CS491I: Approximation Algorithms
Chapter : Integer Programming (Modeling)
Lecture Notes

Habib Ammari

February 13th and 15th, 2001

1 Introduction

A *linear program* is defined by $\mathbf{A}$, $\vec{b}$, and $\vec{c}$ as follows:

$$\begin{cases} max \;\; \vec{c}\vec{x} \\ \mathbf{A}\vec{x} \leq \vec{b} \\ \vec{x} \geq \vec{0} \end{cases}$$

An *integer program* is identical to a *linear program* with the restriction that the solutions must be *integer* valued.

$$\begin{cases} max \;\; \vec{c}\vec{x} \\ \mathbf{A}\vec{x} \leq \vec{b} \\ \vec{x} \geq \vec{0} \\ \vec{x} : integer \end{cases}$$

Example. Let's consider the following program:

$$\begin{cases} max \;\; z = -x_1 + 4x_2 \\ s.t. \\ -10x_1 + 20x_2 \leq 22 \\ 5x_1 + 10x_2 \leq 49 \\ 8x_1 - x_2 \leq 36 \\ \vec{x} \geq \vec{0} \\ \vec{x} : integer \end{cases}$$

□

Pictorially, the graphical solution gives us $(3.8, 3)$ as optimum. The first coordinate is not integer whereas the second is integer. We may get just integer by *truncating*, $(3, 3)$, or *rounding*, $(4, 3)$. However, in either case some constraints are violated, and, so, the obtained solutions are no longer feasible. If the solution were $(3, 3)$, then the first constraint is no longer satisfied $(-10 \times 3 + 20 \times 3 > 22)$. If the solution were $(4, 3)$, then the second constraint does not hold $(5 \times 4 + 10 \times 3 > 49)$.

In the feasibility region only discrete points are feasible. But the line that joins two feasible solutions is not necessarily feasible. So here we are dealing with a *discrete set* and not a *convex set* where the optimum is on a face.

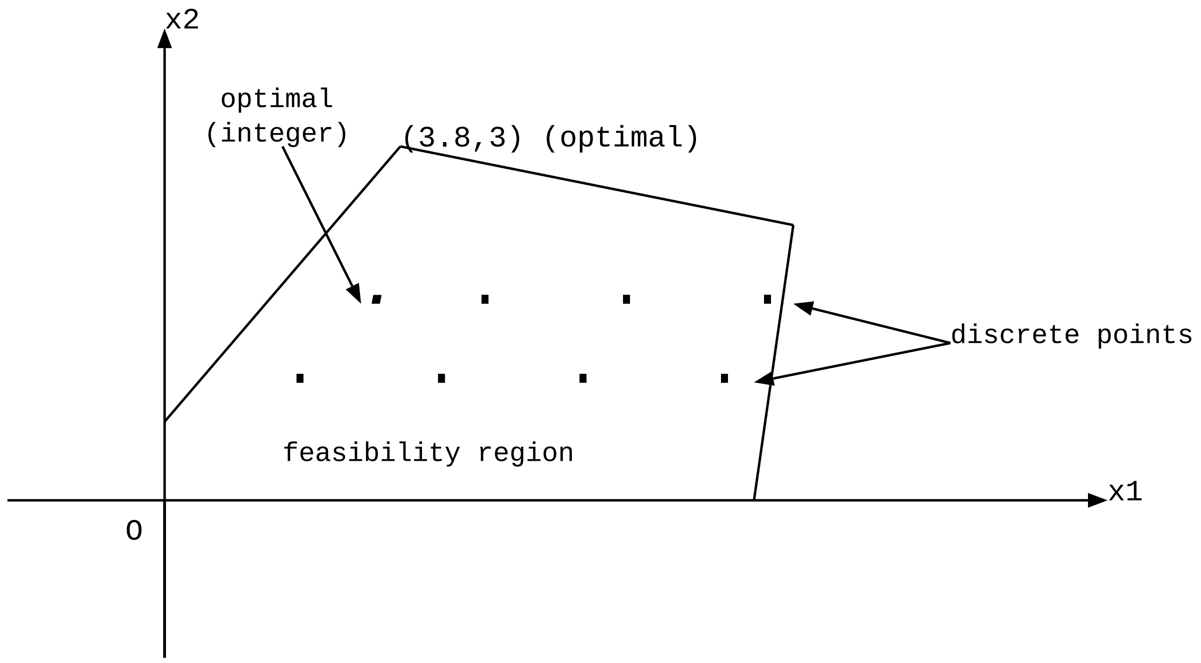

Figure 1: Graphical Solution of the Problem

2 Integer Programs

Theorem 1 *The class of integer programs can be expressed by restricting the variable values to* $\{0, 1\}$.

