1 Linear Programming

Linear programming is a technique for solving optimization problems. It has the form:

\[ c^T x \rightarrow \text{minimize} \]

such that

\[ Ax = b \]
\[ x \geq 0 \]

where we are given a matrix \( A \) of size \( m \times n \) where \( m \) is the number of constraints and \( n \) is the number of variables. We are given \( m \)-vector \( b = (b_1, b_2, \ldots, b_m)^T \) and \( n \)-vector \( c = (c_1, c_2, \ldots, c_n)^T \).

Find an \( n \)-vector \( x = (x_1, x_2, \ldots, x_n)^T \) to minimize the

\[ c^T x \]

subject to constraints

\[ Ax = b \]
\[ x \geq 0 \]

Linear programming is a technique for finding an optimized feasible solution for an objective function while following the linear inequalities and linear equalities. The constraint in linear program forms what is called a polytope. Linear programming algorithm finds a point in feasible region, if there exist, on which the objective function has minimum (or maximum) value.

Solutions of LPs can be categorized into three useful types:

**Basic feasible solution**: Suppose the rows of \( A \) are linearly independent, i.e., \( m \leq n \) (otherwise we can delete redundant rows of \( A \)).

Now, a **Basic Solution** is defined as follows: Let \( S \subseteq [n] \) be such that columns of \( A_S \) are linearly independent and \( |S| = m \). Then \( A_S \) has size \( m \times m \) and rank \( m \). Then define

\[ x_S = (A_S)^{-1}b \]

and define \( x \) to be equal to \( x_S \) for the indices in \( S \) and 0 for all other indices. All such completed \( x \)'s are called basic solutions. Additionally if \( x_s \geq 0 \) the solution is called a basic feasible solution.

Of course, we set \( x_{[n] \setminus S} = 0 \).

**Theorem 1.1.** *If polytope is feasible then there exist a basic feasible solution*

**Theorem 1.2.** *If linear programming has optimal solution then there exist a feasible solution which is also a basic feasible solution, means optimal solution is also a basic feasible solution*

*Proof.* Proof was covered in Lecture 08.
Now we define the second and third kind of solutions, which take a geometric view point. **Definition 1**: \( x \in P \) is called a vertex/corner point, where \( P \) is the polytope.

iff \( \exists c \) such that (minimization problem)

\[
c^T x < c^T x'
\]

\( \forall x' \in P \) such that \( x \neq x' \)

**Definition 2** \( x \in P \) called extreme point if \( x' \) can not be written as

\[
\lambda x_1 + (1 - \lambda)x_2
\]

for \( x_1, x_2 \in P \)

**Definition 3** \( x \in P \) is basic feasible solotion if \( \exists , S \subseteq [n] , |S| = m \) \( A_S \) has rank \( m \)

\[
x_S = (A_S)^{-1}b \succeq 0
\]

\[
x_{[n]\setminus S} = 0
\]

**Theorem 1.3.** \( \text{corner} \iff \text{extreme} \iff \text{bfs} \)

**Proof.**  
1. \( x \) is corner \( \implies \) \( x \) is extreme

   lets prove by contradiction suppose \( x \) is not extreme

   \[
x = \lambda x_1 + (1 - \lambda)x_2
\]

   where \( x_1, x_2 \in p \)

   \[
c^T x = \lambda c^T x_1 + (1 - \lambda)c^T x_2
\]

   \[
   < \lambda c^T x + (1 - \lambda)c^T x \\
   = c^T x \implies \n
\]

2. \( x \) is extreme \( \implies \) \( x \) is basic feasible solution

   \( x \) is extreme \( \implies \) \( x \) is bfs \( s = j : x_j > 0 \) all entries in \( x \) are strictly positive

   we need to show that

   (a) \( |s| \leq m \)

   (b) \( A_s \) has rank \( m \)

   (c) \( x_s = (A_s)^{-1}b \)

   (d) \( x_{[n]\setminus S} = 0 \)

   **proving all above** Suppose that \( |s| > m \)

   We know that \( \text{rank}(A)=m \) which means columns of \( s \) are linearly independent \( \implies \exists w_s \neq 0 \)

   such that

   \[
   A_s w_s = 0
\]

   extended to \( A \) by setting \( w_j = 0 \) if \( j \not\in s \)

   \( A w = 0 w \neq 0 \)
\[ x_1 = x - \varepsilon w \text{ such that } Ax_1 = b \quad (Aw = 0) \]
\[ x_2 = x + \varepsilon w \text{ such that } Ax_2 = b \]
if \( \varepsilon \) is small enough \( x_1 \geq 0 \quad x_2 \geq 0 \)
same proof if \(|s| < m\) but column of \( A_s \) are linearly dependants
\[ |s| \leq m \text{ and } A_s \text{ linearly independent} \]
\[ s' = s \cup \text{some } m - |s| \text{ column such that } \text{rank}(s') = m \text{ and } |s'| = m \]
\[ x_s = (A_s)^{-1}b \text{ is the desired basic feasible solution here } x_s = x, \]
\( A_s \) has rank \( m \)
\[ \rightarrow X_s \text{ is the unique solution to } (A_s)^{-1}b \]

3. \( x \) is basic feasible solution(bfs) \( \implies \) \( x \) is corner

\[ x \text{ is bfs} \implies \exists S \quad A_s \text{ id full rank } |s| = m \quad x_s = (A_s)^{-1}b \quad x[n]_s = 0 \quad \text{choose } c_i \text{ if } i \in [n] \setminus \]
\[ c_i = 0 \quad i \in s \quad c^T x < c^T x' \quad \text{for } x' \in p \quad x' \neq x, \quad c^T x = 0 \]
claim \( c^T x > 0 \) for all \( x' \neq x, \quad x' \in p \) if \( Ax' = b \) and \( x' = 0 \) implies \( x' = x \)

\[ \square \]

2 Application- Machine Scheduling

There are \( n \) jobs and \( m \) machines. We have to find assignment of jobs to machine to minimize the load maximum load. This problem is similar to graham’s list scheduling algorithm. This machine scheduling problem NP-hard problem. It is also similar to subset sum problem.

one approach to solve this problem is greedy approach which is 2 approximation. This greedy algorithm perform worse when there are \( m \) jobs of size 1 and one job of size \( m \).where \( p_j \) is the processing time of job \( j \).

