IE 511: Integer Programming, Spring 2021 **26 Jan, 2021**

Lecture 1: Introduction, Combinatorial Explosion, IP Formulations

Lecturer: Karthik Chandrasekaran *Scribe: Karthik*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications.*

1.1 Introduction: Discrete Optimization Problems

We frequently encounter problems related to managing and efficiently using scarce resources to increase productivity. These problems are broadly known as optimization problems. Several optimization problems of practical relevance require us to optimize over a discrete decision domain. These are known as discrete optimization problems. Discrete optimization problems occur in numerous areas including production planning, scheduling for airlines, trains and for sports leagues (MLB, NBA, NFL), network design, graph theory, and Gerrymandering (a disruptive application).

Integer Programming (IP) is a convenient formulation of discrete optimization problems. The purpose of this course is to provide the mathematical foundations underlying IPs and their solving techniques.

1.2 Combinatorial Optimization Problem (COP) Formulation

A natural way to formulate a discrete optimization problem (given in words) is as a combinatorial optimization problem, abbreviated as COP. A COP is a formulation of the following type:

Input: A finite set N with element costs $c : N \to \mathbb{R}$ and a collection $\mathcal{F}$ of "feasible" subsets of N.

Goal: $min\{\sum_{j \in S} c_j : S \in \mathcal{F}\}$.

Below, we will see two classic discrete optimization problems that can be formulated as COPs.

1. **Knapsack Problem.** *(In words)* We have a knapsack with weight capacity W (that we can carry while hiking) and n items with weights $w_1, \ldots, w_n$ and utilities/valuations $p_1, \ldots, p_n$ respectively. We would like to choose a subset of items to maximize utility without exceeding the capacity of the knapsack.

   The knapsack problem can be formulated as a COP: The ground set is $N := [n]$ with the element cost function $c : N \to \mathbb{R}$ given by $c_j := -p_j$ and the set $\mathcal{F}$ of feasible sets given by $\mathcal{F} := \{S \subseteq N : \sum_{j \in S} w_j \leq W\}$.

2. **Assignment Problem.** *(In words)* We have n servers and n jobs with cost c_{ij} for server i to perform job j. We would like to assign jobs to servers so that each job is assigned to exactly one server and each server is assigned exactly one job so that the total cost of the assignment is minimized.

   The assignment problem can be formulated as a COP: The ground set is $N := [n] \times [n] = \{(i, j) : i, j \in [n]\}$ with the element cost function $c : N \to \mathbb{R}$ given by c_{ij}. We define a subset S of N is to be an *assignment/perfect matching* if (i) for every $i \in [n]$, there exists a unique

pair (i, k) in S and for every $j \in [n]$, there exists a unique pair (k, j) in S. The set $\mathcal{F}$ of feasible sets in the COP is given by $\mathcal{F} :=$ collection of assignments.

1.2.1 Drawbacks of COP formulation: Combinatorial Explosion

It all started with a big bang ...

Although we had a concise way to state both discrete optimization problems above (in words), note that both COP formulations are very large. This is because the number of feasible solutions is exponential-sized. In fact,

1. the number of feasible solutions to the knapsack problem over n items could be as large as $2^{n/2}$ (consider the instance where $w_i = 1$ for every item $i \in [n]$ while the weight capacity is $W = n/2$), and
2. the number of feasible solutions to the assignment problem over n servers and n jobs is $n!$ (as we have a one-to-one correspondence between assignments and permutations).

This phenomenon is known as *Combinatorial Explosion*.

A natural algorithmic technique to solve a COP is to enumerate the cost of all feasible solutions and output the best. However, if the number of feasible solutions is large (as is the case in the above two problems), then such an explicit enumeration technique will take a lot of time and space for large values of n. Consequently, COP formulations are not amenable to solution techniques. On the contrary, Integer Programming formulations (IPs) are typically concise and lend themselves to solution techniques that are broadly applicable to solve numerous discrete optimization problems. For this reason, IP is considered to be a robust formulation. A remarkably rich variety of problems can be formulated as IPs.