Proof. *Setp 1:* Assume that the polyhedron $A\vec{x} \leq \vec{b}$ is bounded.
Find *min*, *max* for all x_i over the polyhedron which may not be integer nor $0, 1$.
Suppose that x_i is required to take a value between 7 and 24. In fact x_i can be expressed as $x_i^1.16 + x_i^2.8 + x_i^3.4 + x_i^4.2 + x_i^5.1$, where $x_i^j \in \{0, 1\}$ and we add the following two constraints: $7 \leq x_i^1.16 + x_i^2.8 + x_i^3.4 + x_i^4.2 + x_i^5.1$ and $x_i^1.16 + x_i^2.8 + x_i^3.4 + x_i^4.2 + x_i^5.1 \leq 24$. In general, if we assume the maximal value (upper bound) is M, then x_i can be defined as: $2^0 x_i^1 + 2^1 x_i^2 + \ldots + 2^{logM} x_i^n$, where x_i^j are restricted to 0 and 1. However, we increase the number of variables by $nlogM$.
qed

2.1 Combinatorial Optimization (CP)

Let $M = \{1, .., m\}$ be a finite set and $\vec{c} = (c_1, ..., c_m$ be an m-vector where c_i is the cost associated with element i, $i = 1, ..., m$. Suppose we are given a collection of subsets F of M, i.e., $F = \{\{M_i\}_{i=1,...,n}\}$. For $F' \subseteq F$ let $c(F') = \sum_{j \in F'} c_j$. Then CP can be formulated as follows:

$$\Big\{ \; min\{c(F') : F' \subseteq F\}$$

2.2 Knapsack Problem

This problem is characterized by n objects $O_1, O_2, O_3, ..., O_n$ where each O_i is associated with a *profit* p_i and a *weight* w_i. We want to choose a subset of objects to maximize the profit without violating the fact that the *knapsack* has a limit weight W. To formulate this problem as an integer program, we introduce a binary *indicator variable* x_i. So picking or not picking an object depends on x_i such that:

$$x_i = \begin{cases} 1 & \text{if object } O_i \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

- The **objective function** we want to maximize is $\sum_{i=1}^{n} p_i x_i$.
- The only **constraint** deals with limit weight W of the *knapsack*. The subset of all selected objects must respect this upper bound, i.e., their weight must not excede W. This can be expressed by $\sum_{i=1}^{n} w_i x_i \leq W$.

The whole integer program that models this problem is as follows:

$$\begin{cases} max \sum_{i=1}^{n} p_i x_i \\ \sum_{i=1}^{n} w_i x_i \leq W \\ x_i \in \{0,1\} \end{cases}$$

2.3 Assignment Problem

Given m jobs and n people, we define a profit c_{ij} if person i does job j. We want to assign these jobs to these people to maximize the total profit under two constraints: one person does at most one job (first constraint), and one job is done by exactly one person (second constraint). To formulate the problem as an integer program, we introduce an indicator variable x_{ij} such that:

$$x_{ij} = \begin{cases} 1 & \text{if person } i \text{ does job } j \\ 0 & \text{otherwise} \end{cases}$$

- The **objective function** we want to maximize is $\sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$.
- To satisfy the **first constraint**, at most one x_{ij} can be non zero value for a given person i and all the jobs j, for all j in $[1, m]$. This constraint can be written as $\sum_{j=1}^{m} x_{ij} \leq 1, \forall i \in [1, n]$.
- The **second constraint** holds if and only if just one x_{ij} is equal to 1 for a given job j and all the persons i, for all i in $[1, n]$. This constraint can be expressed by $\sum_{i=1}^{n} x_{ij} = 1, \forall j \in [1, m]$.

Hence the following formulation:

$$\begin{cases} max \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} \\ \sum_{j=1}^{m} x_{ij} \leq 1, i = 1, ..., n \\ \sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., m \\ x_{ij} \in \{0,1\} \end{cases}$$

2.4 Another Assignment Problem

Given $2n$ students and n dorm rooms, we define c_{ij} as the affinity of student i for student j (the corresponding matrix is symmetric, so we consider $j > i$). We want to maximize the satisfaction: how can we pair students to reach this goal? We introduce an indicator variable x_{ij} such that:

$$x_{ij}(i \neq j) = \begin{cases} 1 & \text{if student } i \text{ is paired with student } j \\ 0 & \text{otherwise} \end{cases}$$

- The **objective function** we want to maximize is $\sum_{i=1}^{2n-1} \sum_{j=i+1}^{2n} c_{ij}x_{ij}$. For every student i, except the last one ($i \in [1, 2n-1]$), we check whether one of the rest of the student, say j ($j \in [i+1, 2n]$, can be paired with i. The index j in the objective function varies from $i+1$ to $2n$ and not from 1 to $2n$ in order to avoid counting $c_{ij}x_{ij}$ twice given the property of symmetry inherent to this problem (if i is paired with j, then necessarily j is paired with i).

- The only **constraint** that must hold is that a given student i can be paired with only one student j. This is can be written as $\sum_{k<i} x_{ki} + \sum_{j>i} x_{ij} = 1, i = 1, ..., 2n$.

The problem is formulated as an integer program as follows:

$$\begin{cases} max \sum_{i=1}^{2n-1} \sum_{j=i+1}^{2n} c_{ij}x_{ij} \\ s.t. \\ \sum_{k<i} x_{ki} + \sum_{j>i} x_{ij} = 1, i = 1, ..., 2n \\ \vec{x} \in \{0,1\}^{n(2n-1)} \\ x_{ij} \in \{0,1\} \end{cases}$$

2.5 Covering, Packing and Partitioning

Suppose we are given a ground set $M = \{1, ..., m\}$. Then we define $G = \{\{M_i\}\}, i = 1, ..., n$ as a collection of subsets of M.

Definition 1 *A subset $F \subseteq G$ is said to* cover M *if* $\bigcup_{M_i \in F} M_i = M$.

Definition 2 *A subset $F \subseteq G$ is said to* pack M *if* $M_i \cap M_j = \emptyset$*, for* $M_i, M_j \in F$.

Definition 3 *A subset $F \subseteq G$ is said to* partition M *if it both covers and packs* M.

