IE 631 Integer Programming **KAIST, Spring 2023**

Lecture #2: Terminologies and complexity of integer programming March 2, 2023

Lecturer: Dabeen Lee

1 Outline

In this lecture, we study

- integer programming formulation,
- computational complexity of integer programming.

2 Integer programming formulation

Let us formally define the intege programming problem. A **pure integer linear program** is an optimization problem of the following form.

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \le b, \\ & x \in \mathbb{Z}^d \end{aligned} \tag{2.1}$$

where

- $c \in \mathbb{R}^d$,
- A is an $m \times d$ matrix that has rows $a_1^\top, \ldots, a_m^\top$ and $b \in \mathbb{R}^m$ has entries $b_1, \ldots, b_m$ so that $Ax \le b$ consists of linear inequalities $a_1^\top x \le b_1, \ldots, a_m^\top x \le b_m$, and
- $\mathbb{Z}^d$ is the set of d-dimensional vectors all whose entries are integers.

We usually assume that all entries of A, b, c are rational, i.e. $A \in \mathbb{Q}^{m\times d}$, $b \in \mathbb{Q}^m$, and $c \in \mathbb{Q}^d$. Hereinafter, we say that a vector x is **integral** if all entries of x are integers. We call $x \in \mathbb{Z}^d$ the **integrality constraint**. The integer program (2.1) is called **pure** because all variables are restricted to be **integral**.

Many applications have their decision variables nonnegative, in which case we need constraints $x \ge 0$. Here, we may assume that the system of linear inequalities $Ax \le b$ includes $x \ge 0$.

Decision-making problems may also have decisions of **fractional** values as well as integer values. To model such a problem, we define a **mixed integer linear program (MIP or MILP)** as follows.

$$\begin{aligned} \max \quad & c^\top x + h^\top y \\ \text{s.t.} \quad & Ax + Gy \le b, \\ & x \in \mathbb{Z}^d, \ y \in \mathbb{R}^p \end{aligned} \tag{2.2}$$

where A, b, c, G, h are vectors/matrices of appropriate dimension with rational entries. It is common to omit the constraint $y \in \mathbb{R}^p$ from (2.2). When there is no continuous variable, i.e. $p = 0$, (2.2) reduces to a pure integer linear program (2.1). Throughout the course, we refer to mixed integer linear programs simply as **integer programs**.

The **feasible region** or the **solution set** of (2.2) is the set of solutions satisfying the linear constraints and the integrality constraints:

$$S = \left\{(x,y) \in \mathbb{Z}^d \times \mathbb{R}^p : \; Ax + Gy \leq b\right\}.$$

A set of the form S is often referred to as a **mixed integer linear set**.

An important concept is the notion of **relaxation**. The **linear programming relaxation** or the **LP relaxation** of (2.2) is the same optimization problem except that the integrality constraints are relaxed:

$$\begin{aligned} \max \quad & c^\top x + h^\top y \\ \text{s.t.} \quad & Ax + Gy \leq b. \end{aligned} \tag{2.3}$$

Hence, the LP relaxation is, by definition, a linear program. In fact, the notion of relaxation applies to the feasible region intself. Basically, we say that a set S_0 is a relaxation of the feasible region S if $S \subseteq S_0$. Hence,

$$P = \left\{(x,y) \in \mathbb{R}^d \times \mathbb{R}^p : \; Ax + Gy \leq b\right\}$$

which is the feasible region of the LP relaxation (2.3) gives rise to a relaxation of S. We could take

$$P' = \left\{(x,y) \in \mathbb{R}^d \times \mathbb{R}^p : \; A'x + G'y \leq b'\right\}$$

with a different system of linear inequalities as long as $S \subseteq P'$.

Example 2.1. Recall the simple example from the last lecture. The LP relaxation of

$$\begin{aligned} \max \quad & 4x + 5y \\ \text{s.t.} \quad & x + 3y \leq 10, \\ & 3x + y \leq 10, \\ & x, y \geq 0, \\ & (x,y) \in \mathbb{Z}^2. \end{aligned}$$

is given by

$$\begin{aligned} \max \quad & 4x + 5y \\ \text{s.t.} \quad & x + 3y \leq 10, \\ & 3x + y \leq 10, \\ & x, y \geq 0. \end{aligned}$$

Moreover, $\{(x,y) \in \mathbb{R}^2_+ : x + 3y \leq 10, 3x + y \leq 10\}$ is a relaxation of $\{(x,y) \in \mathbb{Z}^2_+ : x + 3y \leq 10, 3x+y \leq 10\}$. Here, $\mathbb{R}_+$ is the set of nonnegative real numbers while $\mathbb{Z}_+$ is the set of nonnegative integers.

