AI & Big Data에 꼭 필요한 최적화 기법 실습캠프

Fundamental Optimization Techniques Essential for Artificial Intelligence and Big Data Analytics

06 Computational Complexity & Complexity Analysis in LP

Sang Jin Kweon (권상진)

School of Management Engineering
UNIST

Office hours: after class (or by APPT.)
Email: sjkweon@unist.ac.kr
Tel: +82 (52) 217-3146
Web: http://or.unist.ac.kr

TA

Seungok Woo (우승옥)
- Email: wso1017@unist.ac.kr
- Office Hours: by APPT.

Acknowledgement

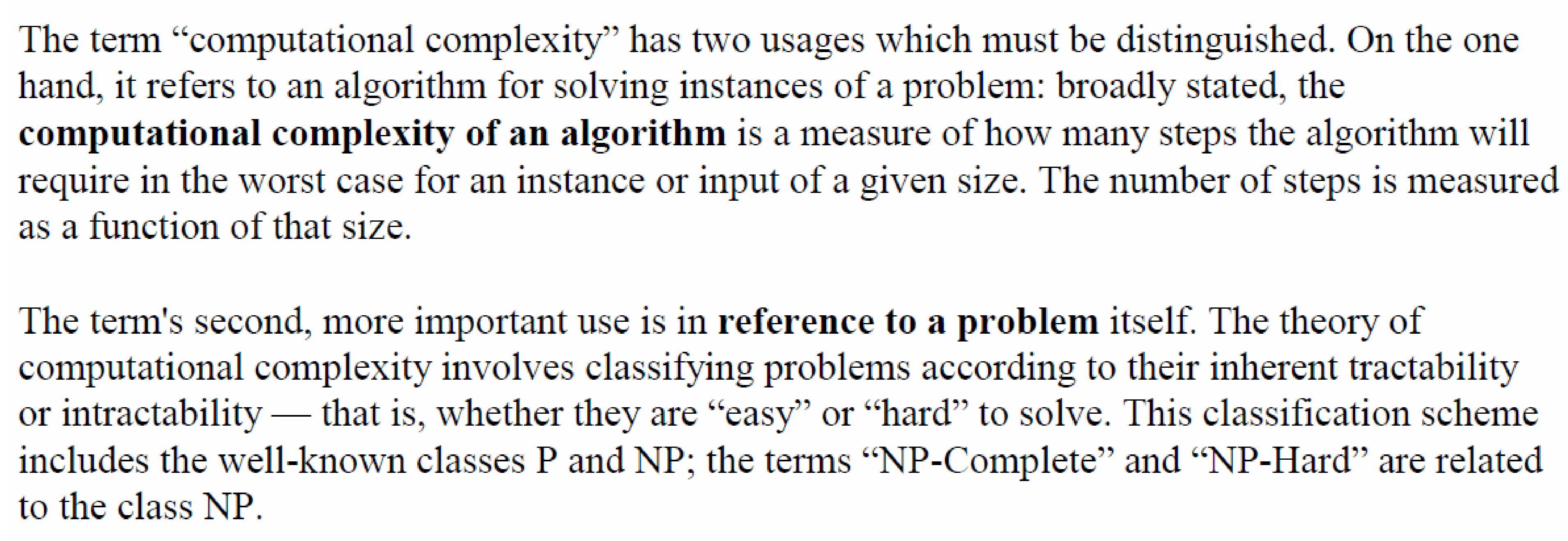
본 캠프는 한동대학교 공학교육혁신센터의
SW 단기교육과정 개발사업비로 개최되었습니다.

Hicee.handong.edu

Overview

- Computational Complexity
- Time Complexity of an Algorithm
  - Definitions
  - Time complexity function
- Time Complexity of a Problem
- Classes P and NP
  - Class P
  - Class NP
  - Polynomially Reducible Problems
  - Class NP-Complete
  - Examples
- Classes co-NP
- Introduction to Complexity Analysis in Linear Programming (LP)
- Complexity of the Simplex Method
- Klee-Minty LPs on Perturbed Cubes
- Polynomial-Time Algorithms for LP

Computational Complexity



The term "computational complexity" has two usages which must be distinguished. On the one hand, it refers to an algorithm for solving instances of a problem: broadly stated, the **computational complexity of an algorithm** is a measure of how many steps the algorithm will require in the worst case for an instance or input of a given size. The number of steps is measured as a function of that size.