Let a_{ij} an indicator variable defined as follows:

$$a_{ij} = \begin{cases} 1 & \text{if element } i \text{ belongs to the set } M_j \\ 0 & \text{otherwise} \end{cases}$$

The problem can be formulated as follows:

- **Covering:** $\mathbf{A}\vec{x} \geq \vec{1}, \vec{x} \in \{0,1\}^n$.
  This means that we have exactly m constraints and that for any constraint $i, i = 1, ..., m$ there is at least one $a_{ij} \neq 0$. This means that element $i, i = 1, ..., m$ is included in at least one set M_j, and, so, the set M is entirely covered.

- **Packing:** $\mathbf{A}\vec{X} \leq \vec{1}$.
  Given that $\vec{x} \in \{0,1\}$, we may find at most one non zero a_{ij} per constraint, which means that element i belongs to at most one set M_j. So, all the M_j are mutually disjoint.

- **Partitioning: $\mathbf{A}\vec{X} = \vec{1}$.**
  In order for $F \subseteq G$ to both cover and pack M, both $A\vec{x} \geq \vec{1}$ and $A\vec{x} \leq \vec{1}$ must be satisfied. Thus, we have equality, i.e., $A\vec{x} = \vec{1}$.

2.6 Logical Modeling

Now we consider logical assertions that we want to model. Assume we are given a set of events x, and let $x_1, x_2 \in x$.

- Suppose you want to model the fact that x_1 happens *if and only if* x_2 happens. This means that if x_1 takes place, then x_2 will take place too, and vice versa. On the other hand, if x_1 cannot happen, then x_2 will not happen, and vice versa. Thus, we have $x_1 = x_2$. In terms of integer programming modeling, x_1 and x_2 have the same value at the same time either 0 or 1.

- Assume we want to model the fact that x_2 holds *only if* x_1 holds. This means that if $x_2 = 1$, then necessarily $x_1 = 1$. This can be expressed as $x_2 - x_1 \leq 0$.

- Assume we have a real variable $y \in R^+$. Further y can take any value in $[0, u]$. We want to model that if $x_1 = 0$ then $y = 0$, and if $x_1 = 1$ then $y \in [0, u]$. This is can be modeled by $y - ux_1 \leq 0$.

2.7 Facility Location Problem

Given two sets $N = \{1, ..., n\}$ of sites, and $I = \{1, ..., m\}$ of clients. Associated with each of the sites i is a cost c_i for opening i.
Let h_{ij} be the cost of satisfying the demand of client i by site j, and y_{ij} (real number) the fraction of client i's demand satisfied by site j. We want to select which sites to open to minimize the cost of satisfying all the client demands. An indicator variable x_i is introduced, and is defined as follows:

$$x_i = \begin{cases} 1 & \text{if it has been decided to open the site } i \\ 0 & \text{otherwise} \end{cases}$$

- As we can see, it is to minimize the cost of opening sites $i, i = 1, ..., n$ which is modeled by the first part of the objective function, i.e., $\sum_{i=1}^{n} c_i x_i$, as well as the cost of transportation of the products requested by the clients which is modeled by the second part of the objective function, i.e., $\sum_{i=1}^{n} \sum_{j=1}^{m} y_{ij} h_{ij}$ where h_{ij} represents the transportation cost from site j to client i. So the **objective function** to minimize is $\sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{m} y_{ij} h_{ij}$.

- This problem assumes that all clients' demands are normalized to be 1, This assumption is modeled by the **first constraint**, i.e., $\sum_{j=1}^{n} y_{ij} = 1, i = 1, ..., m$.

- We also suppose that any site can produce unlimited quantity of products. This is modeled by the **second constraint**, i.e., $y_{ij} - x_j \leq 0$. So if $x_j = 1$, then y_{ij} may take any value in $[0, 1]$, and if x_j equals 0, then necessarily y_{ij} equals 0.

The entire integer program that models this problems is given below:

$$\begin{cases} min \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{m} y_{ij} h_{ij} \\ \sum_{j=1}^{n} y_{ij} = 1, i = 1, ..., m \\ y_{ij} - x_j \leq 0 \end{cases}$$

However the problem can be modeled another way if we ruled out these non realistic assumptions.

- The objective function to minimize remains the same as for the first version of the problem.
- If we assume b_i represents the demands of client i, then the **first constraint** becomes $\sum_{j=1}^{n} y_{ij} = b_i$, for any client $i, i = 1, ..., m$.
- If u_j is the fixed capacity of site j, which means that a site j can produce at most u_j, then the **second constraint** is $\sum_{i=1}^{m} y_{ij} - u_i x_j \leq 0$ for all j. In fact if $x_j = 1$, then $\sum_{i=1}^{m} \leq u_j$, u_j is an upper bound.

Thus the new modeling of the problem as an integer program is:

$$\begin{cases} min \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{m} y_{ij} h_{ij} \\ \sum_{j=1}^{n} y_{ij} = b_i, i = 1, ..., m \\ \sum_{i=1}^{m} y_{ij} - u_j x_j \leq 0, j = 1, ..., n \end{cases}$$

2.8 Fixed-Charge Network Problem

We have some source sites with their positive demands. So every site i is associated with its demand d_i. A fixed-charge network problem can be modelled by a graph $G = (V, E)$ in which nodes V, where $\mid V \mid = n$, represent sites along with their demands, and arcs E the cost of shipping from one site to another.
In order for the problem to be feasible, we must have $\sum_{i=1}^{n} d_i = 0$. In addition, we have the following parameters:
h_{ij}: cost of unit flow from node i to node j, and
y_{ij}: amount of flow from node i to node j, $y_{ij} \geq 0$.

