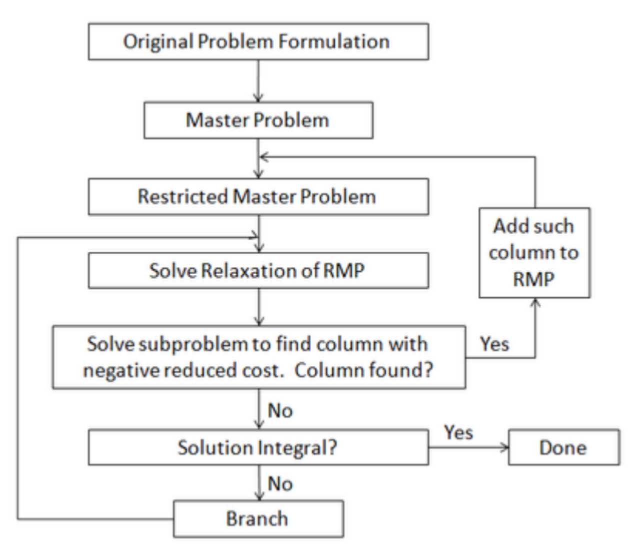
Cutting Plane Method



Gomory's Cut

- Using the simplex method: $x_i + \sum \bar{a}_{i,j} x_j = \bar{b}_i$
  where x_i are the basic variables and the x_j are the nonbasic variables.
- Rewrite this equation: integer parts(left) and the fractional parts(right):
  $x_i + \sum \lfloor \bar{a}_{i,j} \rfloor x_j - \lfloor \bar{b}_i \rfloor = \bar{b}_i - \lfloor \bar{b}_i \rfloor - \sum (\bar{a}_{i,j} - \lfloor \bar{a}_{i,j} \rfloor) x_j.$
- Right side is less than 1 and the left side is an integer, therefore 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.
- The inequality above 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
  $x_k + \sum (\lfloor \bar{a}_{i,j} \rfloor - \bar{a}_{i,j}) x_j = \lfloor \bar{b}_i \rfloor - \bar{b}_i$, $x_k \geq 0$, x_k an integer.

Branch and price



Exact Methods

- Branch and X
- Dynamic programming
- Constraint programming
- Enumeration method

Exact Methods

- Branch and X
- Dynamic programming
- Constraint programming
- Enumeration method

Exact Methods

- Branch and X
- Dynamic programming
- Constraint programming
- Enumeration method

General Optimization Methods for MILP
Approximate Methods

Approximate Methods

- Heuristic algorithms
  1. Problem-specific heuristics
  2. Metaheuristic
     - Single solution-based metaheuristics
     - Population-based metaheuristics
- Approximate algorithms

Approximate Methods

- Heuristic algorithms
  1. Problem-specific heuristics
  2. Metaheuristic
     * Single solution-based metaheuristics
     * Population-based metaheuristics
- Approximate algorithms

Approximate Methods

- Heuristic algorithms
  1. Problem-specific heuristics
  2. Metaheuristic
     * Single solution-based metaheuristics
     * Population-based metaheuristics
- Approximate algorithms

Approximate Methods

- Heuristic algorithms
  1. Problem-specific heuristics
  2. Metaheuristic
     * Single solution-based metaheuristics
     * Population-based metaheuristics
- Approximate algorithms

Approximate Methods

- Heuristic algorithms
  1. Problem-specific heuristics
  2. Metaheuristic
     * Single solution-based metaheuristics
     * Population-based metaheuristics
- Approximate algorithms

