**IE 631 Integer Programming** **KAIST, Spring 2023**
**Lecture #7: Integer programming formulations II** March 21, 2023
Lecturer: Dabeen Lee

## 1 Outline

In this lecture, we study

- the cutting stock problem,
- set packing, partitioning, and covering,
- stable set problem,
- *st*-paths and *st*-cuts,
- airline crew scheduling.

## 2 Cutting stock problem

A steel mill manages $p$ steel plate production lines. Each line can produce large steel plate of width $W$. The steel mill receives orders of different widths. Suppose that there are $m$ different orders of width values $w_1, \ldots, w_m$. Assume also that the number of orders for width $w_i$ is $b_i$ for $i \in [m]$. Upon receiving these orders, the steel mill decides how to allocate the production orders over production lines. Basically, a steel plate of width $W$ from a production line can be cut into multiple plates of different widths. Here, the steel mill wants to run as few production lines as possible.

To model this problem as an integer program, we use a binary variable $y_j$ to indicate whether production line $j \in [p]$ is on use or not.

$$y_j = \begin{cases} 1, & \text{if production line } j \text{ is on operation} \\ 0, & \text{oterwise} \end{cases}.$$

Next, for the orders of width $w_i$, we can allocate the $b_i$ orders over the production lines that are being used. Then we use a nonnegative variable $z_{ij}$ to encode the number of orders of width $w_i$ allocated to production line $j$. Then the problem can be formulated as

$$\begin{aligned} \min \quad & \sum_{j=1}^{p} y_j \\ \text{s.t.} \quad & \sum_{i=1}^{m} w_i z_{ij} \leq W y_j, \quad \forall j \in [p], \\ & \sum_{j=1}^{p} z_{ij} \geq b_i, \quad \forall i \in [m], \\ & y_j \in \{0,1\}, \quad \forall j \in [p], \\ & z_{ij} \in \mathbb{Z}_+, \quad \forall i \in [m],\ j \in [p]. \end{aligned}$$

Although this gives rise to a valid formulation, computational experiments show that it is not a strong formulation.

Next we propose a different formulation. We use the idea of **cutting patterns**. The basic idea is as follows. A cutting pattern determines how to cut a steel plate of width $W$ into pieces of different widths. A cutting pattern can be represented as a vector $s \in \mathbb{Z}_+^m$ where $s_i$ represents the number of pieces of width $w_i$. To make sure that $s_i$ pieces of width $w_i$ for $i \in [m]$ can be produced from a steel plate of width $W$, we impose a knapsack constraint as follows.

$$\sum_{i=1}^{m} w_i s_i \leq W.$$

Then the set of nonnegative integer vectors satisfying the knapsack constraint, given by

$$\mathcal{S} = \left\{ s \in \mathbb{Z}_+^m : \sum_{i=1}^{m} w_i s_i \leq W \right\}$$

collects all possible cutting patterns. Essentially, using a cutting pattern is equivalent to operating a production line.

Let $x_s$ denote the number of production lines that produce cutting pattern $s$. Then

$$\sum_{s \in \mathcal{S}} x_s$$

is the total number of production lines in use. Moreover,

$$\sum_{s \in \mathcal{S}} s_i x_s$$

is equal to the total number of pieces of width $w_i$. Then the problem can be formulated as

$$\begin{aligned} \min \quad & \sum_{s \in \mathcal{S}} x_s \\ \text{s.t.} \quad & \sum_{s \in \mathcal{S}} s_i x_s \geq b_i, \quad \forall i \in [m], \\ & x_s \in \mathbb{Z}_+, \quad \forall s \in \mathcal{S}. \end{aligned}$$

Here, the issue with this integer program is that the number of variables is equal to the number of all possible cutting patterns. However, the number of possible patterns is the number of points in $\mathcal{S}$, which can be huge depending on the problem parameters. Instead of enumerating all cutting patterns, we use the so-called **column generation** approach.

## 3 Set packing, covering, partitioning

Let $E = \{1, \ldots, n\}$ be a finite set of elements, and $\mathcal{F} = \{F_1, \ldots, F_m\}$ be a family of subsets of $E$. Assume that each element $e \in E$ has weight $w_e \in \mathbb{R}$. A set $S \subseteq E$ is said to be a **packing** of the family $\mathcal{F}$ if $S$ intersects every member of $\mathcal{F}$ **at most once**. The family of packings of $\mathcal{F}$ is given by

$$\mathcal{S}^1 = \left\{ x \in \{0,1\}^n : \sum_{j \in F_i} x_j \leq 1, \ \forall F_i \in \mathcal{F} \right\}.$$

Here, we may represent $\mathcal{S}^1$ with a matrix inequality. Let $A$ be the subset-element incidence matrix of $\mathcal{F}$, i.e., $A$ is an $m \times n$ matrix with

$$a_{ij} = \begin{cases} 1, & \text{if member } i \text{ contains element } j, \\ 0, & \text{otherwise.} \end{cases}$$