2.8.1 Linear Program

This problem, which is called *network flow problem*, is a linear programming problem.

- We want to minimize the total cost of transportation. The **objective function** is, thus, $\sum_{i=1}^{n} \sum_{j=1}^{n} h_{ij} y_{ij}$.
- The only **constraint**, which is called *flow conservation constraint*, means that the sum of all the incoming from other sites j to the site i, which is a positive quantity, and the outgoing from i to other sites of the network, which is a negative quatity, must be equal to the demand of site i. This can be expressed as $\sum_{j=1}^{n} y_{ji} - \sum_{j=1}^{n} y_{ij} = d_i, i = 1, ..., n$.

Hence the following linear program that models this problem:

$$\begin{cases} min \sum_{i=1}^{n} \sum_{j=1}^{n} h_{ij} y_{ij} \\ \sum_{j=1}^{n} y_{ji} - \sum_{j=1}^{n} y_{ij} = d_i, i = 1, ..., n \end{cases}$$

2.8.2 Integer Program

Let's now consider another version of the problem. Assume there is a fixed cost c_{ij} between node i and node j if $y_{ij} > 0$. We also assume that every arc (i, j) has a maximal capacity u_{ij}. In order to model this problem as an integer program, we introduce an indicator variable x_{ij} such that:

$$x_{ij} = \begin{cases} 1 & \text{if } y_{ij} > 0 \\ 0 & \text{if there is no flow from } i \text{ to } j \end{cases}$$

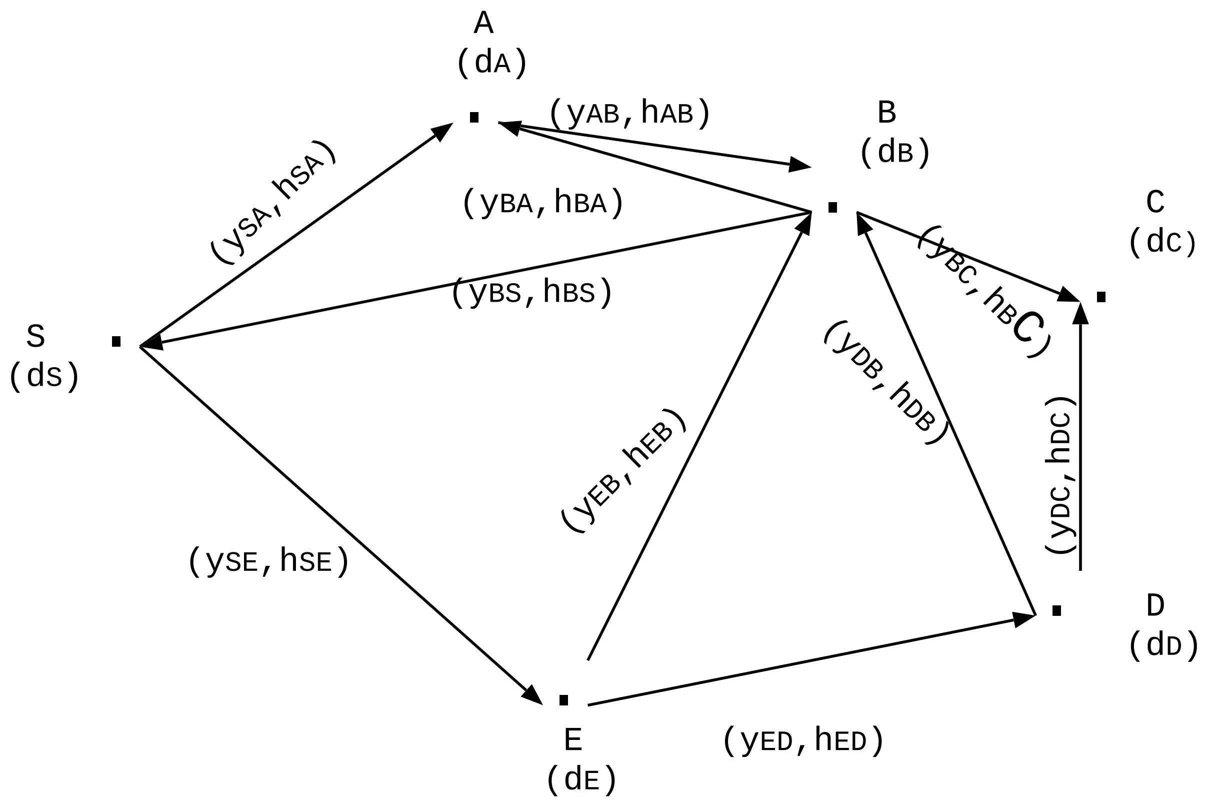

Figure 2: Fixed-Charge Problem

- To solve this problem, we want to minimize the total cost related to the total fixed cost, i.e., $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$, as well as the total transportation cost, i.e., $\sum_{i=1}^{n}\sum_{j=1}^{n} h_{ij}y_j$. So, the **objective function** that we want to monimize is $\sum_{i=1}^{n}\sum_{j=1}^{n}(h_{ij}y_{ij} + c_{ij}x_{ij})$.
- Given that u_{ij} is the maximal capacity of an arc (i, j), then we must have $y_{ij} \leq u_{ij}$ if $x_{ij} = 1$. Also if $x_{ij} = 0$, then y_{ij} should be equal to 0. So the **first constraint** can be expressed as $y_{ij} - u_{ij}x_{ij} \leq 0, i = 1, ..., n, j = 1, ..., n$.
- In addition, for any node i the sum of the amount of flow going out of i to all other node j over the network, and its demand d_i must be equal to the amount of flow coming into i from the other nodes j. This **second constraint** can be written as $\sum_{j=1}^{n} y_{ij} + d_i = \sum_{j=1}^{n} y_{ji}$.