Combinatorial optimization problems and other problems in operations research involve decisions of binary values. For such problems, we need binary variables. A **mixed 0,1 linear program** or a **mixed binary linear program** is of the form

$$\begin{aligned} \max \quad & c^\top x + h^\top y \\ \text{s.t.} \quad & Ax + Gy \leq b, \\ & x \in \{0,1\}^d, \; y \in \mathbb{R}^p \end{aligned} \tag{2.4}$$

where $\{0,1\}^d$ is the set of all length d binary strings. The feasible region

$$S = \left\{(x,y) \in \{0,1\}^d \times \mathbb{R}^p : \ Ax + Gy \le b\right\}$$

is often referred to as a **mixed 0,1 linear set** or a **mixed binary linear set**.

3 Computational complexity of integer programming

Take a mixed 0,1 linear program as in (2.4). Note that it can be written as

$$\begin{aligned} \max \quad & f(x) \\ \text{s.t.} \quad & x \in \{0,1\}^d \end{aligned} \tag{2.5}$$

where

$$f(x) = c^\top x + \max\left\{h^\top y : \ Gy \le b - Ax\right\}.$$

Given a binary vector $x \in \{0,1\}^d$, evaluating $f(x)$ boils down to solving a linear program, for which there are known methods. When (2.4) is a pure 0,1 linear program, we do not have any continuous variables, so evaluating $f(x)$ is equivalent to checking whether $0 \le b - Ax$ holds or not.

Then a pure mathematician would say that the problem is trivial because there is only a finite number of binary vectors, so we can simply enumerate all vecrtors. However, we are concerned about the computational complexity. The number of binary vectors is 2^d, which grows exponentially fast in the ambient dimension d, and enumerating 2^d vectors is what we want to avoid.

Example 2.2 (Traveling Salesman Problem (TSP))**.** Given n points in the Euclidean space, the Traveling Salesman Problem is to find a minimum length **tour** that visits every point precisely once. The total number of tours is $(n-1)!/2$. Note that $10!/2$ is roughly 1.8 million. Hence, enumerating all poissble tours of n points requires going through 1.8 million tours!

In fact, 100! is approximately 9.9×10^{157}, and 1000! is roughly 4×10^{2567}. Note that the number of entire atoms in the universe is about 10^{80}.

Recall that integer programming is NP-hard. What does this really mean? To understand the computational complexity of integer programming, let us briefly touch upon the basics of complexity theory.

3.1 Problems, instances, enconding size

A **problem** means a question to be answered for any set of data. For example, linear programming, integer programming, and TSP. An **instance** of a problem is given by a specific data set. (2.1) gives an instance of linear programming and an instance of integer programming.

Given a problem instance, we consider its **encoding size** to measure the amount of space required to write down the data of the instance. All integers are written in binary encoding, and to encode an integer n, we need one bit to represent its sign (+ or -) and $\lceil \log_2(|n|+1)\rceil$ bits to encode $|n|$. Then the encoding size of integer n is $1 + \lceil \log_2(|n|+1)\rceil$. For a rational number p/q where q is a positive integer, we need $1 + \lceil \log_2(|p|+1)\rceil + \lceil \log_2(|q|+1)\rceil$ bits.

The encoding size of a rational vector

$$\left(\frac{p_1}{q_1}, \ldots, \frac{p_d}{q_d}\right)$$

would be

$$\sum_{j=1}^{d} \left(1 + \lceil \log_2(|p_j| + 1) \rceil + \lceil \log_2(|q_j| + 1) \rceil\right)$$

plus some extra space to construct a vector from scalars. Similarly, the encoding size of an $m \times d$ matrix whose entry at row i and column j is given by p_{ij}/q_{ij} is

$$\sum_{i=1}^{m} \sum_{j=1}^{d} \left(1 + \lceil \log_2(|p_{ij}| + 1) \rceil + \lceil \log_2(|q_{ij}| + 1) \rceil\right).$$

Recall that a mixed integer linear program as in (2.2) is given by data A, b, c, G, h. Hence, the encoding size of an MILP is the encoding size of A, b, c, G, h, and the encoding size will basically be the input size of the MILP. Therefore, if all entries of A, b, c, G, h are integers, then the encoding size is given by

$$(1 + \lceil \log_2(M + 1) \rceil) \cdot (d + p + m + (d + p)m)$$

where M is the largest absolute value of an entry in A, b, c, G, h.

Proposition 2.3. *Let A be an invertible $d \times d$ rational matrix and b be a d-dimensional rational vector. Then the encoding size of the unique solution to the system $Ax = b$ is polynomially bounded by the encoding size of (A, b).*

Proof. Suppose that the entry of A at row i and column j is given by p_{ij}/q_{ij} where p_{ij} is an integer and q_{ij} is a positive integer. Let

$$M = \prod_{i=1}^{d} \prod_{j=1}^{d} q_{ij}, \quad A' = M \cdot A, \quad b' = M \cdot b.$$

Then the system $Ax = b$ is equivalent to $A'x = b'$, and all entries of A', b' are integers. Then, by Cramer's rule, each entry of the unique solution x is given by

$$x_i = \frac{\det(A'_i)}{\det(A')}$$

where A'_i is what is obtained from A' after replacing its ith column by b'.