1.3 IP: Formulations

The standard recipe to formulate a discrete optimization problem (typically given in words) as an IP is the following:

1. Identify decision variables (sometimes you will need auxiliary variables).
2. Define constraints so that feasible points correspond to feasible solutions of the problem.
3. Define objective so that an optimal solution of the IP corresponds to an optimal solution of the problem.

We will formulate the above discrete optimization problems as IPs now.

1. **Knapsack Problem.**
   - Decision variables:

$$x_j := \begin{cases} 1 & \text{if item } j \text{ is chosen} \\ 0 & \text{otherwise} \end{cases}$$

- Constraints:

$$\sum_{j=1}^{n} w_j x_j \leq W$$
$$x_j \in \{0,1\}\ \forall j \in [n]$$

- Objective: $\max \sum_{j=1}^{n} p_j x_j$

Summarizing, we have the following IP formulation for the knapsack problem:

$$\max \sum_{j=1}^{n} p_j x_j$$
$$\sum_{j=1}^{n} w_j x_j \leq W$$
$$x_j \in \{0,1\}\ \forall j \in [n]$$

Note that the above formulation is concise - the size of the formulation is polynomial in the size of the input.

2. **Assignment Problem.**

- Decision Variables:

$$x_j := \begin{cases} 1 & \text{if job } j \text{ is assigned to server } i \\ 0 & \text{otherwise} \end{cases}$$

- Constraints:

$$\sum_{j=1}^{n} x_{ij} = 1\ \forall i \in [n]$$
$$\sum_{i=1}^{n} x_{ij} = 1\ \forall j \in [n]$$
$$x_{ij} \in \{0,1\}\ \forall i,j \in [n]$$

- Objective: $\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$

Summarizing, we have the following IP formulation for the assignment problem:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$
$$\sum_{j=1}^{n} x_{ij} = 1\ \forall i \in [n]$$
$$\sum_{i=1}^{n} x_{ij} = 1\ \forall j \in [n]$$
$$x_{ij} \in \{0,1\}\ \forall i,j \in [n]$$

Note that the above formulation is concise - the size of the formulation is polynomial in the size of the input.

3. **Traveling Salesman Problem (TSP).** Several techniques developed to solve and understand IPs originated with the goal of solving the TSP. *(In words)* We have a salesman who wants to visit n cities exactly once and return to the starting city with least travel cost. The travel cost c_{ij} to travel from city i to city j is known in advance. A "tour" is a sequence of cities that starts and ends at the same city and visits every city exactly once. Given the costs, the goal is to find a min cost tour.

   Exercise. Formulate TSP as a COP.

   We now formulate an IP for TSP.

   - Decision Variables:

$$x_{ij} := \begin{cases} 1 & \text{if city } j \text{ is visited immediately after city } i \\ 0 & \text{otherwise} \end{cases}$$

   - Objective function: min $\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$
   - Constraints:

$$x_{ij} \in \{0,1\} \ \forall i, j \in [n]$$

$$\sum_{j=1}^{n} x_{ij} = 1 \ \forall i \in [n] \text{ (tour leaves city } i \text{ exactly once)}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \ \forall j \in [n] \text{ (tour arrives at city } j \text{ exactly once)}$$

     The above constraints are insufficient to correctly formulate TSP. This is because, subtours are feasible for the above constraints but they do not give a tour. For instance, the solution $x_{12} = x_{23} = x_{31} = x_{45} = x_{56} = x_{64} = 1$ satisfies the above contraints, but does not give a feasible tour. This suggests that we need to add more constraints to eliminate points corresponding to subtours.