The entire modeling of this problem as an integer program can be done as follows:

$$\begin{cases} min \sum_{i=1}^{n}\sum_{j=1}^{n}(h_{ij}y_{ij} + c_{ij}x_{ij}) \\ y_{ij} - u_{ij}x_{ij} \leq 0, i = 1, ..., n, j = 1, ..., n \\ y_{ij} \geq 0, i = 1, ..., n, j = 1, ..., n \\ \sum_{j=1}^{n} y_{ij} + d_i = \sum_{j=1}^{n} y_{ji}, i = 1, ..., n \\ x_{ij} \in \{0, 1\}, i = 1, ..., n, j = 1, ..., n \end{cases}$$

2.9 Travelling Salesman

This problem can be represented by a graph $G = (V, E)$ where nodes are cities, $\mid V \mid= n$, and arcs the costs for a salesman to go from one city to another (see Figure 3). Let c_{ij} be the cost to go from city i to city j. We define an indicator variable x_{ij} such that:

$$x_{ij} = \begin{cases} 1 & \text{if city } i \text{ is the immediate predecessor of city } j \text{ in the tour} \\ 0 & \text{otherwise} \end{cases}$$

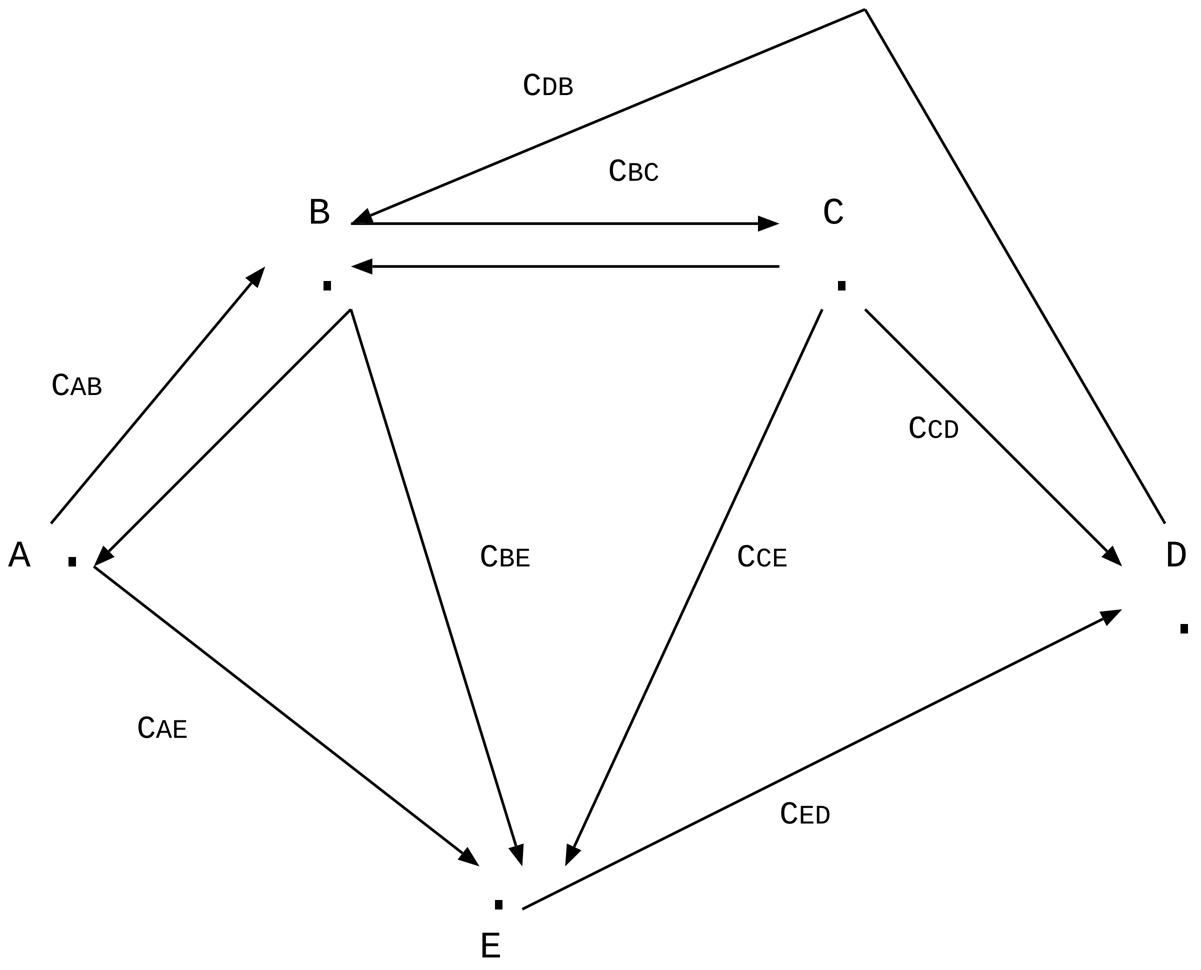

Figure 3: Traveling Salesman Problem

- We want to find the optimum tour of the cities which corresponds to minimize the cost of visiting all the cities exactly one time. The objective function that we want to minimize is $\sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}$.
- The **first constraint** says that at most one arc can come in node i. This can be expressed by $\sum_{j \in V} x_{ji} = 1, i \in V$.
- The **second constraint** tells us that from node i it is possible to go out to at most one node. This can be written as $\sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n$.