The term's second, more important use is in **reference to a problem** itself. The theory of computational complexity involves classifying problems according to their inherent tractability or intractability — that is, whether they are "easy" or "hard" to solve. This classification scheme includes the well-known classes P and NP; the terms "NP-Complete" and "NP-Hard" are related to the class NP.

Time Complexity of an Algorithm

Problem: class of problems.

Instance: particular case of a problem.

Size of an instance (ν): depends upon the type of problem and the encoding convention used to list the data of the instance.

Time complexity function ($f(\nu)$): of an algorithm gives the maximum number of operations (steps) that would be required to solve an instance of size ν.

$f(\nu)$ is of order $g(\nu)$, $O(g(\nu))$, if $\lim_{\nu\to\infty} f(\nu)/g(\nu) = c$,

e.g., the non-polynomial function $a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_1 2^1 + a_0$ is $O(2^n)$.

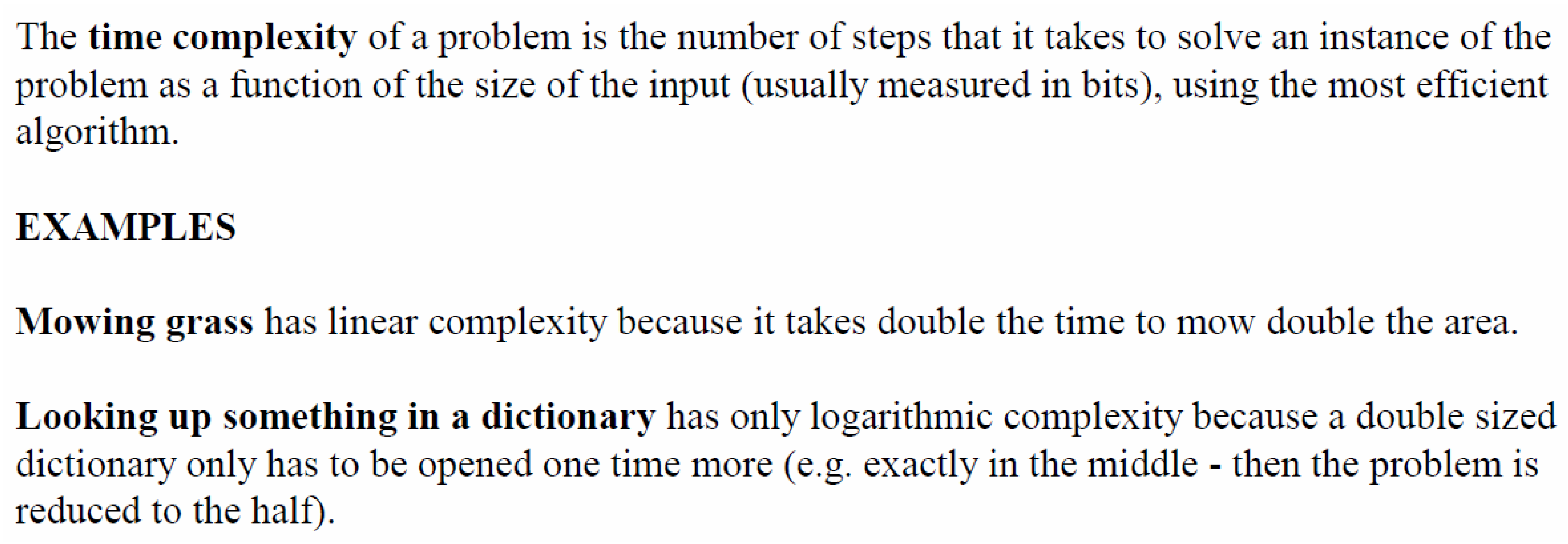
An algorithm has **polynomial time complexity** if its time complexity function $f(\nu)$ is $O(p(\nu))$ for some polynomial $p(\nu)$. Otherwise, the algorithm has **exponential time complexity**.

EXAMPLE

If an algorithm requires $3n^4 + 2n^2 + 5n$ steps to solve a problem, where n is the size of an instance, then

$$\lim_{n \to \infty} \frac{3n^4 + 2n^2 + 5n}{n^4} = 3 \quad \Rightarrow \quad \text{The time complexity of the algorithm is } O(n^4).$$

Time Complexity of a Problem



The **time complexity** of a problem is the number of steps that it takes to solve an instance of the problem as a function of the size of the input (usually measured in bits), using the most efficient algorithm.