     For every subset of cities, we need to avoid a subtour through them. We note that for each subset, a valid tour will leave the subset at least once. This is expressed by the constraints

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1, \forall \ \emptyset \neq S \subsetneq [n].$$

   Summarizing, we have the following IP formulation for TSP:

$$\begin{aligned} \min \ & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ & x_{ij} \in \{0,1\}, \ \forall i, j \in [n] \\ & \sum_{j=1}^{n} x_{ij} = 1 \ \forall i \in [n] \\ & \sum_{i=1}^{n} x_{ij} = 1 \ \forall j \in [n] \\ & \sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1, \forall \ \emptyset \neq S \subsetneq [n] \end{aligned}$$

Note that the number of constraints in this IP formulation is exponential in the input size. However, there are also polynomial-sized IP formulations for TSP (for those of you who enjoy mathematical challenges, this a good exercise - come up with a polynomial-sized IP formulation for TSP).

4. **Facility Location Problem.** *(In words)* We have m clients and n potential depots. For each depot, there is a cost o_j associated with opening the depot. Furthermore, there is a transportation cost c_{ij} to serve client i from depot j. A depot can serve a client only if it is open. We would like to determine which depots to open and which amongst the open depots will serve each client so that the total cost (opening + serving cost) is minimized.

   Exercise. Formulate the facility location problem as a COP.

   We now formulate an IP for the facility location problem.

   - Decision Variables:

$$y_j := \begin{cases} 1 & \text{if depot } j \text{ is opened} \\ 0 & \text{otherwise} \end{cases}$$

$$x_{ij} := \begin{cases} 1, & \text{if client } i \text{ is served by depot } j \\ 0, & \text{otherwise} \end{cases}$$

   - Constraints:

$$\sum_{j=1}^{n} x_{ij} = 1 \ \forall i \in [m] \text{ (each client is served)}$$

$$\sum_{i=1}^{m} x_{ij} \leq m y_j \ \forall j \in [n] \text{ (each depot can serve at most } m \text{ clients provided it is open)}$$

$$x_{ij} \in \{0,1\} \ \forall i \in [m], j \in [n]$$

$$y_j \in \{0,1\} \ \forall j \in [n]$$

   - Objective: min $\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{j=1}^{n} o_j y_j$

   Summarizing, we have the following IP formulation for the facility location problem:

$$\min \ \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{j=1}^{n} o_j y_j$$

$$\sum_{j=1}^{n} x_{ij} = 1, \ \forall i \in [m]$$

$$\sum_{i=1}^{m} x_{ij} \leq m y_j \ \forall j \in [n]$$

$$x_{ij} \in \{0,1\} \ \forall i \in [m], j \in [n]$$

$$y_j \in \{0,1\} \ \forall j \in [n]$$

   Note that the above formulation is concise - the size of the formulation is polynomial in the size of the input.

1.3.1 Integer Non-Linear Programs vs Integer Linear Programs

Non-Linear Programs and *Integer Linear Programs* are important topics in optimization that aim to go beyond Linear Programs. Both formulations are difficult to solve for several reasons. Naturally, the combination of them, namely Integer Non-Linear Programs are the most difficult to solve. For this reason, if you intend to formulate a problem with integer variables, it is better to have an integer *linear* programming formulation than an integer *non-linear* programming formulation.

In this course, we will focus only on Integer Linear Programs. The term 'linear' here refers to linearity in constraints and objective. For the sake of brevity, we will simply use Integer Programs or IPs to refer to Integer Linear Programs.

1.4 IP: Standard form

A "standard form" of a formulation allows one to design solution techniques (i.e., algorithms) for that form. Other forms of the formulation can also be solved using the solution technique by bringing them to the standard form. To state the standard form of IPs, we begin by recalling the standard form of linear programs:

Linear Program (LP):

$$\begin{aligned} \max\ & c^T x \\ Ax &\leq b \\ x &\geq 0 \end{aligned}$$

Here the input consists of the constraint matrix $A \in \mathbb{R}^{m \times n}$, the RHS vector $b \in \mathbb{R}^m$ and an objective vector $c \in \mathbb{R}^n$. We recall that $x \in \mathbb{R}^n$ is the variable vector.

