Revision notes - MA4254

Ma Hongqiang

November 18, 2018

Contents

1 Graph and Digraph

1.1 Graphs

Definition 1.1 (Graph)**.**
A graph G is a pair (V, E), where

- V is a finite set, and
- E is a set of unordered pairs of elements of V.

Elements of V are called vertices and elements of E edges.
A pair of distinct vertices are **adjacent** if they define an edge. The edge is said to be **incident** to its defining vertices.
The **degree** of a vertex v, denoted as $\deg(v)$, is the number of edges incident to that vertex.

Definition 1.2 (Path, Cycle)**.**
A v_1v_k-**path** is a sequence of edges

$$\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\}$$

A **cycle** is a sequence of edges

$$\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\}, \{v_k, v_1\}$$

In both case we require the vertices to be all *distinct*.
A graph is said to be **acyclic** if it has no cycle.

Theorem 1.1.
If every vertex of G has degree at least two, then G has a cycle.

Definition 1.3 (Connected Graph)**.**
G is connected if each pair of vertices is connected by a path.

Theorem 1.2.
Let G be a connected graph with a cycle C and let e be an edge of C. Then $G - e$ is connected.

Definition 1.4 (Subgraph)**.**
H is a **subgraph** of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.
It is a spanning subgraph if in addition $V(H) = V(G)$.

Definition 1.5 (Tree)**.**
A **tree** is a connected acyclic graph.

Theorem 1.3.
If $T = G(V, E)$ is a tree, then $|E| = |V| - 1$.

Theorem 1.4.
:et $G = (V, E)$ be a connected graph. Then $|E| \geq |V| - 1$. Moreover, G is a tree if equality holds.

Definition 1.6 (Bipartite Graph)**.**
$G = (S, T, E)$ is a bipartite graph if any edge in E has one vertex in S and the other in T.

Definition 1.7 (Vertex-Edge Incidence Matrix)**.**
The **vertex-edge incidence matrix** of a graph $G = (V, E)$ is a matrix $A \in \{0, 1\}^{|V| \times |E|}$ such that

- The rows correspond to the edges of G,
- the columns correspond to the edges of G,
- entry $A_{v,ij}$ for vertex v and edge ij is

$$A_{v,ij} = \begin{cases} 0 & \text{if } v \neq i \text{ and } v \neq j \\ 1 & \text{if } v = i \text{ or } j \end{cases}$$

Definition 1.8 (Digraph)**.**
A **directed graph**(digraph) D is a pair (N, A) where

- N is a finite set and
- A is a set of ordered pairs of elements of N

Elements of N are called nodes and elements of A arcs.

- Node i is the tail of arc (i, j).
- Node j is the head of arc (i, j).

The in-degree (resp. out-degree) of node v, denoted $\deg^+(v)$(resp. $\deg^-(v)$) is the number of arcs with head(resp. tail) v.

Definition 1.9 (Bipartite Digraph)**.**
A bipartite digraph is defined as $D = (S, T, A)$ where for all edges in A, it is incident from a node in S to a node in S.

Definition 1.10 (Node-Arc Incidence Matrix)**.**
The **node-arc incidence matrix** of a graph $D = (N, A)$ is a matrix $M \in \{0, \pm 1\}^{|V| \times |A|}$ such that

- The rows correspond to the nodes of D.
- The columns correspond to the arcs of D
- the entry $M_{v,ij}$ for node v and arc ij is

$$M_{v,ij} = \begin{cases} 0 & \text{if } v \neq i \text{ and } v \neq j \\ -1 & \text{if } v = j, \\ +1 & \text{if } v = i \end{cases}$$

1.2 Convex Set

Definition 1.11 (Convex Set)**.**
A set $S \subseteq \mathbb{R}^n$ is **convex** if for any $x, y \in S$ and any $\lambda \in [0,1]$, we have $\lambda x + (1-\lambda)y \in S$

1.3 Hyperplanes and Half Spaces

Definition 1.12 (Hyperplane)**.**
Let $a \in \mathbb{R}^n \setminus \{0\}$ abd $b \in \mathbb{R}$. Then the set

$$\{x \in \mathbb{R}^n \mid a^{\mathrm{T}} x = b\}$$

is called a **hyperplane**.

Geometrically, the hyperplane above can be understood by expressing it in the form

$$\{x \in \mathbb{R}^n \mid a^{\mathrm{T}}(x - x^0) = 0\}$$

where x^0 is any point in the hyperplane, i.e., $a^{\mathrm{T}} x^0 = b$.

Definition 1.13 (Half Space)**.**
Let $a \in \mathbb{R}^n \setminus \{0\}$ and $b \in \mathbb{R}$. Then the set

$$\{x \in \mathbb{R}^n \mid a^{\mathrm{T}} x \geq b\}$$

is called a **halfspace**.

Notably, a halfspace is a convex set.

1.4 Polyhedra

Definition 1.14 (Polyhedron)**.**
A **polyhedron** is a set of the form $\{x \in \mathbb{R}^n \mid Ax \geq b\}$, for some $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

By letting $A = \begin{pmatrix} a_1^{\mathrm{T}} \\ \vdots \\ a_m^{\mathrm{T}} \end{pmatrix}$ and $b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$, we can understand the polyhedron to be the intersection of the halfspaces

$$\{x \in \mathbb{R}^n \mid a_i^{\mathrm{T}} x \geq b_i\}, \quad i = 1, \ldots, m$$

From this point of view, we can see that the intersection of two polyhedra is again a polyhedron.
A bounded polyhedron is sometimes called a **polytype**.

Definition 1.15 (Convex Combination)**.**
Let $x^1, \ldots, x^k \in \mathbb{R}^n$ and let $\lambda_1, \ldots, \lambda_k$ be nonnegative scalars whose sum is one.