EXAMPLES

Mowing grass has linear complexity because it takes double the time to mow double the area.

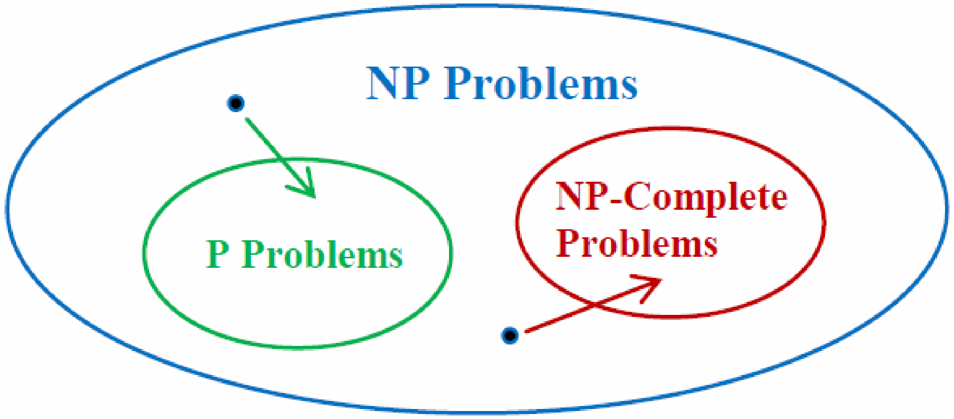
Looking up something in a dictionary has only logarithmic complexity because a double sized dictionary only has to be opened one time more (e.g. exactly in the middle - then the problem is reduced to the half).

Classes P and NP

Class P consists of all decision (recognition) problems for which algorithms with polynomial time behavior have been found.
Class NP is essentially the set of decision (recognition) problems for which algorithms with exponential behavior have been found and whose positive solutions can be verified in polynomial time given the right information.

$$\Rightarrow \quad P \subseteq NP$$

$$P \neq NP \quad \text{or} \quad P = NP\,? \quad \textbf{(Unsolved Problem)}$$



The question of whether P is the same set as NP is the most important open question in theoretical computer science. There is even a \$1,000,000 prize for solving it offered by the **Clay Mathematics Institute** (**CMI**), Cambridge, MA.

Problem Π_1 is **polynomially reducible** to problem Π_2 if we can reduce Π_1 to Π_2 in polynomial time ($\Pi_1 \propto \Pi_2$).

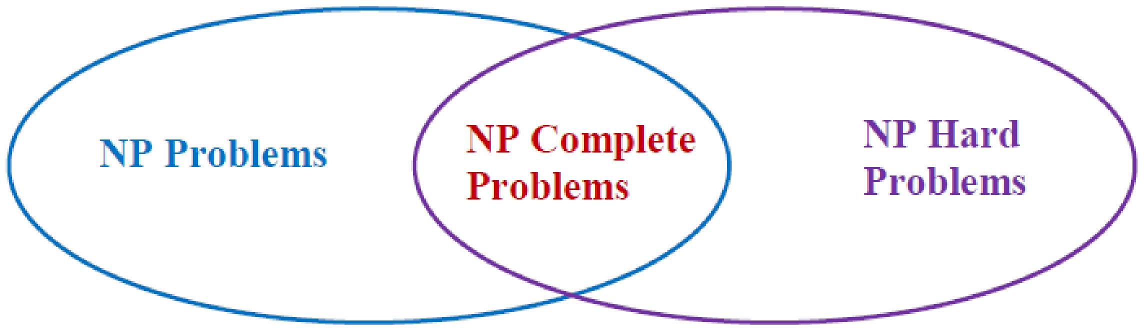
$$\Pi_1 \propto \Pi_2 \text{ and } \Pi_2 \propto \Pi_3 \Rightarrow \Pi_1 \propto \Pi_3 \text{ (transitive).}$$

A problem Π lying in NP is **NP-Complete** if every other problem in NP is polynomially reducible to Π.

An algorithm has **pseudo-polynomial time complexity** if its maximum number of operations is bounded by a polynomial $q(\nu, \lambda)$, where ν is the instance size and λ is an upper bound on the magnitude of each of the data.