Among IPs, we have several types based on whether we have purely integer variables or a mix of integer and real variables or binary integer variables. The standard forms of these are as follows:

Mixed IP (MIP): Here we have both integral as well as real valued variables.

$$\begin{aligned} \max\ & c^T x + h^T y \\ \text{s.t. } & Ax + By \leq b \\ & x \geq 0 \\ & y \geq 0 \\ & y \text{ integers} \end{aligned}$$

Here, the input consists of constraint matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{m \times p}$, the RHS vector $b \in \mathbb{R}^m$ and objective vectors $c \in \mathbb{R}^n$ and $h \in \mathbb{R}^p$.

Integer Program (IP): Here we have only integral variables. Thus, IPs are special cases of MIPs.

$$\begin{aligned} \max\ & c^T x \\ Ax &\leq b \\ x &\geq 0 \\ x &\text{ integers} \end{aligned}$$

Binary IP (BIP): Here, all variables are required to take binary values, i.e., 0 or 1. In many applications, such variables are used to represent logical relationships.

$$\begin{aligned} \max\ & c^T x \\ & Ax \leq b \\ & x \in \{0,1\}^n \end{aligned}$$

Exercise. BIPs are special cases of IPs.

We recall that binary variables are useful to model logical relationships. Here are some examples:

1. Suppose at most k among m items can be chosen. This is represented by the constraint $\sum_{i=1}^{m} x_i \leq k$ with binary variables $x_i \in \{0,1\}\ \forall\ i \in [m]$.
2. Suppose we have m constraints of the form $f_i(x_1, \ldots, x_n) \leq d_i$ for $i \in [m]$ and our goal is to pick a point $(x_1, \ldots, x_n)$ that satisfies at least k among them. This is represented using *big-M* and auxiliary binary variables $y_1, \ldots, y_m$ as

$$\begin{aligned} f_1(x_1, \ldots, x_n) &\leq d_1 + My_1 \\ &\vdots \\ f_m(x_1, \ldots, x_n) &\leq d_m + My_m \\ \sum_{i=1}^{m} y_i &\leq m - k \\ y_i &\in \{0,1\}\ \forall\ i \in [m] \end{aligned}$$

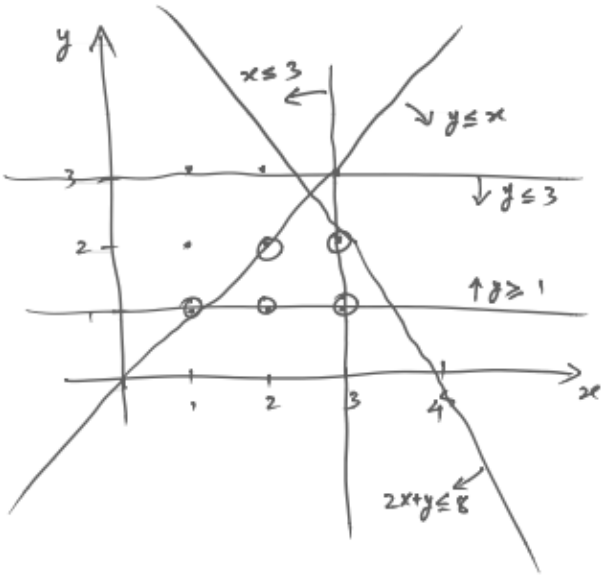
   In the above formulation, the constant M should be treated as infinity. In practice, i.e., while solving IPs, M is set to be the largest possible integer that your computer can handle.

1.4.1 An example with a geometric perspective

Consider the following IP in two variables:

$$\begin{aligned} \max\ & x + 2y \\ \text{s.t. } & 2x + y \leq 8 \\ & y - x \leq 0 \\ & x \leq 3 \\ & 1 \leq y \leq 3 \\ & y \text{ integers} \\ & x \text{ integers} \end{aligned}$$

We can plot the set of feasible points of this problem in a 2-dimensional figure as shown below (the two variables along the two axes with the feasible points circled):



We may solve such 2-variable IPs by computing the objective value of the feasible points and picking the best (we will care about efficiency later).