1. The vector $\sum_{i=1}^n \lambda_i x^i$ is said to be a **convex combination** of the vectors $x^1, \ldots, x^k$.
2. The **convex hull** of the vectors $x^1, \ldots, x^k$ is the set of all convex combinations of these vectors.

1.5 Basic Feasible Solution

Definition 1.16 (Extreme Point)**.**
Letr $P \subseteq \mathbb{R}^n$ be a polyhedron. A vector $x \in P$ is an **extreme point** of P if we cannot find $y, z \in P$ distinct from x, and $\lambda \in [0, 1]$, such that $x = \lambda y + (1 - \lambda)z$.

Definition 1.17 (Active Constraint)**.**
Consider a polyhedron $P \in \mathbb{R}^n$, partition the constraints according to the sign:

- $a_i^{\mathrm{T}} x \geq b_i, i \in M_1$
- $a_i^{\mathrm{T}} x \leq b_i, i \in M_2$
- $a_i^{\mathrm{T}} x = b_i, i \in M_3$

where M_1, M_2, M_3 are finite index sets, each $a_i \in \mathbb{R}^n \setminus \{0\}$ and each $b_i \in \mathbb{R}$.
If $x^* \in \mathbb{R}^n$ satisfies $a_i^{\mathrm{T}} x^* = b_i$ for some $i \in M_1, M_2, M_3$, we say that the corresponding constraint is **active** at x^*. The **active set** of P at x^* is

$$I(x^*) = \{i \in M_1 \cup M_2 \cup M_3 \mid a_i^{\mathrm{T}} x^* = b_i\}$$

i.e., $I(x^*)$ is the set of indices of constraints active at x^*.

Theorem 1.5 (Linear Algebra Equivalence)**.**
Let $x^* \in \mathbb{R}^n$. The following are equivalent.

1. There are n linearly independent veectors in the set $\{a_i \mid i \in I(x^*)\}$.
2. The span of the vectors $a_i, i \in I(x^*)$, is all of $\mathbb{R}^n$.
3. The system $a_i^{\mathrm{T}} x = b_i, i \in I(x^*)$, has a unique solution.

Definition 1.18 (Basic Solution, Basic Feasible Solution)**.**
The vector $x^* \in \mathbb{R}^n$ is called a **basic solution** if

1. $a_i^{\mathrm{T}} x^* = b_i, i \in M_3$
2. There are n linearly independent vectors in $\{a_i\}_{i \in I(x^*)}$.

A **basic feasible solution**(BFS) is a basic solution satisfying all constraints, namely in M_1, M_2, M_3.

1.6 Finite Basis Theorem for Polyhedra

Definition 1.19 (Cone)**.**
A set $C \subseteq \mathbb{R}^n$ is a **cone** if $\lambda x \in C$ for all $\lambda \geq 0$ and $x \in C$.

It is obvious from definition that $0 \in C$.

Definition 1.20 (Convex Cone)**.**
For vectors $x^1, \ldots, x^k \in \mathbb{R}^n$, let

$$\text{cone}\{x^1, \ldots, x^k\} := \{x \in \mathbb{R}^n \mid x = \sum_{i=1}^{k} \lambda_i x^i, \lambda_i \geq 0, i = 1, \ldots, k\}$$

Then, $\text{cone}\{x^1, \ldots, x^k\}$ is a cone and convex set, which is called the **convex cone** generated by $x^1, \ldots, x^k$.

Definition 1.21 (Polyhedral Cone)**.**
The set $P = \{x \in \mathbb{R}^n \mid Ax \geq 0\}$ is called a **polyhedral cone**.

By a theorem of Weyl, we have $\text{cone}\{x^1, \ldots, x^k\}$ is a polyhedral cone.

Definition 1.22 (Recession Cone)**.**
Given $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, consider

$$P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$$

and $y \in P$.
The **recession cone** $R(P, y)$ of P at y is the set

$$\{d \in \mathbb{R}^n \mid A(y + \lambda d) \geq b, \text{ for all } \lambda \geq 0\}$$

It is easy to see the recession cone of P at any y is

$$\{d \in \mathbb{R}^n \mid Ad \geq 0\}$$

and is a polyhedral cone. Thus, the recession cone is independent of the starting point y, so we can denote the recession cone as $R(P)$.

For $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$, where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, the recession cone is

$$\{d \in \mathbb{R}^n \mid Ad = 0, d \geq 0\}$$

Definition 1.23 (Extreme Rays)**.**
Extreme rays of a polyhedral cone $C \subseteq \mathbb{R}^n$ are elements $d \neq 0$ such that there are $n - 1$ linearly independent constraints active at d.
Extreme rays of a nonempty polyhedron P are the extreme rays of the recession cone of P.

Definition 1.24 (Minkowski Sum)**.**
Let $A, B \subseteq \mathbb{R}^n$. The Minkowski sum $A + B$ is

$$A + B := \{a + b : a \in A, b \in B\}$$

Theorem 1.6 (Finite Basis Theorem for Polyhedra)**.**
Let $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$, where $\text{rank}(A) = n$, $A \in \mathbb{R}^{m \times n}$. Let $\{x^1, \ldots, x^q\}$ be the set of extreme points of P, which P has finitely many of, and let $\{d^1, \ldots, d^r\}$ be the set of extreme rays of P. Then,

$$P = \text{conv}\{x^1, \ldots, x^q\} + \text{cone}\{d^1, \ldots, d^r\}$$

Therefore, it can be shown that P is **bounded** *if and only if* the recession cone of P contains zero vector only.

Theorem 1.7 (Farkas' Lemma)**.**
This lemma can be used, together with LP duality to prove the Finite Basis Theorem. The lemma goes:
Exactly one of the following holds:

1. $\{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\} \neq \varnothing$ or
2. $\{y \in \mathbb{R}^m \mid A^{\mathrm{T}}y \geq 0, y^{\mathrm{T}}b < 0\} \neq \varnothing$.