An algorithm is **genuinely or strongly polynomial** if its complexity depends only on the instance size (ν) and is independent of the magnitude of each of the data.

The term **NP-hard** refers to any problem that is at least as hard as any problem in NP. Thus, the **NP-Complete problems** are precisely **the intersection of the class of NP-Hard problems with the class NP**. In particular, optimization problems whose recognition versions are NP-Complete (such as the TSP) are NP-Hard, since solving the optimization version is at least as hard as solving the recognition version.



For a problem Π and a polynomial $p(v)$ of the problem size, we define problem Π_p as Π with the restriction that all data are bounded by $p(v)$.

Problem Π is **strongly NP-hard or NP-complete**, or **NP-Complete in the strong sense** if for some polynomial $p(v)$, Π_p is NP-Complete.

Some NP-complete and NP-hard problems are pseudo-polynomially solvable (sometimes these are called **weakly NP-hard or-complete**, or **NP-complete in the ordinary sense**).

Classes P and NP

Examples

Knapsack Problem with Integer Variables

The Knapsack problem with integer variables is an NP-Hard problem in the ordinary sense. It can be solved by a dynamic programming algorithm requiring a number of steps polynomial in the size of the knapsack (W) and the number of items (n). The pseudo-polynomial time complexity is $O(n\ W)$.

Travelling Salesman Problem (TSP)

The **travelling salesman problem** (**TSP**) is one of the most studied combinatorial optimization problems in operations research and theoretical computer science. Given a list of cities and their pairwise distances, the task is to find a shortest possible tour that visits each city exactly once. TSP is **strongly NP-Hard** (or **NP-Complete in the strong sense**).

Maximum Flow Problem

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 (i.e., arc capacities are irrational). When the **arc capacities are integers**, the runtime of Ford-Fulkerson is bounded by $O(E\,f)$, 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. Thus, the maximum flow problem is a polynomial problem. Thus, Ford-Fulkerson algorithm is **genuinely or strongly polynomial**.

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, where V is the number of nodes. This algorithm is also **genuinely or strongly polynomial**.

Minimum Spanning Tree

The minimum spanning tree (MST) is a minimum tree that spans the entire graph. **MST problem can be solved in polynomial time**. If the edge weights are integers, then deterministic algorithms are known that solve the problem in $O(m+n)$ integer operations, where m is the number of edges, n is the number of vertices.

Steiner Tree

The problem of finding the **Steiner tree** of a subset of the vertices, that is, minimum tree that spans the given subset, is known to be **NP-Complete in the strong sense**.

Chromatic Number

The **chromatic number** of a graph is the smallest number of colors needed to color the vertices of the graph so that no two adjacent vertices share the same color. This problem is **NP-Complete in the strong sense**.

Clique Problem

A clique in a graph is a set of pairwise adjacent vertices. The clique problem is the optimization problem of finding a clique of maximum size in a graph (the maximal complete subgraph). The **Clique Problem** is **NP-Complete in the strong sense**.

Introduction to Complexity Analysis in LP

$$
\begin{array}{lll}
\text{(LP)} & \text{minimize } z = & c\,x\,, \\
 & & \\
 & \text{subject to} & A\,x = b\,, \\
 & & x \geq \bar{0}\,,
\end{array}
$$

where A : $m \times n$ matrix,
b : m-dim column vector,
c : n-dim row vector,
and x : n-dim column vector of decision variables.

(A, b, c) : input data (integer values) ,
L : number of binary digits needed to record all the data,
(m, n, A, b, c) defines an instance of the LP,
(m, n, L) defines the size of the instance.

$$L \approx \left[\begin{array}{l} 1+\log_2 m+\log_2 n+\sum\limits_{j=1}^{n}\left[1+\log_2\left(1+\left|c_j\right|\right)\right] \\ +\sum\limits_{i=1}^{m}\sum\limits_{j=1}^{n}\left[1+\log_2\left(1+\left|a_{ij}\right|\right)\right]+\sum\limits_{i=1}^{m}\left[1+\log_2\left(1+\left|b_i\right|\right)\right] \end{array} \right]$$

- Complexity of an algorithm becomes a function of the size, i.e., $f(m, n, L)$.

- If $\exists\ \tau > 0$ such that the total # of elementary operations required by the algorithm in any instance is $\leq \tau f(m, n, L)$, then the algorithm is of order $O[f(m, n, L)]$.