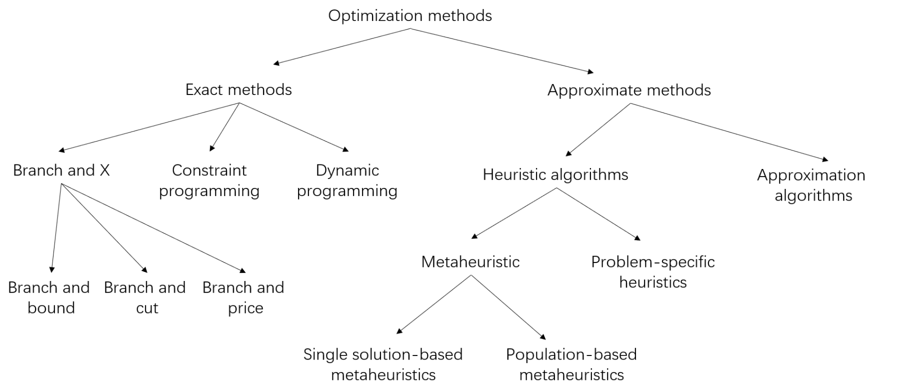
Single solution-based metaheuristics

- Simulated Annealing Algorithm
- Tabu Search
- Variable Neighborhood Search
- Adaptive Large Neighborhood Search

Population-based heuristics

- Genetic Algorithm
- Ant Colony Optimization
- Partical Swarm Optimization

General Optimization Methods for MILP



Combinatorial Optimization

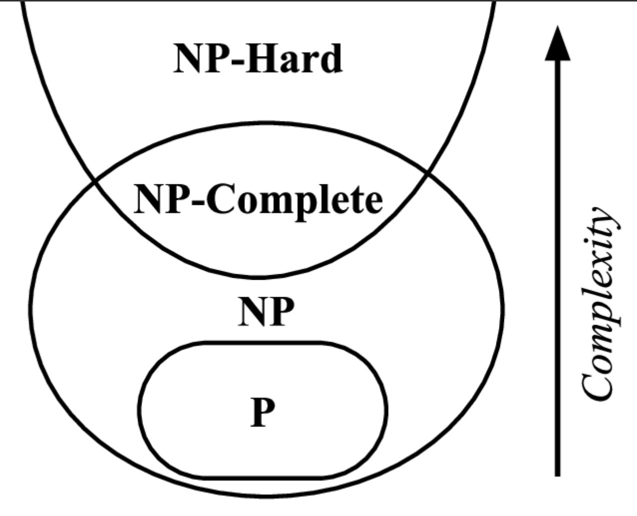
Classical problems

- Knapsack problem
- Traveling salesman problem
- Set covering problem
- Matching problem
- Vehicle routing problem
- Facility location problem
- Production scheduling problem

NP-complete

- P(PTIME)
  It contains all decision problems that can be solved by a deterministic Turing machine using a polynomial amount of computation time, or polynomial time.
- NP(Nondeterministic Polynomial time)
  Class of computational decision problems for which a given yes-solution can be verified as a solution in polynomial time by a deterministic Turing machine.
- NP-complete
  Class of decision problems which contains the hardest problems in NP. Each NP-complete problem has to be in NP.
- NP-hard
  A problem H is NP-hard when every problem L in NP can be reduced in polynomial time to H. Class of problems which are at least as hard as the hardest problems in NP.

NP-hard



How to solve combinatorial optimization problem

- Branch and bound
- Cutting plane
- Column generation
- ......

Summary

- Finally, we should master how to settle a problem in that way, so that we will have more ideas and try different methods.
- In theory, the exact methods can guarantee to find the global solution, but in solving some typical practical problems, the price is too high. Because it's hard to accelerate the algorithm that can guarantee to find the optimal solution.
- For most practical problems, it is difficult to find polynomial algorithms, because most of them are NP-hard problems, then the rest selection is to design an algorithm that can jump out of local optimum.

General Optimization problem

Optimization problem

- Linear Optimization
- Non-Linear Optimization

- Convex Optimization
- Non-Convex Optimization

Convex Optimization

- Convex set: A set S is convex if for all members $x, y \in S$ and all $\theta \in [0,1]$, we have that $\theta x + (1-\theta)y \in S$.
- Convex function: For all $\theta \in [0,1]$ and all x, y in S, the following condition holds: $f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$.
- Convex optimization:

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0, i = 1, 2, ..., m \\ & h_j(x) = 0, j = 1, 2, ..., n \end{aligned}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the optimization variable, the function $f : \mathcal{D} \subseteq \mathbb{R}^n \to \mathbb{R}$ is convex, $g_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m$, are convex, and $h_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, p$, are affine.