1.7 Simplex Method Revisited

Please read `MA3252.pdf`.

2 Total Unimodularity

Definition 2.1 (Submatrix)**.**
Let $A \subseteq \mathbb{R}^{m\times n}$, where $B \subseteq \{1,\dots,m\}$ and $C \subseteq \{1,\dots,n\}$. We can take submatrix $A_{(B,C)}$, where rows selected by *ordered* set B and columns by C.

By convention, when we highlight rows $B \subseteq \{1,\dots,m\}$, we use A_B to denote the submatrix.

Similarly, we use A_J to denote the submatrix whose columns are selected by index set J.

2.1 Total Unimodularity

Consider the **integer** LP

$$\begin{aligned} &\max c^{\mathrm{T}}x \\ (P)&\text{s.t.} Ax = b \\ &x \geq 0, x \in \mathbb{Z} \end{aligned}$$

where $A \in \mathbb{Z}^{m\times n}$, $b \in \mathbb{Z}^m$ and $c \in \mathbb{Z}^n$ are all integral.

Definition 2.2 (Unimodular, Totally Unimodular)**.**
A *square, integer* matrix B is **unimodular** if $|\det(B)| = 1$.

A *square* matrix A is **totally unimodular**(TU) if *every square, nonsigular* submatrix of A is unimodular.

Hence, a TU matrix is a $\{0, \pm 1\}$-matrix.

Theorem 2.1. Suppose $A \in \mathbb{Z}^{n\times n}$ is unimodular and $b \in \mathbb{Z}^n$, then $x = A^{-1}b$ is integral.

Theorem 2.2. If $A \in \mathbb{Z}^{m\times n}$ where $m > n$, is TU, then every basic solution to $P := \{x : Ax \geq b\}$, where $b \in \mathbb{Z}^m$, is integral.

We have the following proposition:

Theorem 2.3.
Suppose $A \in \mathbb{Z}^{m\times n}$ is TU. Then

1. $-A$ and A^{T} are TU.
2. $\begin{pmatrix} A & e_i \end{pmatrix}$ is TU, where e_i is the ith unit vector of $\mathbb{R}^m$.
3. $\begin{pmatrix} A & I \end{pmatrix}$ is TU, where $I \in \mathbb{R}^{m\times m}$ is the identity matrix.
4. $\begin{pmatrix} A \\ I \end{pmatrix}$ is TU, where $I \in \mathbb{R}^{n\times n}$ is the identity matrix.

From the proposition we have

Theorem 2.4. If $A \in \mathbb{Z}^{m\times n}$, where $m < n$ is TU, then every basic solutino to $P := \{x : Ax = b, x \geq 0\}$, where $b \in \mathbb{Z}^m$, is integral.

We have the following equivalence on TU.

Theorem 2.5.
For any $A \in \mathbb{Z}^{m\times n}$, the following are equivalent:

1. A is TU.
2. For any integral b, the extreme points, if any, of $S(b) := \{x : Ax \leq b, x \geq 0\}$ are integral.
3. Every square nonsigular submatrix A has integer inverse.

Theorem 2.6 (A sufficient condition of TU)**.**
An integer matrix A with all $a_{ij} \in \{0, \pm 1\}$ is TU if

1. At most two nonzero element appear in each column,
2. Rows of A can be partitioned into 2 subsets M_1 and M_2, such that
   (a) if a column contains two nonzero elements with the same sign, one element is in each of teh subsets.
   (b) if a column contains two nonzero elements of opposite signs, both elements are in the same subset.

Theorem 2.7.
The vertex-edge incidence matrix of a **bipartite graph** is TU.
The node-arc incidence matrix of a **digraph** is TU.

2.2 Applications

Definition 2.3 (Network, Minimum Cost Capacitated Problem)**.**
We represent a **network** as a directed graph $G = (V, E)$. For each arc $(i, j) \in E$, associate it with the unknown flow x_{ij} and (possibly) infinite capacity d_{ij}. Clearly, $0 \leq x_{ij} \leq d_{ij}$.
We partition the set V into three sets:

- V_1: set of sources or origins;
- V_2: set of intermediate points;
- V_3: set of destinations or sinks;

We define, for each $i \in V_1$, a_i to be the **supply** of the commodity; for each $i \in V_3$, b_i to be the **demand** of the commodity.
Each $i \in V_2$ are intermediate nodes, which should have zero net flow.

Additionally denote

- $V_{\text{out}}(i) = \{j \mid (i,j) \in E\}$ to be the set of nodes connected by outgoing edges
- $V_{\text{in}}(i) = \{j \mid (j,i) \in E\}$ to be the set of nodes connected by incoming edges

The **minimum cost capacitated problem** is

$$v(P) = \min \sum_{(i,j)\in E} c_{ij}x_{ij}$$

subject to

$$\sum_{j\in V_{\text{out}}(i)} x_{ij} - \sum_{j\in V_{\text{in}}(i)} x_{ji} \begin{cases} \leq a_i, & i \in V_1 \\ = 0, & i \in V_2 \\ \leq -b_i & i \in V_3 \end{cases}$$

and

$$0 \leq x_{ij} \leq d_{ij}, \quad (i,j) \in E$$

Theorem 2.8.
The constraint matrix corresponding to the minimum cost capcitated problem is TU.

In fact, assignment problem can be a subset of MCCP, where $a_i = 1$ for all i and $b_j = 1$ for all j, and $|V_1| = |V_3|$ and all constriants' equality holds exactly.
Shortest path problem can also be formulated as MCCP, where $a_1 = b_m = 1$ and $V_1 = \{1\}, V_3 = \{m\}$.
Maximum flow problem is with $a_1 = b_m = \infty$ and the objective is to maximise flow out of 1. For shortest path problem, we need to assume there is no uncapcitated negative cycles. This can be achieved by having

- the sum of costs of arcs in the cycle is nonnegative, or
- the minimal capacity of an arc in the cycle is bounded.