- If $f(m, n, L)$ is a polynomial function of m, n, L, then the algorithm is a polynomial-time algorithm.

- Otherwise, it is a non-polynomial-time (or exponential-time) algorithm.

Complexity of the Simplex Method

- Total # of elementary operations = (# of elementary operations at each iteration) $\times$ (# of iterations)

- # of elementary operations at each iteration of the Simplex method is $O(m\,n)$.

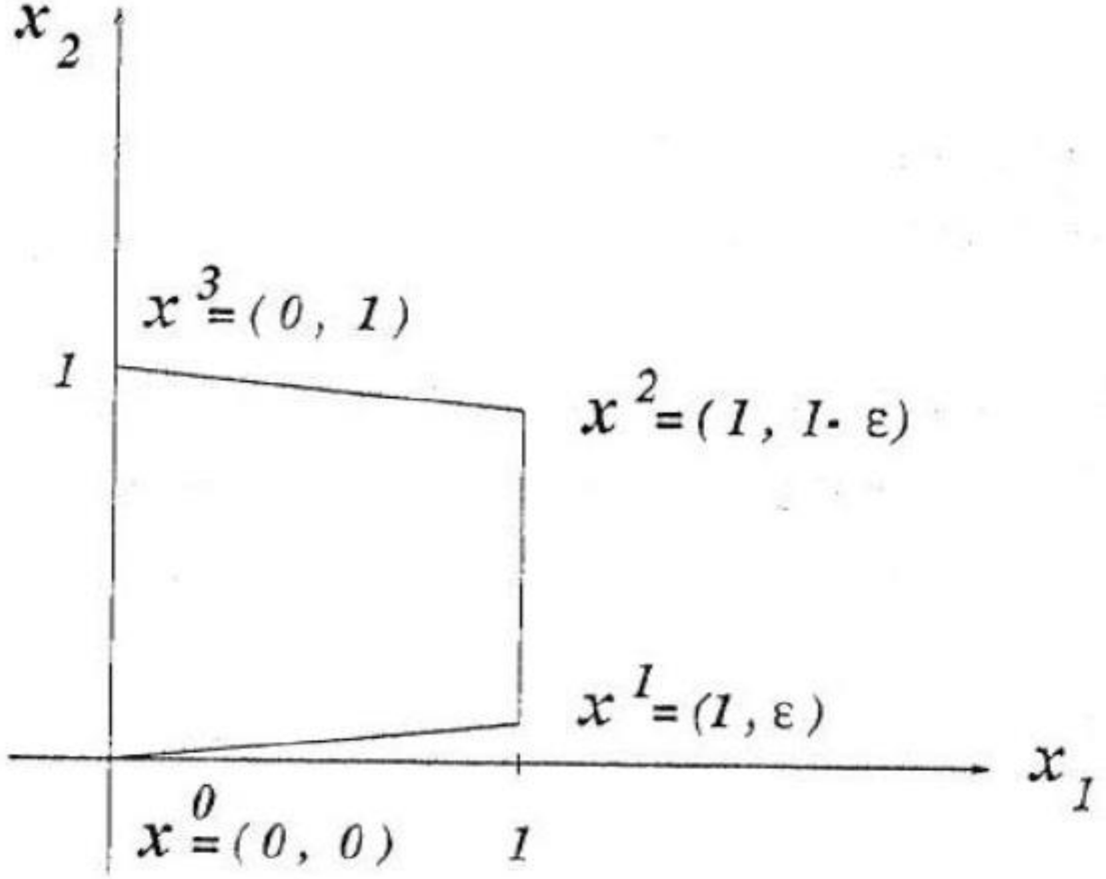
- From practical experience, the Simplex method takes about $(\alpha\, m)$ iterations, where $e^{\alpha} < \log_2 \left(2 + n/m\right)$. Hence, it is of $O\left(m^2\, n\right)$.

- From the worst-case analysis, Klee and Minty [1972] identified a class of LPs (on perturbed d-dimensional cubes) for which the Simplex method requires $2^d - 1$ iterations to generate their optimal tableaus.