**makespan minimization on unrelated machines:** let \( p_{i,j} \) processing time of \( j^{th} \) job on \( i^{th} \) machine. Machines are identical if \( p_{i,j} = p_j \quad \forall \quad i. \)
\( x_{i,j} \) is variable for job \( j \) assign to \( i^{th} \) machine. Minimise \( T(\text{makespan}) \) using LP

\[
\sum_{i=1}^{n} x_{i,j} = 1 \quad \forall j(\text{jobs})
\]
\[
\sum p_{i,j} x_{i,j} \leq T
\]
\[ x_{i,j} \geq 0 \quad \forall i, j \]

Consider the instance where there are \( m \) machines and only one job, and each machine needs a time \( m \) to process the job. The optimal integral solution will assign the job arbitrarily to some machine, with a makespan of \( m \), while the optimal solution of our LP relaxation is fractional, and splits the job in equal parts to all the machines, with a makespan of 1; so the integrality gap is at least \( m \).

let have one job \( p_{i,j} = p_j \quad \forall \quad i \in [\text{machine}] \)

optimal LP, \( x_{i,j} = \frac{1}{m} \)
\[ T = \frac{p_j}{m} \]
by itself LP is not useful for multiplicative comparison. It can not find schedule for makespan \( \leq cT^* \) where \( c \) is constant.

**circumvent the LP gap** we guess the optimal makespan value \( T^* \) so the optimization problem become the feasibility checking problem

\[
\sum x_{i,j} = 1 \quad \forall j (\text{jobs}) \\
\sum p_{i,j} x_{i,j} \leq T^* \quad \rightarrow \quad \sum p_{i,j} x_{i,j} + s_i = T^* \\
\text{where} \quad \forall x_{i,j} \geq 0 \quad s_i \geq 0
\]

**How to round this LP**

solve LP, \( X = \) optimal solution

\( X \) is basic feasible point and extreme point

\( M=\)number of constraints= \( m(\text{machines})+n(\text{jobs}) \)

number of non-zero \( x_{i,j} \)'s \( \leq m+n \)

truly fractional variables \( 0 \leq x_{i,j} \leq 1 \quad \leq m \)

Let represent job scheduling by the graph \( G \). this graph has an edge \((i,j)\) if \( x_{i,j} > 0 \). this is bipartide graph.

![Figure 1.1: Bipartide of machine and job](image)

**lemma**: Any extreme point of \( LP(T^*) \) can have atmost \( m \) fractionally assigned jobs.

**Proof.** Let \( X \) be an extreme point of \( LP(T) \), and \( k \) represents the number of job-machine assignations in \( X \), i.e. the number of non-zero variables, so by the previous lemma we know that \( k \leq m+n \). Each job needs at least one such assignation, and it needs more than one assignation iff it is fractionally assigned. Therefore the number of fractionally assigned jobs in \( X \) is \( k \leq m \). 

**Lemma** for any extreme point \( X \) claim graph \( G(X) \) look like forest without cycle -on cycle.

**Proof.** Fix an extreme point \( X \). The graph \( G(X) \) is bipartite, it has \( m+n \) vertices and at most \( m+n \) edges. consider two case .

If \( G \) is connected, then \( G(X) \) necessarily corresponds to a bipartide tree ( one vertices set is having \( m \) vertices other set is having \( n \) vertices ) with one extra edge means one cycle. so it be come a tree with one cycle.
If $G$ is not connected, then we must prove that every connected component is a tree with at most one cycle. Let’s prove by contradiction, let’s there is a connected component $C$ which is not a tree with at most one cycle $C’$ represent remaining $G(x)$. Then $C$ and $C’$ are two separate scheduling instances, and the restrictions of $X$ in them, $X_C$ and $X_{C’}$. But the $X_C$ cannot be an extreme point, because $C = G(X_C)$ is connected and not tree with at most one cycle. So it would violate case 1. $x_C$ can be written as combination of other feasible points such as $X_C = \lambda y_1 + (1 - \lambda)y_2$ where $y_1 \neq y_2$. But $X$ could also be explained in terms of other feasible points of original instances such that $X = \lambda(X_{C’} + y_1) + (1 - \lambda)(X_{C’} + y_2)$, which is contradicts the fact that $x$ was an extreme point. Therefore each components of $G$ is a tree with at most one cycle.

Let $X$ be the optimal extreme point of $\text{LP}(T^*)$. It gives makespan of value $\leq T^* \leq \text{OPT}$. If we consider the graph $G(X)$ with many components and each components is tree with at most one cycle. Graph is actually contain machine and fractionally assigned jobs, where edge represent job assigned to machine. All leaves are machine. We just need to find a matching that have all the jobs. Till there are leaves in the graph, pick a leaf add the corresponding edge to the matching, and remove from the graph the corresponding job with all its child when their are no more leaves. Graph become either an empty graph or an even cycle. In the end we have perfect matching so we can find a matching that covers all fractionally assigned jobs.