Theorem 2.9.
A shortest path problem with $x \in \{0,1\}^{|V|}$ as constraint on x is equivalent to the problem with $x \geq 0$ as constraint.

3 The Shortest Path

3.1 Dijkstra's Algorithm

Dijkstra's algorithm solves the shortest path problem from a chosen point to **all other points**. In this subsection, we assume that $c_{ij} \geq 0$.
Dijkstra's algorithm can be summarised as follows. Denote

- P: permanently labeled nodes
- T: temporarily labeled nodes

P and T is a partition of $V = \{1, \ldots, n\}$, i.e.,

$$P \cap T = \varnothing \quad P \cup T = V$$

Label for node j $[u_j, l_j]$, where

- u_j: the length of the (may be temporary) shortest path from node i to j
- l_j: the preceding node in the path

The main steps, after the setup are

Step 0 $P = \{1\}, u_1 = 0, l_1 = 0, T = V \setminus P$. Compute

$$u_j = \begin{cases} c_{1j} & \text{if } (1,j) \in E \\ \infty & \text{if } (1,j) \notin E \end{cases}$$

and

$$l_j = \begin{cases} 1 & \text{if } (1,j) \in E \\ 0 & \text{if } (1,j) \notin E \end{cases}$$

Step 1 Find $k \in T$ such that

$$u_k = \min_{j \in T} \{u_j\}$$

Let $P = P \cup \{k\}$ and $T = T \setminus \{k\}$. If $k = n$, stop.

Step 2 For $j \in T$, if $u_k + c_{kj} < u_j$, let $[u_j = u_k + c_{kj}, l_j = k]$ and go back to step 1.

The running time of Dijkstra's algoritm is $O(n^2)$.

3.2 Bellman's Equation

Let c_{ij} be the length of arc (i,j). Let u_{ij} be the length of the shortest path from $i \to j$. Define

$$u_i = u_{1i}$$

We can use Bellman's equations to prove the correctness of Dijkstra.

Theorem 3.1 (Bellman's Equations)**.**
The Bellman's Equations below hold

$$\begin{cases} u_1 = 0 \\ u_i = \min_{k \neq i} \{u_k + c_{ki}\} \end{cases}$$

3.3 PERT or CPM Network

A large project can be divided into many unit tasks, partially ordered.

Theorem 3.2.
A digraph is acyclic if and only if its nodes can be renumbered so that for all arcs (i, j), $i < j$.

Theorem 3.3.
For any acyclic graph,at least one node has indegree 0.

Theorem 3.4 (Bellman's Equation on Acyclic Graphs)**.**
For reordered acyclic graphs, the bellman's equation becomes

$$\begin{cases} u_1 = 0 \\ u_i = \min_{k<i}\{u_k + c_{ki}\} \end{cases}$$

Theorem 3.5 (Equivalence between Shortest and Longest Path)**.**
They are equivalent since $\max c^{\mathrm{T}}x = -\min c^{\mathrm{T}}x$.

3.4 Bellman-Ford Method

The Bellman-Ford method solves the shortest path problem from one starting node to all other nodes. Also, it allows $c_{ij} < 0$ for some $(i, j) \in E$.
Bellman-Ford also can detect presense of negative cycles.

The algorithm is

Step 1 $u_1^{(1)} = 0, u_j^{(1)} = c_{1j}, j \neq 1$.

Step k For $k = 2, \ldots, n$,

$$u_j^{(k)} = \min\{u_j^{(k-1)}, min\{u_i^{(k-1)} + c_{ij}\}\} \quad \text{for } j = 1, \ldots, n$$

The total computational cost is $O(n^3)$.

3.5 Primal Dual Method

Let $\tilde{A}$ be the incidence matrix of the digraph $G = (V, E)$ where $V = \{1. \ldots, m\}$.
Each arc $(i, j) \in E$ has length $c_{ij} \geq 0$ and flow $x_{ij} \geq 0$. The shortest path problem can be formulated as

$$
\begin{aligned}
(\tilde{P}) \min & \sum_{(i,j)\in E} c_{i,j}x_{i,j} \\
\text{s.t.} & \tilde{A}x = \begin{pmatrix} +1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{pmatrix} \\
& x \geq 0
\end{aligned}
$$

Let A be the remaining submatrix of $\tilde{A}$ by removing the last row of $\tilde{A}$. Then we have

$$
\begin{aligned}
(P) \min & \sum_{(i,j)\in E} c_{i,j}x_{i,j} \\
\text{s.t.} & \tilde{A}x = \begin{pmatrix} +1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\
& x \geq 0
\end{aligned}
$$

Next we establish the primal dual method for general LPs.
For $A \in \mathbb{R}^{m\times n}, c \in \mathbb{R}^n, b \in \mathbb{R}^m_+$, consider the LP and its dual

$$
\begin{aligned}
(P) \min_x & \, c^{\mathrm{T}}x \\
\text{s.t.} & \, Ax = b \geq 0 \\
& \, x \geq 0
\end{aligned}
$$

and

$$
\begin{aligned}
(D) \max_\pi & \, \pi^{\mathrm{T}}b \\
\text{s.t.} & \, \pi^{\mathrm{T}}A \leq c^{\mathrm{T}}
\end{aligned}
$$

Suppose an LP arising from a shortest path problem has $m \ll n$, strong duality shows that we can solve (D) to solve (P).