Klee-Minty LPs on Perturbed Cubes

2-dim Example:

$$
\begin{aligned}
&\text{minimize } z = && -x_2, \\
&\text{subject to} && x_1 \geq 0, \\
& && x_1 \leq 1, \\
& && x_2 \geq \varepsilon\, x_1, && \text{where } 0 < \varepsilon < 1/2, \\
& && x_2 \leq 1 - \varepsilon\, x_1 .
\end{aligned}
$$

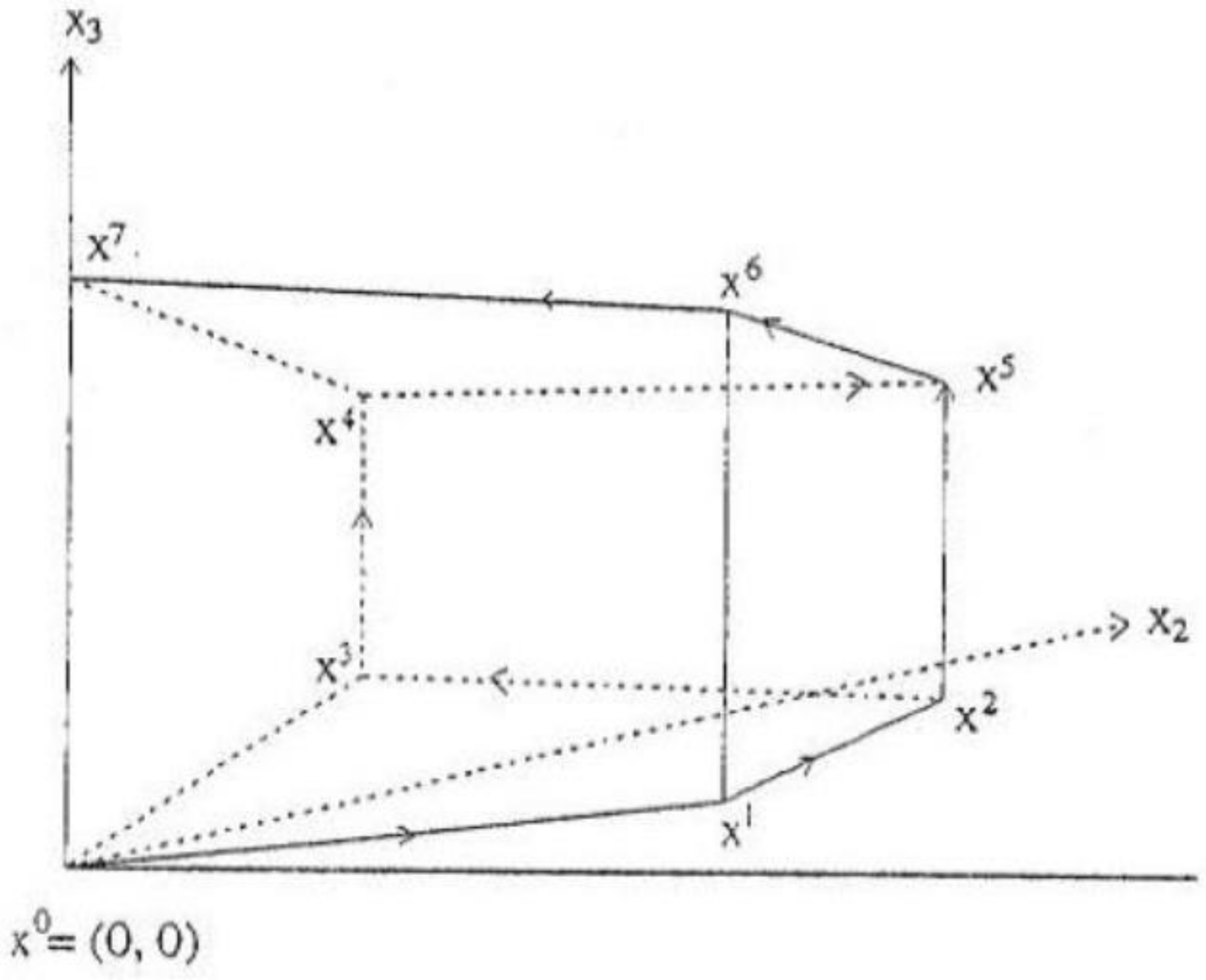


$x^0 \rightarrow x^1 \rightarrow x^1 \rightarrow x^3$ (optimal)