The complete formulation of the problem as an integer program is as follows:

$$\begin{cases} min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \\ \sum_{j \in V} x_{ji} = 1, i \in V \\ \sum_{j \in V} x_{ij} = 1, i \in V \end{cases}$$

But this formulation is incomplete since we may fullfil these two equations by ending up with some situation in which we have for instance two disjoint cycles (see Figure 4). So we need to add more constraints to avoid falling in this kind of situation.

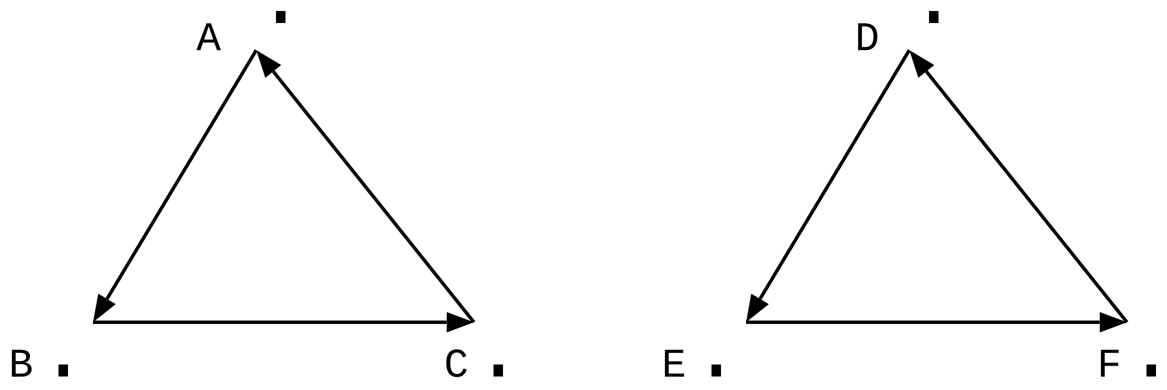

Figure 4: Two Disjoint Cycles

For each $U \subset V$, we have:

$$\begin{cases} 2 \leq \mid U \mid \leq \mid V \mid -2 \\ \sum_{i \in U, j \in V/U} x_{ij} \geq 1 \end{cases}$$

Thus the modeling of the problem as an integer program is given below:

$$\begin{cases} min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ \sum_{j=1}^{n} x_{ji} = 1, i = 1, ..., n \\ \sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n \\ 2 \leq \mid U \mid \leq \mid V \mid -2, U \subset V \\ \sum_{i \in U, j \in V/U} x_{ij} \geq 1, U \subset V \end{cases}$$

Remark. We have at most 2^n constraints, which is an exponential number. But the above formulation avoids us to specify all these constraints. In fact, in actual implementations, constraints are added one at a time. $\square$

2.10 Modeling Non-Linear Functions

Assume we have a non-linear function, which is given by $(a_i, f(a_i)), i = 1, ..., r$, to optimize over the polyhedron $A\vec{x} \leq \vec{b}$ (see Figure 5). Any point y on this function is expressed as a *convex combination* of other points:

$$\begin{cases} y = \sum_{i=1}^{r} \lambda_i a_i \\ \sum_{i=1}^{r} \lambda_i = 1 \\ \lambda_i \geq 0 \end{cases}$$