One observation is that the starting feasible π can be $\pi = 0$ if $c \geq 0$.
Suppose we have a π feasible to dual (D). Define $J \subseteq \{1, \ldots, n\}$, the set of admissible columns, by

$$J = \{j : \pi^{\mathrm{T}}A_j = c_j\}$$

Then for any $j \notin J$, we have $\pi^{\mathrm{T}} A_j < c_j$.
Consider

$$\begin{aligned}(DRP_J) w_J^* = \max_{\bar{\pi}} & \bar{\pi}^{\mathrm{T}} b \\ \text{s.t.} & \bar{\pi}^{\mathrm{T}} A_j \leq 0, \quad j \in J \\ & \bar{\pi}_i \leq 1, i = 1, \ldots, m\end{aligned}$$

(DRP_J) searches for direction $\bar{\pi}$ to update π. Also, (DRP_J) is the dual of (RP_J), to be defined later.
There are many parts behind the primal dual algorithm, so we can list down some important points here:

$$(P) \leftarrow (RP_J) \rightarrow (DRP_J) \rightarrow (D)$$

1. Solve (D) by solving a sequence of (DRP_J)s.
2. Solve (DRP_J) by solving (RP_J).
3. Each (RP_J) is easier to solve than (P).
4. When (RP_J) has objective 0, then optimal solution to (P) is found.

For step 1, note that (DRP_J) provides a feasible direction $\bar{\pi}$ to (D), which can be used to improve π to $\pi + \theta\bar{\pi}$ where $\theta \geq 0$.
For step 2, we first define (RP_J).

Definition 3.1 (Restricted Primal)**.**
The restricted primal of (P) is

$$\begin{aligned}(RP_J) \ \ \xi_J^* = \min_{x, x^a} & 0^{\mathrm{T}} x + \sum_{i=1}^{m} x_i^a \\ \text{s.t.} & Ax + x^a = b \\ & x_j \geq 0 \text{ for all } j \\ & x_j = 0 \text{ for all } j \notin J \\ & x_i^a \geq 0, i = 1, \ldots, m\end{aligned}$$

It can be simplified as

$$\begin{aligned}(RP_J) \ \ \xi_J^* = \min_{x_J, x^a} & 0^{\mathrm{T}} x + \sum_{i=1}^{m} x_i^a \\ \text{s.t.} & A_J x_J + x^a = b \\ & x_J \geq 0, x^a \geq 0\end{aligned}$$

It can be shown that (RP_J) is dual of (DRP_J).
From strong duality, we have $\xi_J^* = w_J^*$.
To get $\bar{\pi}$, we have $\bar{\pi}^{\mathrm{T}} = c_B^{\mathrm{T}} A_B^{-1}$.
With the definition of (RP_J), we can establish the general primal dual algorithm:

Definition 3.2 (Primal Dual Method)**.**

1. Start with π feasible in (D).
2. Loop
   (a) Set $J = \{j : \pi^{\mathrm{T}} A_j = c_j\}$. (Update J)
   (b) Solve (RP_J) by simplex method to get $(\bar{x}, \bar{x}^a)$ and $\bar{pi}$. (This step essentially solves (DRP_J) via (RP_J) which is easier to solve.)
   (c) If $\xi_J^* = 0$, then $\bar{x}$ is optimal for (P).
   (d) Else if $\bar{\pi}^{\mathrm{T}} A_j \leq 0$ for all $j \notin J$, (P) infeasible.
   (e) Else, we let $\pi \leftarrow \pi + \theta_1 \bar{\pi}$, where

$$\theta_1 := \max\{\theta : [\pi + \theta \bar{\pi}^{\mathrm{T}} A \leq c^{\mathrm{T}}\} = \min\{\frac{c_j - \pi^{\mathrm{T}} A_j}{\bar{\pi}^{\mathrm{T}} A_j} : j \notin J \text{ and } \bar{\pi}^T A_j > 0\}$$

   (f) End if
3. End Loop

For step 3, we claim that (RP_J) is easier to solve than (P). When solving RP_J, we will essentially remove some columns from J_{old}, which evolves J as J_{new}. Furthermore, we have $\xi^*_{\text{new}} \leq \xi^*_{\text{old}}$.

Theorem 3.6 (Complementary Slackness)**.**
Let x and π be feasible in (P) and D respectively. Let $\mathbf{a}_i$ be the ith row of A and $\mathbf{A}_j$ be the jth column of A. Then x and π are optimal if and only if

$$\pi_i(\mathbf{a}_i^{\mathrm{T}} x - b_i) = 0 \quad \text{for all } i$$
$$(c_j - \pi^{\mathrm{T}} \mathbf{A}_j) x_j = 0 \quad \text{for all } j$$

With this theorem, we can proof the following:

Theorem 3.7.

1. If $\xi_J^* = 0$, then $\bar{x}$ is optimal for (P) and π optimal for (D).
2. If $w_J^* = \xi_J^* > 0$ and $\bar{\pi}^{\mathrm{T}} A_j \leq 0$ for all j, then (P) is infeasible.
3. Any BFS of (RP_J) is a BFS of $(RP_{\{1,\ldots,n\}})$.

The following theorem establishes that primal dual algorithm ends in finitely many iterations.

Theorem 3.8.
Consider the primal dual algorithm,

1. In line 3, J^{new} contains all elements in J^{old} which are the basic indices of the optimal solution x obtained from solving $(RP_{J^{\text{old}}})$.
2. If an anticycling rule is used in simplex method to solve (RP_J), the primal dual algorithm ends in finite time.