$2^2 - 1 = 3$ iterations

3-dim Example:

$$
\begin{aligned}
\text{minimize } z = \; & -x_3, \\
\text{subject to } \; & x_1 \geq 0, \\
& x_1 \leq 1, \\
& x_2 \geq \varepsilon x_1, \qquad \text{where } 0 < \varepsilon < \tfrac{1}{2}, \\
& x_2 \leq 1 - \varepsilon x_1, \\
& x_3 \geq \varepsilon x_2, \\
& x_3 \leq 1 - \varepsilon x_2 .
\end{aligned}
$$



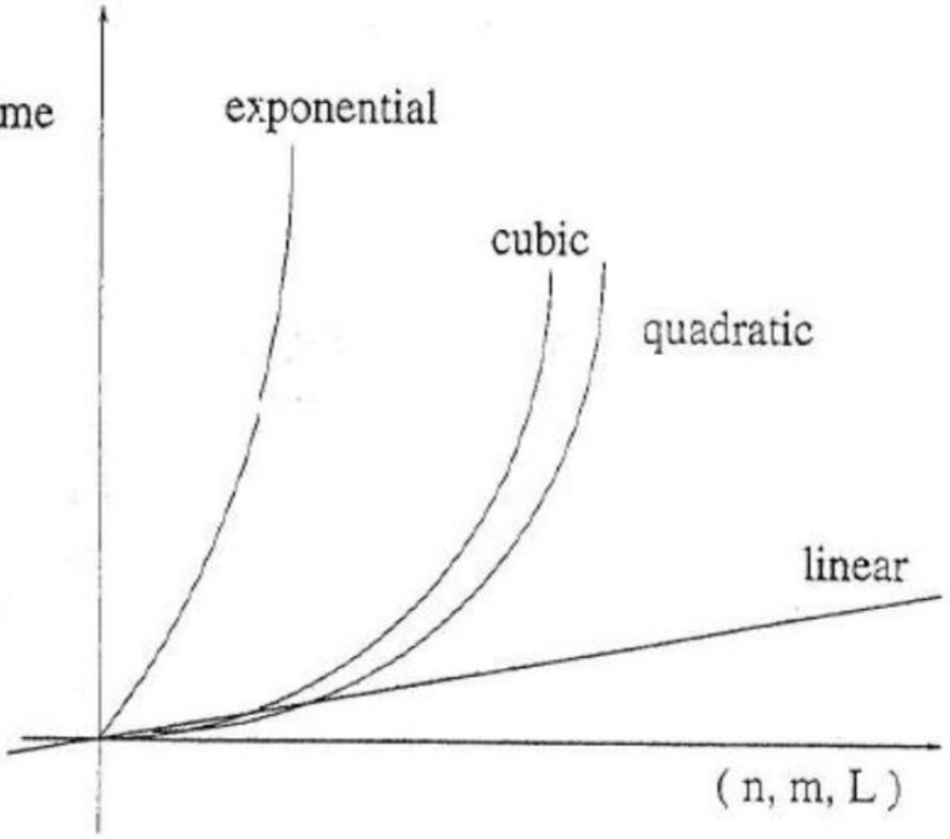
$2^3 - 1 = 7$ iterations

d-dim Example:

