15.083J/6.859J Integer Optimization

Lecture 2: Efficient Algorithms
and Computational Complexity

1 Outline

Slide 1

- Efficient algorithms
- Complexity
- The classes $\mathcal{P}$ and $\mathcal{NP}$
- The classes $\mathcal{NP}$-complete and $\mathcal{NP}$-hard
- What if a problem is $\mathcal{NP}$ hard?

2 Efficient algorithms

Slide 2

- The LO problem

$$\begin{array}{rl} \min & \boldsymbol{c}'\boldsymbol{x} \\ \text{s.t.} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{array}$$

- A LO instance

$$\begin{array}{rrcrcl} \min & 2x & + & 3y & & \\ \text{s.t.} & x & + & y & \leq & 1 \\ & x & , & y & \geq & 0 \end{array}$$

- A problem is a collection of instances

2.1 Size

Slide 3

- The **size** of an instance is the number of bits used to describe the instance, according to a prespecified format
- A number $r \leq U$

$$r = a_k 2^k + a_{k-1} 2^{k-1} + \cdots + a_1 2^1 + a_0$$

is represented by $(a_0, a_1, \ldots, a_k)$ with $k \leq \lfloor \log_2 U \rfloor$

- Size of r is $\lfloor \log_2 U \rfloor + 2$
- Instance of LO: $(\boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b})$
- Size is

$$(mn + m + n)\big(\lfloor \log_2 U \rfloor + 2\big)$$

- What is an instance of the Traveling Salesman Problem (TSP)?
- What is the size of such an instance?

2.2 Running Time

SLIDE 4

Let A be an algorithm which solves the optimization problem Π.
If there exists a constant $\alpha > 0$ such that A terminates its computation after at most $\alpha\, f(|I|)$ elementary steps for each instance I($|I|$ is the size of I), then A runs in $O(f)$ time.

Elementary operations are

- variable assignments
- random access to variables
- conditional jumps
- comparison of numbers
- arithmetic operations
- ...

SLIDE 5

A "brute force" algorithm for solving the min-cost flow problem:

Consider all spanning trees and pick the best tree solution among the feasible ones.

Suppose we had a computer to check 10^{15} trees in a second. It would need more than 10^9 years to find the best tree for a 25-node min-cost flow problem.
It would need 10^{59} years for a 50-node instance.

That's not efficient!

SLIDE 6

Ideally, we would like to call an algorithm "efficient" when it is sufficiently fast to be usable in practice, but this is a rather vague and slippery notion.

The following notion has gained wide acceptance:

> An algorithm is considered efficient if the number of steps it performs for any input is bounded by a polynomial function of the input size.

Polynomials are, e.g., n, n^3, or $10^6 n^8$.

2.3 The Tyranny of Exponential Growth

SLIDE 7

	$100\, n \log n$	$10\, n^2$	$n^{3.5}$	2^n	$n!$	n^{n-2}
10^9/sec	$1.19 \cdot 10^9$	$600,000$	$3,868$	41	15	13
10^{10}/sec	$1.08 \cdot 10^{10}$	$1,897,370$	$7,468$	45	16	13

Maximum input sizes solvable within one hour.

2.3.1 Pros of the Polynomial View

SLIDE 8

- Extreme rates of growth, such as n^{80} or $2^{n/100}$, rarely come up in practice.
- Asymptotically, a polynomial function always yields smaller values than any exponential function.
- Polynomial-time algorithms are in a better position to take advantage of technological improvements in the speed of computers.
- You can add two polynomials, multiply them, and compose them, and the result will still be a polynomial.

2.4 Punch line

SLIDE 9

The equation

$$\text{efficient} \quad = \quad \text{polynomial}$$

has been accepted as the best available way of tying the empirical notion of a "practical algorithm" to a precisely formalized mathematical concept.

2.5 Definition

SLIDE 10

An algorithm runs in *polynomial time* if its running time is $O(|I|^k)$, where $|I|$ is the input size, and all numbers in intermediate computations can be stored with $O(|I|^k)$ bits.

3 Complexity Theory

3.1 Recognition Problems

SLIDE 11

- A **recognition problem** is one that has a binary answer: YES or NO.
- Example: Is the value of an IO problem less than or equal to B?
- Example: Can a graph be colored with 4 colors?
- Example: Is a number p composite?

3.2 Transformations-reductions

SLIDE 12

- Definition: Let Π_1 and Π_2 be two recognition problems. We say that Π_1 transforms to Π_2 in polynomial if there exist a polynomial time algorithm that given an instance I_1 of of problem Π_1, outputs an instance I_2 of Π_2 with the property that I_1 is a YES instance of Π_1 if and only if I_2 is a YES instance of Π_2.
- Suppose there exists an algorithm for some problem Π_1 that consists of a polynomial time computation in addition to a polynomial number of subroutine calls to an algorithm for problem Π_2. We then say that problem Π_1 **reduces** (in polynomial time) to problem Π_2.

3.3 Properties

SLIDE 13

- Theorem: If problem Π_1 transforms to problem Π_2 in polynomial time, and if Π_2 is solvable in polynomial time, then Π_1 is also solvable in polynomial time.
- Interpretation: a) Π_1 is "no harder" than Π_2; b) Π_2 is "at least as hard" as Π_1; if there existed a polynomial time algorithm for Π_2, then the same would be true for Π_1.
- If we have some evidence that $\Pi_1 \notin \mathcal{P}$, a transformation of Π_1 to Π_2 would provide equally strong evidence that $\Pi_2 \notin \mathcal{P}$.
- Property: If problem problem Π_1 transforms to problem Π_2 and problem Π_2 transforms to problem Π_3, then problem Π_1 transforms to problem Π_3.