3.6 Primal Dual Method for Shortest Path Problem

Specific to shortest path problem, we have for (i,j)th column of A,

$$\pi^{\mathrm{T}} A_{(i,j)} = \pi_i - \pi_j$$

and the set J now has a simpler expression:

$$J = \{\text{arcs}(i,j) : \pi_i - \pi_j = c_{ij}\}$$

Furthermore, (DRP_J) is also simpler:

$$\begin{aligned}(DRP_J) w_J^* = \max & \bar{\pi}_1 \\ \text{s.t.} & \bar{\pi}_i - \bar{\pi}_j \leq 0 \quad \text{for all } (i,j) \in J \\ & \bar{\pi}_i \leq 1 \text{ for all } i = 1, \ldots, m-1 \\ & \bar{\pi}_m = 0\end{aligned}$$

It is easy to see that $w^* = 0$ if there is a path from 1 to m along edges in J and 1 otherwise. Also, specific to shortest path,

$$\begin{aligned}\theta_1 &= \min\{\frac{c_{(i,j)} - (\pi_i - \pi_j)}{\bar{\pi}_i - \bar{\pi}_j} : (i,j) \notin J, \bar{\pi}_i - \bar{\pi}_j > 0\} \\ &= \min\{c_{(i,j)} - (\pi_i - \pi_j) : (i,j) \notin J, \bar{\pi}_i - \bar{\pi}_j > 0\}\end{aligned}$$

Theorem 3.9.
If $c > 0$, then every arc (i,j) that becomes admissible stays admissible throughout the algorithm. Also, $pi_i, i \in W$ is the length of the shortest path from node i to node m and algorithm proceeds by adding to W, at each stage, the nodes not in W next closest to node m.

3.7 Floyd-Warshall Method

The Floyd Warshall method finds the shortest path between all pairs, and also detect negative cycles:

Step 0 $u_{ij}^{(1)} = c_{ij}$, $i,j = 1, \ldots, n$

Step k For $k = 1, \ldots, n$,

$$u_{ij}^{(k+1)} = \min\{u_{ij}^{(k)}, u_{ik}^{(k)} + u_{kj}^{(k)}\} \quad i,j = 1, \ldots, n$$

4 Greddy Algorithm and Computational Complexity

4.1 Matriod

Definition 4.1 (Independent System)**.**
Suppose we have a finite ground set S, $|S| < \infty$, and a collection Ξ, of subsets of S.
$H := (S, \Xi)$ is an **independent system** if

- $\varnothing \in \Xi$, and
- $X \subseteq Y \in \Xi \Rightarrow X \in \Xi$.

We call elements in Ξ independent sets, and subsets of S not in Ξ dependent sets.

Definition 4.2 (Matriod)**.**
$H = (S, \Xi)$ is a matriod, if it is

- an independent system, and
- $X, Y \in \Xi$ with $|X| = |Y| + 1$ implies $\exists e \in X \setminus Y$ such that $Y + e \in \Xi$.

4.2 Greedy Algorithm

Suppose $H = (S, \Xi)$ is an independent system. Let $W : S \to \mathbb{R}_+$ be a weight function with $W(e) \geq 0$ for all $e \in S$.
For any $X \subseteq S$, define the weight of the set as

$$W(X) := \sum_{e \in X} W(e)$$

The matriod problem is the following optimisation problem:

$$\begin{aligned} &\max W(X) \\ &\text{s.t.} X \in \Xi \end{aligned}$$

Theorem 4.1 (Greedy Algorithm)**.**
Suppose we rearrange the elements such that $W(e_1) \geq W(e_2) \geq \cdots \geq W(e_n)$.

Step 0 Let $X = \varnothing$.

Step k If $X + e_k \in \Xi$, let $X := X + e_k$, where $k = 1, \ldots, n$

This greedy algorithm works for the matriod problem if and only if H is a matriod.

Theorem 4.2.
Let $H = (S, \Xi)$ be an independent system. The following are equivalent:

1. H is a matriod, i.e., $X, Y \in \Xi, |X| = |Y| + 1$ implies $\exists e \in X \setminus Y \text{s.t.} Y + e \in \Xi$.
2. Greedy algorithm solves problem for any $W : S \to \mathbb{R}_+$.

3. If $A \subseteq S$, and X, Y are **maximal independent subsets** of A, then $|X| = |Y|$.
   X is a maximal independent subset of A means $X \in \Xi$ and $X \subseteq A$, and there is no $X' \subsetneq X$ such that $X' \in \Xi$ and $X' \subseteq A$.

The following lemma is introduced in proving the above theorem:

Theorem 4.3.
Suppose $H = (S, \Xi)$ and $W : S \to \mathbb{R}_+$.
Suppose the greedy algorithm finds $X = \{x_1, \ldots, x_i\}$. Let $\bar{e} \in S$. Define $A = \{e \in S : W(e) \geq W(\bar{e})\}$.
Let m be such that $W(x_{m-1}) \geq W(\bar{e}) > W(x_m)$, where $m = i + 1$ means $W(x_i) \geq W(\bar{e})$, then $\{x_1, \ldots, x_{m-1}\}$ is a maximal independent set of A.

4.3 Introduction to Computational Complexity

Definition 4.3 (Instance, Problem)**.**
An **instance** of an optimization problem consists of a feasible set F and a cost function $c : F \to \mathbb{R}$.
An optimization **problem** is a collection of instances.
The **size** of an instance is defined as the number of bits used to describe the instance, according to a prescribed format.

Generally, we need $O(\log r)$ bits to encode an integer r.
We need $O(V^2)$ to encode a graph using adjacency matrix and $O(E \log V)$ to encode a graph using edge list.

Definition 4.4 (P and NP)**.**
An algorithm runs in **polynomial time** if there exists an integer k such that $T(n) = O(n^k)$.
A combinatorial optimization problem resides in class $\mathcal{P}$ if there is a polynomial time algorithm under bit model.
A problem resides in class $\mathcal{NP}$ if for all YES instance, there exists a polynomial length certificate that verifies in polynomial time that the answer is indeed yes.