Standard Problems

- Linear Programming(LP)

$$\begin{aligned} \min \quad & c^T x + d \\ s.t. \quad & G(x) \preceq h \\ & A(x) = b \end{aligned}$$

- Quadratic Programming(QP)

$$\begin{aligned} \min \quad & \frac{1}{2} x^T P x + c^T x + d \\ s.t. \quad & G(x) \preceq h \\ & A(x) = b \end{aligned}$$

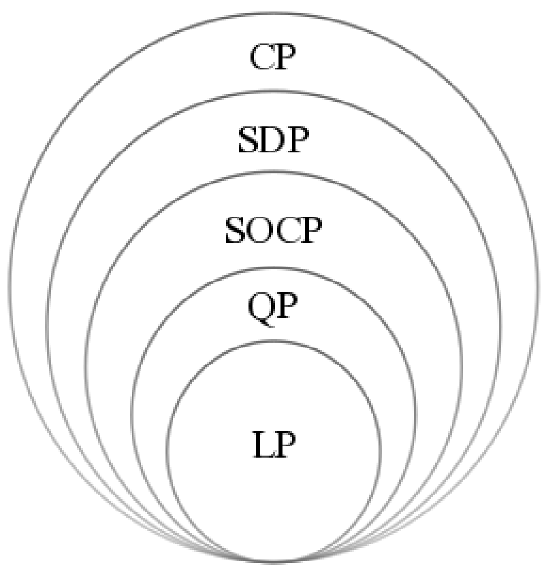
- Semidefinite Programming(SDP)

$$\begin{aligned} \min \quad & tr(CX) \\ s.t. \quad & tr(A_i X) = b_i, i = 1, 2, \ldots p \\ & X \succeq 0 \end{aligned}$$

Other problems

- Least Squares
- Support Vector Machine(SVM)
- Quadratically Contrained Quadratic Program(QCQP)
- Second-Order Cone Program(SOCP)
- Geometric Programming(GP)
- Cone Programming(CP)

A hierarchy of convex optimization problems.



Property

- Every local minimum is a global minimum.
- If the objective function is strictly convex, then the problem has at most one optimal point.
- Non-Convex $\rightarrow$ Convex
- Many methods can be used to solve and wide applications.

Basic iterative approaches

- Line search:
  1. Compute a descent direction p_k
  2. Choose α_k to 'loosely' minimize $h(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{p}_k)$ over $\alpha \in \mathbb{R}_+$
  3. Update $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$, and $k = k + 1$
  4. Until $\|\nabla f(\mathbf{x}_{k+1})\| <$ tolerance.
- Gradient descent:

$$x_{k+1} = x_k + \alpha_k d_k$$

$$f(x_{k+1}) = f(x_k + \alpha_k d_k) < f(x_k)$$

- Subgradient method(for non-differentiable)
  Convergence conditions and Step size rules.

Gradient

Two essential elements: step size and descent direction. All kinds of methods are based on the fundamental.

- Stochastic gradient descent
- Batch gradient descent
- Proximal gradient method
- Newton's Method
- Conjugate gradient method
- Quasi-Newton method

Convergence condition

KKT

$$L(\mathbf{x}, \mu, \lambda) = f(\mathbf{x}) + \mu^\top \mathbf{g}(\mathbf{x}) + \lambda^\top \mathbf{h}(\mathbf{x})$$

where $\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_m(\mathbf{x}))^\top$, $\mathbf{h}(\mathbf{x}) = (h_1(\mathbf{x}), \ldots, h_\ell(\mathbf{x}))^\top$.

