

Linear and Combinatorial Optimization

Fredrik Kahl

Matematikcentrum

Lecture 9: Algorithm complexity and Dynamic programming

- Algorithm complexity.
 - The simplex method not polynomial.
 - LP polynomial.
 - Interior point methods - Karmarkar
 - The ellipsoid method
- Dynamic programming

Algorithm complexity

A problem is called **decidable** if there is an algorithm that solves the problem. (In finite time).

Example: The traveling salesman's problem is decidable, for example, through exhaustive search.

Time complexity: How long time does the algorithm take?

Memory complexity: How much memory does the algorithm need?

Since it takes time to use memory, the memory complexity is not worse than the time complexity.

Time bounds

We write $f(x) = o(g(x))$ if $\lim_{x \rightarrow \infty} f/g = 0$.

Example: $x^2 = o(x^7)$.

We write $f(x) = O(g(x))$ if there is a constant k such that $f(x) \leq kg(x)$ when x goes to ∞ (that is if f/g is bounded when $x \rightarrow \infty$).

Assumption: all kinds of operations require unit time.

Of all inputs possible, the worst-case behavior decides the complexity of an algorithm.

Rates of growth for algorithms

- polynomial - if $f(n) = O(n^k)$ for some k
- subexponential - if f is not $O(n^k)$ for any k but $O((1 + \epsilon)^n)$ for every $\epsilon > 0$.
- exponential - if $f(n) = O(q^n)$ for some $q > 1$.
- superexponential - if f grows faster than every exponentially growing function.

Usually, one is only interested in the two categories **polynomial** versus **exponential**.

Often, in exponential, both sub and super are included.

Several versions of an optimization problem

- **Optimization:** Find an optimal feasible solution x^* .
- **Decision:** Is there a feasible solution x with $f(x) \leq L$

Example: LP *versus* LI.

LI (linear inequalities) is the following problem:
Given an integer $m \times n$ matrix A and m -vector b ,
is there an n -vector x such that $Ax \leq b$?

Problem classification

Most often the decision version is used to determine a problem's degree of complexity.

A decision problem is in the **class P** if there is a polynomial algorithm for its solution.

A decision problem is in the **class NP** if: when the answer is YES for a given solution, then this solution can be *verified* in polynomial time.

(For NP-problems, one only needs to verify if a given solution is feasible and better than L in polynomial time - *certificate*.)

Problem reduction and NP-completeness

Certain problems are related so that if there is a fast algorithm for solving one problem, one can convert this algorithm to solve the other.

We say a problem (P) **reduces** to another (P') if any algorithm that solves (P') can be converted to an algorithm for solving (P).

There is a collection of NP-problems such that if one of them could be solved in polynomial time, then all others could also be solved in polynomial time. These are called **NP-complete problems**.

Example: Traveling salesman problem, knapsack problem.

A crude categorisation of problems: **easy** problems are those that can be solved in polynomial time and **hard** problems are NP-complete which one suspects require exponential time.

Simplex method not polynomial

Theorem 9.1. *For every $d > 1$ there is an LP problem with $2d$ equations, $3d$ variables, and integer coefficients (≤ 4) such that the simplex method may take $2^d - 1$ iterations to find the optimum.*

Further, one can show for any pivoting rule in the simplex, one can always construct a problem which takes exponential time to solve with simplex.

LP is still in the class P

There are other algorithms with polynomial time complexity.

Examples:

- Karmarkar's algorithm
- Ellipsoid method

Karmarkar's algorithm

Consider the problem

$$\min z = c^T x$$

$$Ax = b$$

$$x \geq 0$$

Suppose we are given a feasible solution $x_0 > 0$, that is, in the interior.

We want to choose Δx such that $x_1 = x_0 + \Delta x$ has a lower value of the goal function and still fulfills the constraints.

$$b = Ax_1 = Ax_0 + A\Delta x = b + A\Delta x, \Rightarrow A\Delta x = 0$$

$$c^T x_1 < c^T x_0, \Rightarrow c^T \Delta x < 0$$

$$0 \leq x_1, \Rightarrow \Delta x \geq -x_0$$

First, we choose direction according to

$$-Pc = -(I - A^T(AA^T)^{-1}A)c$$

and then step length as 0.98 times the distance from x_0 to the border in the direction $-Pc$.

One can show that this algorithm has polynomial time complexity.

Binary search

Relationship between LI (decision problem) and LP (optimization problem).

An integer between 1 and B can be determined with $\log(B)$ questions of the type: Is $x > a$?

LP: Given integer matrices A , b and c , solve the problem $\max c^T x, Ax = b, x \geq 0$.

LI: Given integer matrices A and b , find an x with $Ax = b, x \geq 0$.

Theorem 9.2. *There is a polynomial algorithm for LP if and only if there is a polynomial algorithm for LI (decision problem).*

Proof: (\Rightarrow) For the solution of an LP, one also needs to examine if there are feasible solutions.

(\Leftarrow) By adding constraints of the type $c^T x \geq c_k$ one can bound the optimal goal function with binary search. When enough precision is obtained, one can stop.

(In the excerpt of the book, first it is shown how LI can be used to check if the problem has feasible solutions or if the problem is unbounded, and then one can extract the optimal solution by binary search.)

The Ellipsoid algorithm

Idea:

1. Let a large ellipsoid include all feasible solutions and thereby also the optimal solution.
2. Pick a point in the ellipsoid (for example, the centre). If it is feasible, then stop. Otherwise construct a hyperplane through this point in such a way that it is easy to determine in which half of the ellipsoid possible feasible solutions are contained.
3. Compute the smallest ellipsoid containing the correct half of the ellipsoid.
4. Repeat step 2 and 3 until some stop criteria.

One can show that this algorithm is polynomial.

LP solvers

At the course home page, there is a link to free LP solvers, for example, **PCx**. It is written in c, it is fast and it is based on: Mehrotra's 'predictor-corrector' algorithm with Gondzio's higher order correction strategy.

Matlab:

`xopt=linprog(c,A,b)` solves the LP

$\min c^T x$ such that $Ax \leq b$.

Equality and bound constraints are also possible.

Dynamic programming

Related to branch and bound - implicit enumeration of solutions.

Suppose *discrete-time sequential decision process*, $t = 1, \dots, T$ and decision variables x_1, \dots, x_T .

At time t , the process is in *state* s_{t-1} .

Assumptions:

- (i) the contribution to the objective function for x_t depends only s_{t-1} and x_t .
- (ii) s_t depends only on s_{t-1} and x_t .

Dynamic programming

Problem formulation:

$$\max_{x_1, \dots, x_T} \sum_{t=1}^T g_t(s_{t-1}, x_t) \quad (1)$$

$$s_t = \Phi(s_{t-1}, x_t) \text{ for } t = 1, \dots, T \quad (2)$$

$$s_0 \text{ is given.} \quad (3)$$

Recursive optimization scheme:

Define

$$z_k(s_{k-1}) = \max_{x_k, \dots, x_T} \sum_{t=k}^T g_t(s_{t-1}, x_t)$$

for $k = T, T - 1, \dots, 1$. We want to find $z_1(s_0)$.

Theorem 9.3.

$$z_k(s_{k-1}) = \max_{x_k} \{g_k(s_{k-1}, x_k) + z_{k+1}(s_k)\}$$

Examples: Shortest path problem, knapsack problem.