Theorem 4.4 (Numerical Problems)**.**
Suppose a problem have numerical components, like $\mathbf{A}, \mathbf{b}, \mathbf{c}$ in LP, we require definition for instance to only involve integers and rational numbers so that we can represent the numbers in binary.
Suppose that an algorithm is such that it takes polynomial time under the arithmetic model $+, -, \times, \div$, and on instances of size n, any integer produced in the course of execution has size bounded by a polynomial in n, then the algorithm runs in polynomial time under bit model.

Theorem 4.5.
Problem of finding a minimizer of an LP with integer entries is in $\mathcal{NP}$.
LP is in $\mathcal{P}$.

Definition 4.5 (Polynomial Time Reduction)**.**
Suppose there is an algorithm for a problem A consisting of

- A polynomial time computation, and
- A polynomial number of subroutine calls to an algorithm for problem B.

Then problem A **reduces** in polynomial time to problem B. We denote this reduction as $A \leq B$.

In this definition, all references to polynomiality are with respect to the size of an instance of problem A.
A consequence of such reduction is that if A is known to be hard and $A \leq B$ then B is also hard.

Theorem 4.6. If $A \leq B$ and $B \in \mathcal{P}$, then $A \in \mathcal{P}$.

4.4 Three Forms of a CO Problem

A CO problem is of the form of F is the feasible set and $c : F \to \mathbb{R}$ the cost function,

$$\min c(f) \text{s.t.} f \in F$$

The above CO problem has three versions:

- **Optimization version**: find an optimal solution
- **Evaluation version**: find optimal value
- **Recognition version**: Given an integer L, is there a feasible solution $f \in F$ such that $c(F) \leq L$?

It is worth noting the algorithmic difficulty of these three versions are related.

Theorem 4.7 (Evaluation in P Implies Recognition in P)**.**
If a polynomial time algorithm for the **evaluation** problem exists, then a polynomial time algorithm for the **recognition** problem exists. Equivalently,

$$\text{Recognition} \leq \text{Evaluation}$$

Theorem 4.8 (Optimization in P Implies Evaluation in P)**.**
If a polynomial time algorithm for the **optimization** problem exists and the cost $c(f)$ of any feasible $f \in F$ can be computed in polynomial time, then a polynomial time algorithm for the evaluation problem exists.

For problems with finite $c(F)$ values, a polynomial time algorithm for recognition problem implies a polynomial time algorithm for evaluation problem, via binary search.
Also, sometimes a polynomial time evluation algorithm gives polynomial time optimization algorithms for some problems.

4.5 co-NP and NPC

Definition 4.6 (co-NP)**.**
A combinatorial problem is in co-NP if for all "No" instances, there exists a polynomial length "certificate" that verifies in polynomial time that answer is indeed no.

Obviously, $P \subseteq co - NP$, but it remains open that whether P equals co-NP.

Definition 4.7 (Transformation)**.**
For YES/NO problems A and B, we say that A **transforms** to B in polynomial time, if there exists a polynomial time algorithm which given an instance l_1 of problem A, outputs an instance l_2 of B, such that l_1 is YES instance of A if and only if l_2 is a YES instance of B.

Definition 4.8 (NP Hard)**.**
A problem A is NP hard if for any problem $B \in NP$, $B \leq A$.

Theorem 4.9. Suppose that a problem C is NP-hard, and $C \leq D$, then D is NP-hard.

Definition 4.9 (SAT)**.**
Define a set of boolean variables $\{x_1, \ldots, x_n\}$. To each x_i we assign a label of true or false. We denote x_i as true and $\bar{x}_i$ as false. We refer to x_i and $\bar{x}_i$ as literals.
Let symbol $\vee$ denote or and $\wedge$ denote and. Any Boolean expression has a conjunction normal form(CNF), which is a finite conjunction, each of a finite disjunction of literals. We denote the disjunctive grouping as a **clause**.
Clearly, a clause is true if at least one of the literal is true. And a CNF expression is true if and only if all caluses are true.
The satisfiability problem(SAT), in its decision version, asks, given an expression in CNF, can literals be assigned so that the expression is `true`?

Theorem 4.10.
For any problem $Q \in NP$, $Q \leq SAT$.
This shows that SAT is NP hard.

Definition 4.10 (NP Complete)**.**
A YES/NO problem A is in NPC if

- $A \in NP$ and
- for any problem $B \in NP$, $B \leq A$.

4.6 LP is in NP

Theorem 4.11.
Let x be a BFS of the polyhedron $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$. Then

$$x = (\frac{p_1}{q}, \ldots, \frac{p_n}{q})$$

where $p_i \in \mathbb{N}$ and $0 \leq p_i \leq 2^L$ for $i \in \{1, \ldots, n\}$, $q \in \mathbb{N}$ and $1 \leq q \leq 2^L$ and $L \leq$ size$(LP(A, b, c))$.

The L is defined as below:

Definition 4.11.
For a triple (A, b, c) of an LP, let

$$L := \text{size}(\det_{\max}) + \text{size}(b_{\max}) + \text{size}(c_{\max}) + m + n - 1$$

where $\det_{\max} := \max_{A'}(|\det(A'|)$, $b_{\max} = \max_i(|b_i|)$ and $c_{\max} := \max_j(|c_j|)$.

Theorem 4.12. *1. If $n \in \mathbb{Z}$, then $|n| \leq 2^{size(n)-1} - 1$.*

2. If $v \in \mathbb{Z}^n$, then $\|v\| \leq \|v\_1\| \leq 2^{size(v)-n} - 1$

3. If $A \in \mathbb{Z}^{n \times n}$, then $|\det(A)| \leq 2^{size(A)-n^2} - 1$.

Theorem 4.13. $L < \text{size}(LP)$ for all A, b, c.

Theorem 4.14. $LP \in NP$.