- Stationarity
  $f(x)$: $\nabla f(x^*) + \sum_{i=1}^m \mu_i \nabla g_i(x^*) + \sum_{j=1}^\ell \lambda_j \nabla h_j(x^*) = \mathbf{0}$
- Primal feasibility
  $g_i(x^*) \leq 0$, for $i = 1, \ldots, m$ $h_j(x^*) = 0$, for $j = 1, \ldots, \ell$
- Dual feasibility
  $\mu_i \geq 0$, for $i = 1, \ldots, m$
- Complementary slackness
  $\mu_i g_i(x^*) = 0$, for $i = 1, \ldots, m$.
  When there are no inequality constraints, i.e.,$m = 0$, the KKT conditions turn into the Lagrange conditions, and the KKT multipliers are called Lagrange multipliers.

Duality

$$f(x,\lambda) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x)$$
$$g(\lambda) = \min_{x \in \mathcal{D}} f(x,\lambda) \quad \text{Concave}$$

The dual function yields lower bounds on the optimal value p^* of the initial problem.

$$g(\lambda) \le p^*.$$

$$\max_{\lambda} \min_{x} f(x,\lambda) \le \min_{x} \max_{\lambda} f(x,\lambda).$$

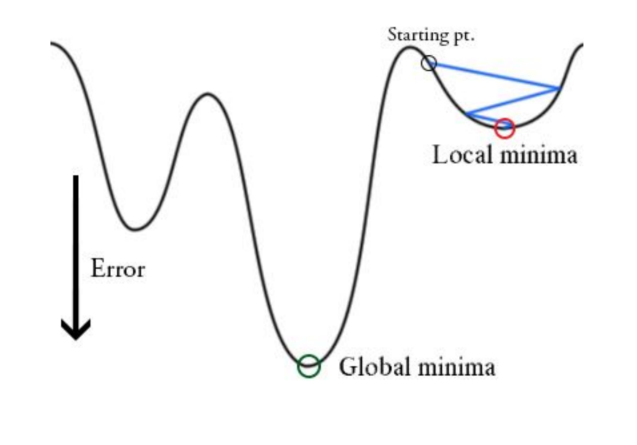
Strong duality: $d^* = \max_{\lambda \ge 0} g(\lambda) = \min f_0 = p^*$.
Constraint qualification such as Slater's condition holds.

Other Methods

- Interior-point method
- Penalty method
- Lagrange multiplier
- Augmented Lagrangian method
- Alternating Direction Method of Multipliers(ADMM)

Non-convex functions

Exponential number of saddle points



Non-convex optimization

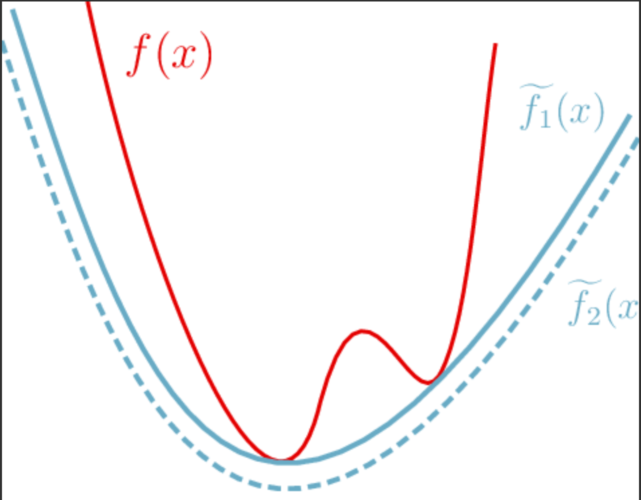
Convex relaxation

Relaxing the non-convex problem to a convex problem.

Heuristics

The important thing is how to jump out of the local optimization.
Change the parameter by experience.

Convex relaxation



Stochastic programming & optimization

- Stochastic programming
  1. Monte Carlo method-sampling
  2. Robust optimization
  3. Stochastic dynamic programming
- Randomized search methods
  1. Simulated annealing
  2. Stochastic hill climbing
  3. Evolutionary algorithms
  4. Swarm algorithms

The End