Then $\mathcal{S}^1$ can be rewritten as

$$\mathcal{S}^1 = \{x \in \{0,1\}^n : \ Ax \le 1\}.$$

The **set packing problem** is

$$\max\left\{\sum_{e\in E} w_e x_e : \ Ax \le 1, \ x \in \{0,1\}^n\right\}. \tag{7.1}$$

Conversely, for any $m \times n$ matrix $A$ all whose entries are either 0 or 1, the problem of the form (7.1) is the set packing problem for a family. In fact, this family corresponds to the rows of the 0,1 matrix $A$.

A set $S \subseteq E$ is said to be a **partitioning** of the family $\mathcal{F}$ if $S$ intersects every member of **exactly once**. The family of partitionings of $\mathcal{F}$ is given by

$$\begin{aligned}\mathcal{S}^2 &= \left\{x \in \{0,1\}^n : \ \sum_{j\in F_i} x_j = 1, \ \forall F_i \in \mathcal{F}\right\} \\ &= \{x \in \{0,1\}^n : \ Ax = 1\}.\end{aligned}$$

The **set partitioning problem** is

$$\min\left\{\sum_{e\in E} w_e x_e : \ Ax = 1, \ x \in \{0,1\}^n\right\}.$$

A set $S \subseteq E$ is said to be a **covering** of the family $\mathcal{F}$ if $S$ intersects every member of $\mathcal{F}$ **at least once**. The family of coverings of $\mathcal{F}$ is given by

$$\begin{aligned}\mathcal{S}^3 &= \left\{x \in \{0,1\}^n : \ \sum_{j\in F_i} x_j \ge 1, \ \forall F_i \in \mathcal{F}\right\} \\ &= \{x \in \{0,1\}^n : \ Ax \ge 1\}.\end{aligned}$$

The **set covering problem** is

$$\min\left\{\sum_{e\in E} w_e x_e : \ Ax \ge 1, \ x \in \{0,1\}^n\right\}.$$

## 4 Stable set problem

Let $G = (V, E)$ be an undirected graph with $n$ vertices and $m$ edges. A **stable set** or an **independent set** of $G$ is a set of nodes no two of which are adjacent. Figure 7.1 shows a graph on 5 vertices and two stable sets. The family of stable sets is given by

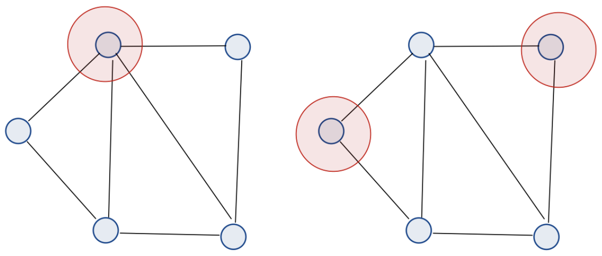

Figure 7.1: Stable sets of a graph

$$\operatorname{stab}(G) = \{x \in \{0,1\}^n : \ x_i + x_j \le 1, \ \forall \{i,j\} \in E\}.$$

The **stable set problem** or the **independent set problem** is the problem of finding a maximum weight stable set in $G$:

$$\max \left\{ \sum_{j \in V} w_j x_j : \ x_i + x_j \le 1, \ \forall \{i,j\} \in E, \ x \in \{0,1\}^n \right\}.$$

where $w_v \in \mathbb{R}$ is the weight of vertex $v \in V$. Note that the stable set problem is an instance of the set packing problem where the vertex set $V$ and the edge set $E$ correspond to the elements and the members, respectively.

The integer programming formulation for the stable set problem has constraints that correspond to edges of $G$. In fact, we may strengthen the formulation based on a combinatorial property of a stable set. A **clique** in a graph is a set of pairwise adjacent vertices. As any two vertices in a clique

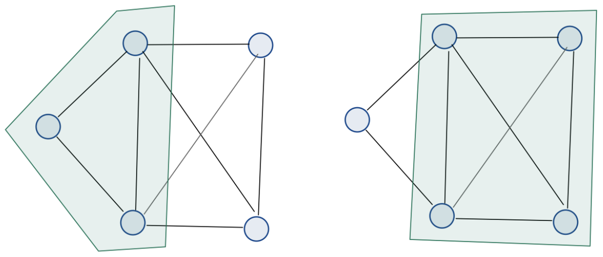

Figure 7.2: Cliques a graph

are adjacent, a stable set intersects a clique with at most one vertex. This implies that inequality

$$\sum_{j \in K} x_j \le 1$$

for any clique $K$ is valid for $\operatorname{stab}(G)$. This inequality is called a **clique inequality**. Note that an edge itself is a clique, and therefore, the edge inequality $x_i + x_j \le 1$ is also a clique inequality. Although all clique inequalities are valid for $\operatorname{stab}(G)$, the ones associated with **maximal cliques** are non-redundant. A clique is **maximal** if it is a clique and there is no other clique that properly contains it. Hence,

$$\operatorname{stab}(G) = \left\{ x \in \{0,1\}^n : \ \sum_{j \in K} x_j \le 1, \ \forall K \in \mathcal{K} \right\}$$

where $\mathcal{K}$ is the family of maximal cliques in $G$.