Feasible points	Objective value	
$(1,1)$	3	
$(2,1)$	4	
$(3,1)$	5	
$(2,2)$	6	
$(3,2)$	7	$\leftarrow$ Optimum

This geometric perspective will be very useful in understanding and solving IPs.

1.5 Comparing IP formulations

We have seen how to formulate some discrete optimization problems as IPs. Are these the only possible IP formulations of these problems? Indeed not. Consider the Facility Location Problem. The constraint $\sum_{i=1}^{m} x_{ij} \leq my_j$ formulates the logical relationship that if x_{ij} is positive for some client $i \in [n]$, then y_j should be one. This relationship can also be formulated as

$$x_{ij} \leq y_j \ \forall i \in [m], j \in [n].$$

This gives the following alternative formulation for the Facility Location Problem:

$$\begin{aligned}
\min \ & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{j=1}^{n} o_j y_j \\
\sum_{j=1}^{n} x_{ij} &= 1, \ \forall i \in [m] \\
x_{ij} &\leq y_j \ \forall i \in [m], j \in [n] \\
x_{ij} &\in \{0,1\} \ \forall i \in [m], j \in [n] \\
y_j &\in \{0,1\} \ \forall j \in [n]
\end{aligned}$$

Thus, IP formulations are not unique.

If there are multiple IP formulations for a problem, then which one is better? From an algorithmic perspective, a formulation would be "ideal" if we can solve it efficiently. Before we attempt to compare formulations let us first formalize the notion of "formulations".

Definition 1. A *polyhedron* is a subset of $\mathbb{R}^n$ satisfying a finite number of linear inequalities.

Example: $P := \{x \in \mathbb{R}^n : Ax \leq b\}$ for some constraint matrix $A \in \mathbb{R}^{m \times n}$ and RHS vector $b \in \mathbb{R}^m$.

In order to formulate a discrete optimization problem/COP as an IP, it is helpful to think of a discrete optimization problem as $\max\{c^T x : x \in S \subseteq \mathbb{Z}^n\}$ for some set S. Here, the set S encodes the collection of feasible solutions of the problem. With this viewpoint, we have the following definition of a formulation:

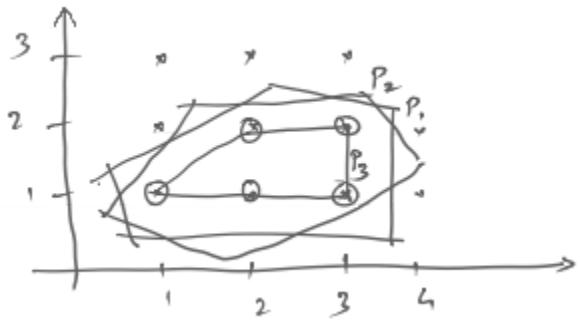
Definition 2. A polyhedron $P \subseteq \mathbb{R}^n$ is a *formulation* for a set $S \subseteq \mathbb{Z}^n$ if $S = P \cap \mathbb{Z}^n$.

Now that we have rigorously defined formulations, go back and verify that the the IPs that we formulated for the knapsack problem and the assignment problem are indeed *correct* formulations for the respective problems. I.e., verify that the set of feasible solutions are exactly the points in $P \cap \mathbb{Z}^n$.

Exercise. Show that our IP formulations for Knapsack, Assignment, TSP, and Facility Location are indeed correct formulations.

1.5.1 A discrete optimization problem could have multiple formulations

Consider a set $S = \{(1,1), (2,1), (3,1), (2,2), (3,2)\}$. What are the possible formulations for this set? The figure below shows three possible formulations for the set S.



In fact there are infinitely many possible formulations for S. Given two formulations P_1 and P_2 for a problem, which one is better? How do we compare them? It should be intuitively clear that P_3 is a better formulation than P_1 or P_2 in the above example. This suggests the following formal way to compare two formulations:

Definition 3. Given $S \subseteq \mathbb{Z}^n$ and two formulations P_1 and P_2 for S, we define P_1 to be a *better formulation* than P_2 if $P_1 \subset P_2$.

We will later see ideal/best formulations for some discrete optimization problems.