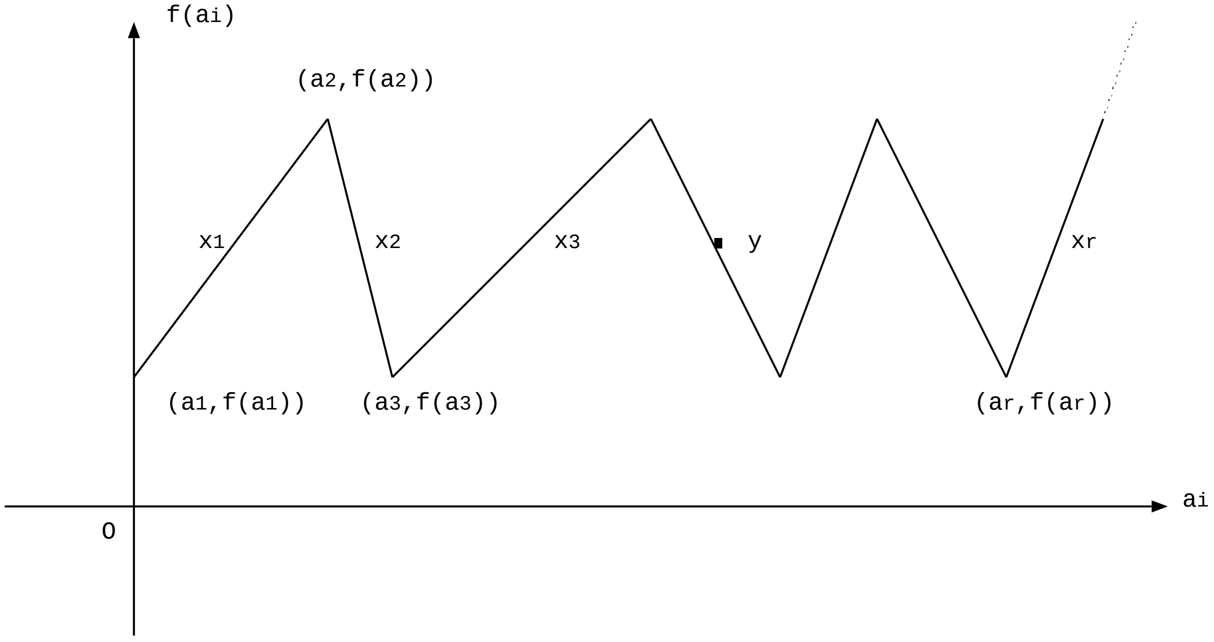

Figure 5: Non-Linear Function Problem

Using this formulation, y may be out of any broken segment of the function. So we introduce an indicator variable x_i such that:

$$x_i = \begin{cases} 1 & \text{if the point } y \text{ is on the broken segment } i \\ 0 & \text{otherwise} \end{cases}$$

In addition, we add more constraints to force y to be on one of the broken segments:

- First of all, only one segment can be picked. This constraint can be written as $\sum_{i=1}^{r-1} = 1$.
- We enforce the λ_i's to be positive and match them to the x_i's. Thus the following constraints.
  1. $\lambda_1 \leq x_1$: if x_1 equals 0, then λ_1 equals 0.
  2. λ_i is positive if x_{i-1} is picked or x_i is picked. This constraint can be expressed by $\lambda_i \leq x_{i-1} + x_i$.
  3. Finally, if x_{r-1} equals 0, then λ_r equals 0 too.

The modeling of this problem as an integer program is as follows:

$$\begin{cases} y = \sum_{i=1}^{r} \lambda_i a_i \\ \sum_{i=1}^{n} \lambda_i = 1 \\ \lambda_i \geq 0 \lambda_1 \leq x_1 \\ \lambda_i \leq x_{i-1} + x_i, i = 2, ..., n-1 \\ \lambda_n \leq x_n \\ \sum_{i=1}^{n-1} x_i = 1 \end{cases}$$

2.11 Disjunction of Constraints

- Let's consider the following equation that aims to find the minimum among two variables u_1 and u_2: $y = min(u_1, u_2)$. It can be expressed using *and/or* operators: $(y \leq u_1$ and $y \leq u_2)$ and $(y \geq u_1$ or $y \geq u_2)$.