4 The classes $\mathcal{P}$-$\mathcal{NP}$

SLIDE 14

- A recognition problem Π is in $\mathcal{P}$ if it is solvable in polynomial time.
- Is $\boldsymbol{Ax} = \boldsymbol{b}$, $\boldsymbol{x} \geq \boldsymbol{0}$ feasible? It is in $\mathcal{P}$.
- A problem Π belongs to $\mathcal{NP}$ if given an instance I of Π, there exists a certificate of size polynomial in the size of I, such that together with this certificate we can decide, whether I is a YES instance in polynomial time.
- BIO: is the problem $\boldsymbol{Ax} \leq \boldsymbol{b}$, $\boldsymbol{x} \in \{0,1\}^n$ feasible?
- Certificate: A feasible solution $\boldsymbol{x}_0$. We can check whether $\boldsymbol{Ax}_0 \leq \boldsymbol{b}$.
- TSP: Is there a tour of length less than or equal to L? Is $TSP \in \mathcal{NP}$?
- Property: $\mathcal{P} \subseteq \mathcal{NP}$.
- Open problem: Is $\mathcal{P} = \mathcal{NP}$?

5 The class $\mathcal{NP}$-complete

SLIDE 15

- A problem Π is $\mathcal{NP}$-complete if $\Pi \in \mathcal{NP}$ and all other problems in $\mathcal{NP}$ polynomially reduce to it.
- Theorem: BIO is $\mathcal{NP}$-complete.
- Theorem: TSP is $\mathcal{NP}$-complete.
- A problem Π is $\mathcal{NP}$-hard if all other problems in $\mathcal{NP}$ polynomially reduce to it.
- A polynomial time algorithm for an $\mathcal{NP}$-hard problem would imply $\mathcal{P} = \mathcal{NP}$.
- Thousands of DOPs are $\mathcal{NP}$-hard. Examples: knapsack, facility location, set covering, set packing, set partitioning, sequencing with setup times, and traveling salesman problems.

5.1 Proving $\mathcal{NP}$-hardness

SLIDE 16

- Theorem: Suppose that a problem Π_0 is $\mathcal{NP}$-hard and that Π_0 can be transformed (in polynomial time) to another problem Π. Then, Π is $\mathcal{NP}$-hard.
- Useful theorem as there are thousands of $\mathcal{NP}$-hard problems. Any one of these problems can play the role of Π_0, and this provides us with a lot of latitude when attempting to prove $\mathcal{NP}$-hardness of a given problem Π.

SLIDE 17

- Given a problem Π whose $\mathcal{NP}$-hardness we wish to establish, we search for a known $\mathcal{NP}$-hard problem Π_0 that appears to be closely related to Π. We then attempt to construct a transformation of Π_0 to Π. Coming up with such transformations is mostly an art, based on ingenuity and experience, and there are very few general guidelines.

5.2 Example

Slide 18

- ΔTSP: Given a complete undirected graph, a bound L and costs $c_{ij} = c_{ji}$:

$$c_{ij} \le c_{ik} + c_{kj}, \qquad \forall\ i, j, k.$$

  Does there exists a tour with cost less than or equal to L?

- Theorem: ΔTSP is $\mathcal{NP}$-complete.
- Hamilton circuit: Given an undirected graph does there exists a tour?

Slide 19

- We transform Hamilton circuit to ΔTSP. Since Hamilton circuit is $\mathcal{NP}$-hard, this will imply that ΔTSP is also $\mathcal{NP}$-hard.
- Given an instance $G = (\mathcal{N}, \mathcal{E})$ of Hamilton circuit, with n nodes, we construct an instance of ΔTSP, again with n nodes:

$$c_{ij} = \begin{cases} 1, & \text{if } \{i, j\} \in E, \\ 2, & \text{otherwise.} \end{cases}$$

  We also let $L = n$.

- This is an instance of ΔTSP.
- The transformation can be carried out in polynomial time [$O(n^2)$ time suffices].

Slide 20

- If we have a yes instance of Hamilton circuit, there exists a tour that uses the edges in $\mathcal{E}$. Since these edges are assigned unit cost, we obtain a tour of cost n, and we have a yes instance of ΔTSP.
- This argument can be reversed to show that if we have a yes instance of ΔTSP, then we also have a yes instance of Hamilton circuit.

6 What if a problem is $\mathcal{NP}$-hard?

Slide 21

- $\mathcal{NP}$-hardness is not a definite proof that no polynomial time algorithm exists. It is possible but unlikely that BIO$\in \mathcal{P}$, and $\mathcal{P} = \mathcal{NP}$. Nevertheless, $\mathcal{NP}$-hardness suggests that we should stop searching for a polynomial time algorithm, unless we are willing to tackle the $\mathcal{P} = \mathcal{NP}$ question.
- $\mathcal{NP}$-hardness can be viewed as a limitation on what can be accomplished; very different from declaring the problem "intractable" and refraining from further work. Many $\mathcal{NP}$-hard problems are routinely solved in practice. Even when solutions are approximate, without any quality guarantees, the results are often good enough to be useful in a practical setting.

Slide 22

- Not all $\mathcal{NP}$-complete problems are equally hard. The knapsack problem can be solved in time $O(n^2 c_{\max})$, exponential in the size $O\big(n(\log c_{\max} + \log w_{\max}) + \log K\big)$ of the input data; the running time may be acceptable for the range of values of $c_{\max}$ that arise in certain applications.
- In the knapsack problem, $\mathcal{NP}$-hardness is only due to large numerical input data. Other problems, however, remain $\mathcal{NP}$-hard even if the numerical data are restricted to take small values. The ΔTSP where the costs c_{ij} are either 1 or 2 is $\mathcal{NP}$-hard. Complexity due to combinatorial structure not numerical data.