Let θ be the largest absolute value of an entry in (A', b'). In fact,

$$\theta \le M \cdot \max_{i \in [d], j \in [d]} |p_{ij}|,$$

and therefore, the encoding size of θ is at most

$$\log M + \max_{i,j \in [d]} (1 + \lceil \log_2(|p_{ij}| + 1) \rceil) = \sum_{i=1}^{d} \sum_{j=1}^{d} (1 + \log_2(|q_{ij}| + 1)) + \max_{i,j \in [d]} (1 + \lceil \log_2(|p_{ij}| + 1) \rceil) \le L$$

where L is the encoding size of (A, b). Moreover, by the Leibniz formula of matrix determinent, it follows that

$$\det(A') \le d!\theta^d.$$

Similarly, we obtain that $\det(A'_i) \le d!\theta^d$ for every $i \in [d]$. As the determinants $\det(A')$ and $\det(A'_i)$ are integers, their encoding sizes are bounded above by

$$O(\log(d!\theta^d)) = O(d \log d + d \times \text{the encoding size of } \theta) = O(d \log d + dL).$$

As x has d entries, it follows that the encoding size of x is

$$O(d^2 \log d + d^2 L),$$

as required. $\square$

3.2 Polynomial algorithm and complexity class P

We say that a function $f : S \to \mathbb{R}$ is **polynomially bounded** by another function $g : S \to \mathbb{R}$ if there is some **polynomial** $\phi : \mathbb{R} \to \mathbb{R}$ such that

$$f(s) \leq \phi(g(s)) \quad \forall s \in S.$$

Next we recall the big O notation. Given functions $f : \mathbb{R} \to \mathbb{R}_+$ and $g : \mathbb{R} \to \mathbb{R}_+$, we write that

$$f(x) = O(g(x))$$

if there exists some fixed constant M and $x_0 \in \mathbb{R}$ such that

$$f(x) \leq Mg(x) \quad \forall x \geq x_0.$$

The **running time** of an **algorithm** is measured as the number of arithmetic operations carried out by the algorithm. A **polynomial (time) algorithm** for a problem is an algorithm which solves the problem in **polynomial time**, and equivalently, in **time polynomially bounded by the encoding size of the input instance**.

For example, the ellipsoid method due to [Kha80] performs $O(d^6 L)$ arithemtic operations for solving a linear programming instance where d is the number of variables and L is the encoding size. Karmarkar's interior point method takes $O(d^{2.5} L)$ operations for a linear programming instance [Kar84, Vai89]. Note that the encoding size L should be at least the number of variables d, and therefore, the ellipsoid method and the interior point method are polynomial algorithms for linear programming.

The **complexity class P** is the class of all problems for which there is a polynomial algorithm. Then linear programming belongs to the complexity class P. Moreover, remember that bipartite matching can be solved by a linear programming formulation, so bipartite matching also belongs to class P.

3.3 Complexity class NP and NP-hardness

A **decision problem** is a problem whose answer is either *yes* or *no*. The **complexity class NP** is the class of decision problems where the yes answer has a **certificate** that can be checked in polynomial time by a **deterministic Turing machine**. For example, the decision version of TSP is the problem of finding a tour whose length less than some value k. A certificate for the yes answer would be a tour of length less than k, and computing the length of the given tour can be done in polynomial time (we just look at the graph). The decision version of integer programming is the problem of determining if there is an integer solution whose objective is less than k (for the minimization problem). Again, a certificate for the yes answer would be an integer solution, and we can compute its objective value in polynomial time (we just plug in the given solution to the objective function).

Alternatively, the class NP is the class of decision problems that can be solved in polynomial time by a **non-deterministic Turing machine**.

Similarly, the **complexity class co-NP** is the class of decision problems where the no answer has a **certificate** that can be checked in polynomial time by a deterministic Turing machine.

As each problem in complexity class P can be solved in polynomial time by a deterministic Turing machine, class P is contained in the intersection of NP and co-NP, denoted as **NP** $\cap$ **co-NP**. A big open question is as to whether P=NP. Most computer scientists believe that the answer is no.

A decision problem Q in NP is called **NP-complete** if all other problems in NP can be reduced to Q in polynomial time. Cook [Coo71] proved that the decision version of integer programming is NP-complete. A problem Q that is not necessarily a decision problem nor a decision problem is **NP-hard** if all other problems in NP can be reduced to Q in polynomial time.

References

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery. 3.3

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, page 302–311, New York, NY, USA, 1984. Association for Computing Machinery. 3.2

[Kha80] Leonid.G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980. 3.2

[Mey74] R. R. Meyer. On the existence of optimal solutions to integer and mixed integer programming problems, 1974.

[Vai89] P.M. Vaidya. Speeding-up linear programming using fast matrix multiplication. In *30th Annual Symposium on Foundations of Computer Science*, pages 332–337, 1989. 3.2