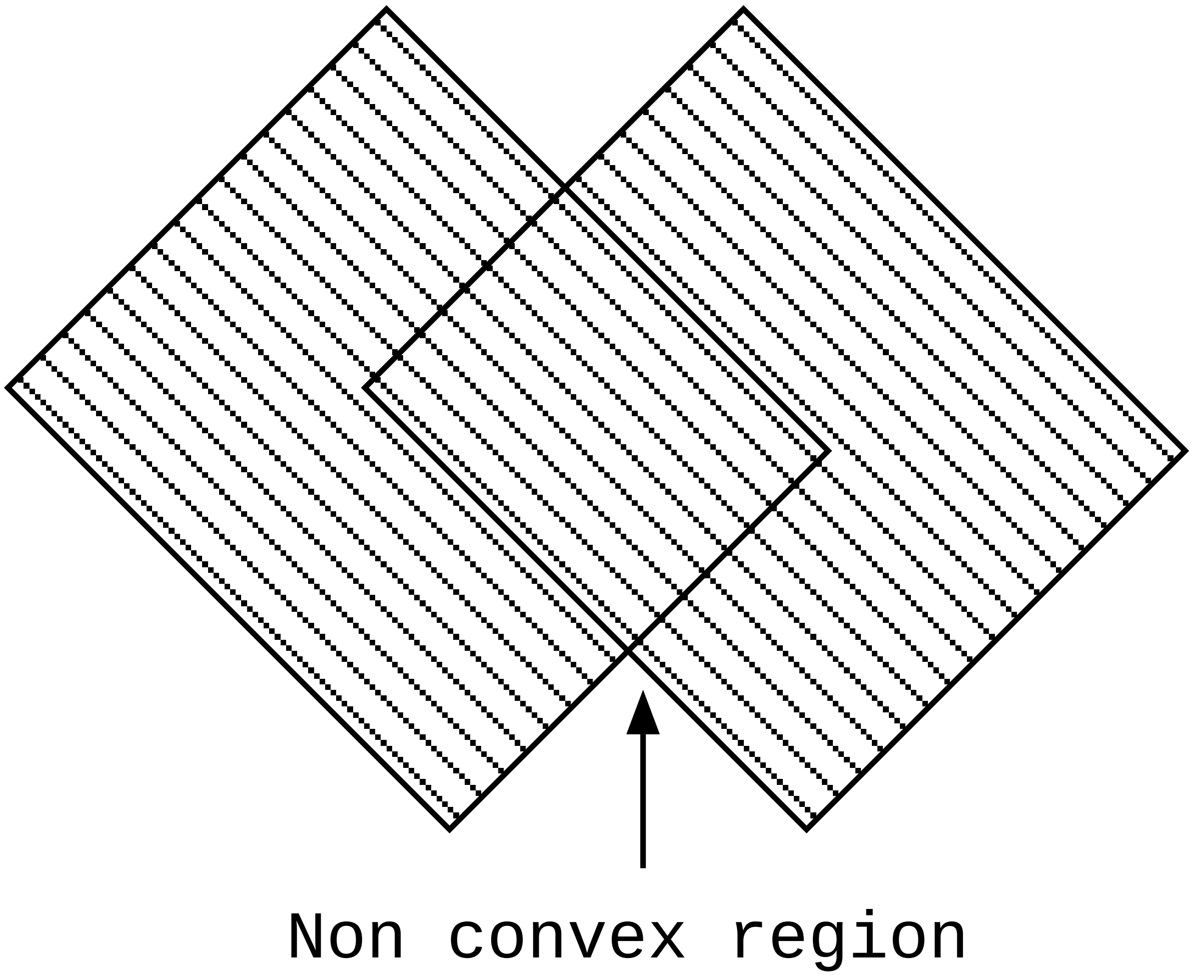

Figure 6: Non Convex Region

- In general we want to satisfy a subset of constraint. Assume we have m constraints

$$\begin{cases} a_{11}y_1 + ... + a_{1n}y_n \leq b_1 \\ ... \\ a_{m1}y_1 + ... + a_{mn}y_n \leq b_m \end{cases}$$

and we want to satisfy at least k out of them. To do so we introduce an indicator variable x_i such that:

$$x_i = \begin{cases} 1 & \text{if constraint } i \text{ holds} \\ 0 & \text{constraint } i \text{ is useless} \end{cases}$$

To the right side of each constraint $i, i = 1, ..., m$, we add the quantity $M(1-x_i)$, where M is a big value. If for instance some x_j, say x_1, is equal to 1, then $M(1-x_1) = 0$ and, thus, the corresponding constraint becomes $a_{11}y_1 + ... + a_{1n}y_n \leq b_1$ and is binding. However, if x_1 is equal to 0, then the corresponding constraint is useless because $b_1 + M >> b_1$.

Thus we have the following formulation:

$$\begin{cases} a_{11}y_1 + ... + a_{1n}y_n \leq b_1 + M(1-x_1) \\ ... \\ a_{1m}y_1 + ... + a_{mn}y_n \leq b_m + M(1-x_m) \\ \sum_{i=1}^{m} x_i \geq k \end{cases}$$

2.12 Scheduling Problem

Assume we have n jobs, and each job i is associated with three parameters:
γ_i: release time (ready time)
d_i: deadline
p_i: processing time.
We want to find the start time s_i for each job i:

- The bf first constraint says that the any job i must start after its release time. This can be written as $s_i \geq \gamma_i$.
- The **second constraint** says that each job must be done by its deadline. This can be expressed by $s_i + p_i \leq d_i$.

We, thus, have the following modeling of the problem:

$$\begin{cases} s_i \geq \gamma_i \\ s_i + p_i \leq d_i \end{cases}$$

If we try to solve this problem as a linear program after writing all these constraints, we may have a preemption problem.

Example.

T\ *J*	J_1	J_2
Release	0	0
Deadline	4	4
Processing	3	3

If we solve this problem, we find: $s_1 = s_2 = 0$. The crucial part is how to schedule (order) these jobs. For each pair of jobs (J_i, J_j), we have:

- $s_i + p_i \leq s_j$: job J_i can start only when job J_j is done, or
- $s_j + p_j \leq s_i$: job J_j can start only when job J_i is done

This means no two jobs can be done at the same time. To model this problem as an integer program, we introduce an indicator variable x_{ij} such that:

$$x_{ij} = \begin{cases} 1 & \text{if } j \text{ is scheduled before } i \\ 0 & \text{otherwise} \end{cases}$$

We now consider the new constraints, where we add the quantity $x_{ij}M$ to the first previous constraint:

- $s_i + p_i \leq s_j + x_{ij}M$:

and the quantity $(1 - x_{ij})M$ to the second previous constraint:

- $s_j + p_j \leq s_i + (1 - x_{ij})M$:

If x_{ij} is equal to 0, then necessarily job i is done before job j starts, given the big value of M. On the other hand, if x_{ij} is equal to 1, in this case job j ends before job i begins. Thanks to this formulation, only one job executes at a time and the problem of preemption is avoided.

Hence the new modeling of the problem as an integer program:

$$\begin{cases} s_i + p_i \leq s_j + x_{ij}M \\ s_j + p_j \leq s_i + (1 - x_{ij})M \end{cases}$$

2.13 Vertex Cover

Given an arbitrary graph $G = (V, E)$, where $\mid V \mid= n$, we want to select a subset of vertices $V' \subseteq V$ such that each edge is covered. The problem can be solved by picking all the vertices, but we need to minimize the set V' where all the edges are still covered. We, thus, introduce an indicator variable x_i such that:

$$x_i = \begin{cases} 1 & \text{if vertex } i \text{ is picked} \\ 0 & \text{otherwise} \end{cases}$$

- The **objective function** that we want to minimize is $\sum_{i=1}^{n} x_i$.
- The only **constraint** that we want to satisfy is to take for each edge (i, j) at least one of the vertices so that edge is covered. This constraint can be expressed by $x_i + x_j \geq 1$, for each edge (i, j).

We obtain the following modeling of the problem:

$$\begin{cases} min \sum_{i=1}^{n} x_i \\ x_i + x_j \geq 1 \quad \text{for each edge } (i, j) \end{cases}$$