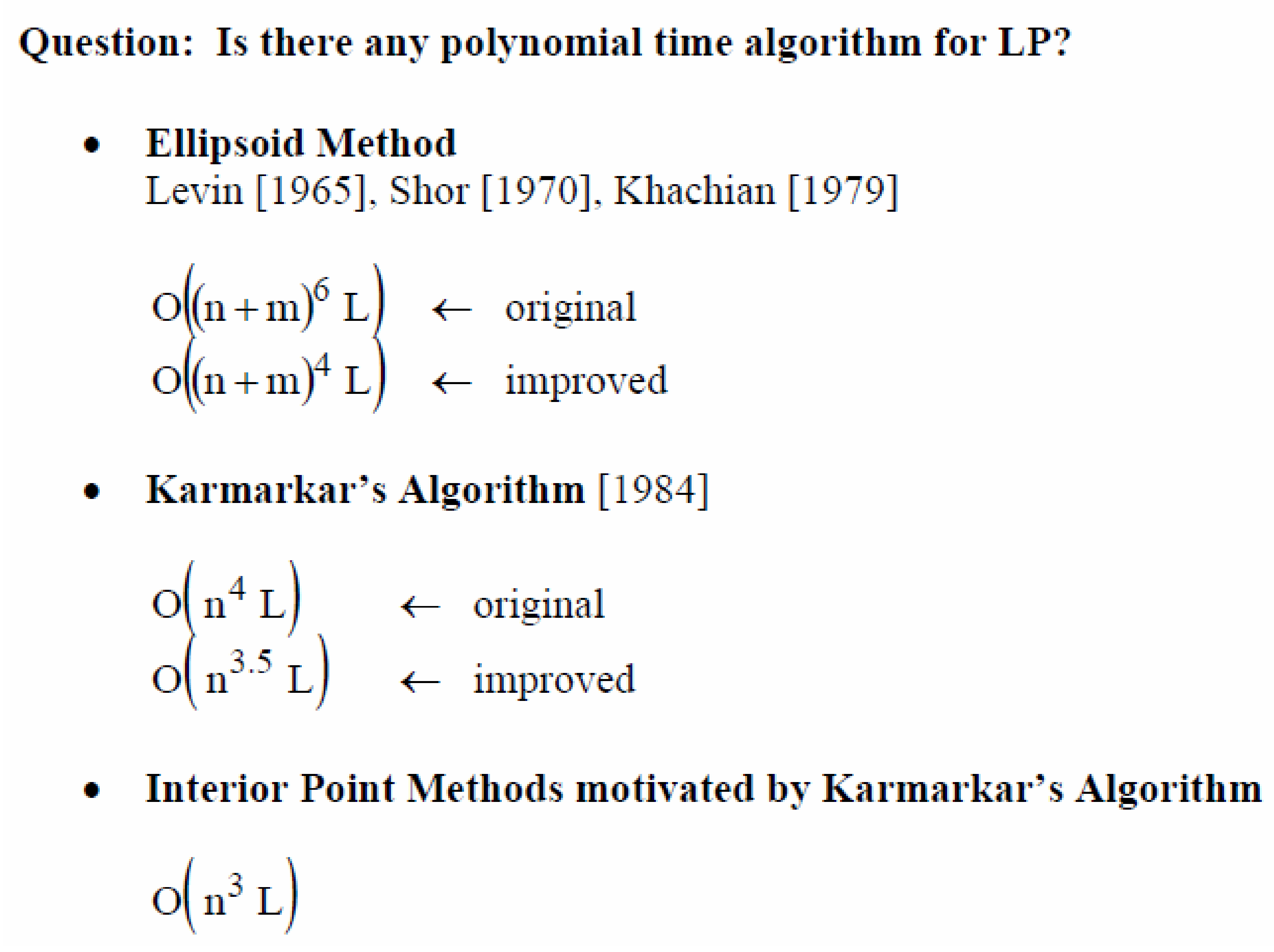
$$\begin{aligned}
\text{minimize } z = \quad & -x_3, \\
\text{subject to} \quad & x_1 \geq 0, \\
& x_1 \leq 1, \\
& x_2 \geq \varepsilon x_1, \qquad\qquad \text{where } 0 < \varepsilon < 1/2, \\
& x_2 \leq 1 - \varepsilon x_1, \\
& \quad \ldots\ldots \\
& x_d \geq \varepsilon x_{d-1}, \\
& x_d \leq 1 - \varepsilon x_{d-1}.
\end{aligned}$$

Conclusion: In theory, the Simplex method is not a polynomial-time algorithm. It is an exponential-time algorithm.

Question: What is wrong with this?



Polynomial-Time Algorithms for LP



Question: Is there any polynomial time algorithm for LP?

- **Ellipsoid Method**
  Levin [1965], Shor [1970], Khachian [1979]

  $O\left((n+m)^6 L\right) \quad \leftarrow \quad \text{original}$

  $O\left((n+m)^4 L\right) \quad \leftarrow \quad \text{improved}$

- **Karmarkar's Algorithm** [1984]

  $O\left(n^4 L\right) \quad \leftarrow \quad \text{original}$

  $O\left(n^{3.5} L\right) \quad \leftarrow \quad \text{improved}$

- **Interior Point Methods motivated by Karmarkar's Algorithm**

  $O\left(n^3 L\right)$

Acknowledgement

본 캠프는 한동대학교 공학교육혁신센터의
SW 단기교육과정 개발사업비로 개최되었습니다.

Hicee.handong.edu