## 5 $st$-paths and $st$-cuts

Let $G = (V, E)$ be a graph with two distinct vertices $s$ and $t$. Let $w_e \in \mathbb{R}_+$ be the length of edge $e \in E$ in graph $G$. An ***st*-path** is a path in $G$ that starts from $s$ and ends at $t$. An ***st*-cut** is a set of edges of the form

$$\delta(S) = \{\{u, v\} \in E : \ u \in S, \ v \notin S\}$$

where $S \subseteq V$ contains $s$ but not $t$. Here, $\delta(S)$ is the set of edges exactly one of whose two ends is in $S$ as in Figure 7.3.

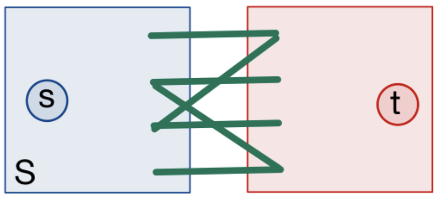

Figure 7.3: An $st$-cut in $G$

Note that an $st$-path $P$ must intersect an $st$-cut $C$. That means that if $x \in \{0, 1\}^E$ gives rise to an $st$-path, then

$$\sum_{e \in C} x_e \geq 1$$

is valid for any $st$-cut $C$. It is known that

$$\begin{aligned} \min \quad & \sum_{e \in E} w_e x_e \\ \text{s.t.} \quad & \sum_{e \in C} x_e \geq 1, \quad \forall st\text{-cuts } C, \\ & x_e \in \{0, 1\}, \quad \forall e \in E \end{aligned}$$

computes a minimum weight $st$-path. In fact, the LP relaxation given by

$$\begin{aligned} \min \quad & \sum_{e \in E} w_e x_e \\ \text{s.t.} \quad & \sum_{e \in C} x_e \geq 1, \quad \forall st\text{-cuts } C, \\ & 0 \leq x_e \leq 1, \quad \forall e \in E \end{aligned}$$

always reutrns an integral solution, meaning that this linear program computes a minimum weight $st$-path. Moreover, the linear program

$$\begin{aligned} \min \quad & \sum_{e \in E} w_e x_e \\ \text{s.t.} \quad & \sum_{e \in C} x_e \geq 1, \quad \forall st\text{-paths } P, \\ & 0 \leq x_e \leq 1, \quad \forall e \in E \end{aligned}$$

finds a minimum weight *st*-cuts.

## 6 Airline crew scheduling

An airline company operate flight schedules to which airline crews are assigned. An airline crew can work for a certain number of hours on a day, so a crew may attend more than one flight on a day. Based on labor regulations and restrictions, an airline can provide a possible list of daily schedules that can be covered by a single crew. For example, a schedule may consist of

- the 8:30-10:00 am flight from Pittsburgh to Chicago,
- the 11:30am-1:30 pm flight from Chicago to Atalanta,
- the 2:45-4:30 pm flight from Atlanta to Pittsburgh.

Another possible schedule is

- the 6:00-9:00 am flight from Gimpo to Jeju,
- the 11:00am-1:00 pm flight from Jeju to Tokyo,
- the 4:00-6:00 pm flight from Tokyo to Gimpo.

We often refer to a schedule of multiple flights that can be covered by a single crew as a **pairing**. Hence, the airline can take care of all its daily flights by pairings. Using a certain pairing of flights corresponds to assigning a crew.

To determine the smallest number of crews to cover all daily flights, we use integer programming. Suppose that there are $m$ flights and $n$ possible pairings of flights. Let $A$ be an $m \times n$ matrix with

$$a_{ij} = \begin{cases} 1, & \text{if pairing } j \text{ covers flight } i, \\ 0, & \text{otherwise.} \end{cases}$$

Next, binary variable $x_j$ indicates

$$x_j = \begin{cases} 1, & \text{if pairing } j \text{ is used,} \\ 0, & \text{otherwise.} \end{cases}$$

Note that a certain pairing $j$ is associated with a cost, depending on its duration and the number of flights, etc. Let $w_j$ be the cost of pairing $j$. Then

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} w_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad \forall \text{ flight } i, \\ & x \in \{0,1\}^n. \end{aligned}$$