5 Algorithms for NPC problem

Solving NP problem cannot be expected to complete in polynomial. We can solve

- Exactly, by exact methods like **branch and bound**, **cutting plane**, **branch and cut**, **dynamic programming**
- Approximately in polynomial time, by **approximation methods**
- in reduced search space, via **local search**
- by using heuristics, such as **genetic algorithm** and **simulated annealing**]
- by limiting to special classes, or using other methods

5.1 Generic Cutting Plane Algorithm

The genetic cuttinng plane algorithm is used to solve **integer LP** problem:

$$\begin{aligned} &\min c^{\mathrm{T}} x \\ &\text{s.t.} Ax = b \\ &\quad x \geq 0, x \text{ integer} \end{aligned}$$

Step 1 Solve LP relaxation(i.e. with x integer requirement removed). Let x^* be an optimal solution/

Step 2 If x^* is integer, stop. x^* is an optima solution to the ILP.

Step 3 If not, add an inequality constraint (#) to LP relaxation that all integer solutions satisfy but not x^*. Go to Step 1.

Here, by Gomory cuts of integer programs, (#) is defined as

$$[\sum_{j \in N} (\lfloor \bar{a}_{ij} \rfloor - \bar{a}_{ij}) x_j] \leq \lfloor \bar{a}_{i0} \rfloor - \bar{a}_{i0}$$

if the solution to x_i in the optimal tableau $\bar{a}_{i0} := (B^{-1}b)_i$ fractional, i.e. $x_i + \sum_{j \in N} \bar{a}_{ij} x_j = \bar{a}_{i0}$. N is just the nonbasic set and $\bar{a}_{ij} = (B^{-1}A_j)_i$.
We can add a slack variable s on the left hand side and put it to the optimal tableau and continue to solve using dual simplex method.

5.2 Gomory's Mixed Integer Cutting-Plane Method

A general Mixed Integer LP model can be written as

$$\begin{aligned} \min\, & c_1^{\mathrm{T}} x_1 + c_2^{\mathrm{T}} x_2 \\ & \text{s.t.} A_1 x_1 + A_2 x_2 = b \\ & \quad x_1 \geq 0 \\ & \quad x_2 \geq 0, \text{integer} \end{aligned}$$

where $A_1 \in \mathbb{R}^{m \times n_1}$, $A_2 \in \mathbb{R}^{m \times n_2}$ and $x_1 \in \mathbb{R}^{n_1}$ and $x_2 \in \mathbb{R}^{n_2}$.

Theorem 5.1 (Gomory cut for MILPs)**.**
Let x^* be an optimal BFS of an LP relaxation. Here A, b, c not necessary to be integral. Consider an equality from *optimal tableau*:

$$x_k + \sum_{j \in N} \bar{a}_{kj} x_j = \bar{a}_{k0}$$

where x_k is to be integral and $\bar{a}_{k9}$ is fractional.
Denote $J^+ = \{j \in N \mid \bar{a}_{kj} \geq 0\}$ and $J^- = \{j \in N \mid \bar{a}_{kj} < 0\}$.
Define

$$\beta_k := \bar{a}_{k0} - \lfloor \bar{a}_{k9} \rfloor > 0$$

Then a mixed cut constraint is

$$-\sum_{j \in J^+} \bar{a}_{kj} x_j - \frac{\beta_k}{\beta_k - 1} \sum_{j \in J^-} \bar{a}_{kj} x_j \leq -\beta_k$$

If we systematically add these cuts and use appropriate anti-cycling rules, this algorithm can terminate finitely for solving general integer LPs.

5.3 Branch and Bound

The idea of branch and bound is to divide the general LP problem

$$\begin{aligned} &\min c^{\mathrm{T}} x \\ &\text{s.t.} x \in F \end{aligned}$$

to a finite collection of subproblems:

$$\min c^{\mathrm{T}} x \text{s.t.} x \in F_i$$

for $i = 1, \ldots, k$. where F_i forms a partition of F.
One way to divide is to use branching, which gives a tree of subproblems.
In the case of solving MILP, we first start from the relaxed problem of general LP, then we do the branching by excluding the fractional solution by adding a constraint, either this component of x no greater than the floor or no less than the ceiling.

Theorem 5.2 (Generic Branch and Bound Algorithm)**.**

Step 0 Initially, U is set either to ∞ or to the cost of some feasible solution, if one happens to be available.

Step 1 Select an active subproblem F_i. Here, we select the subproblem with lowest lower bound.

Step 2 If the subproblem is infeasible, delete it. Otherwise, update the lower bound $b(F_i)$ for the corresponding subproblem.

Step 3 If $b(F_i) \geq U$, delete the subproblem.

Step 4 If $b(F_i) < U$, either

- Obtain an optimal solution to the subproblem
- Break the corresponding subproblem into further subproblems to add to the list of active subproblems.

Step 5 Go to step 1.

5.4 Dynamic Programming

Dynamic programming uses a recurrence relation to try to solve a problem at hand.
Let us consider the travelling salesman problem on $G = (V, E)$ a directed graph with n nodes and c_{ij} be the cost of arc (i, j). The TSP wants to choose a cycle visiting all nodes with lowest cost.
The brute force algorithm requires the evaluation of all $n!$ cycles directly. Dynamic programming will run faster.
In Dynamic programming, let $C(S, k)$ be the minimum cost over all paths that start at node 1 and end at node k, and visit all nodes in the set S exactly once. We call (S, k) a state, and this state can be reached from states $(S \setminus \{k\}, m)$ with $m \in S \setminus \{k\}$ at a transition cost c_{mk}.
We have the recursion

$$C(S, k) = \min_{m \in S \setminus \{k\}} (C(S \setminus \{k\}, m) + c_{mk}), k \in S$$

and the base case $C(\{1\}, 1) = 0$.
Clearly, we have $O(n2^n)$ states and evaluating a state takes $O(n)$. Therefore, we can get teh optimal tour's cost by

$$\min_k (C(\{1, \ldots, n\}, k) + c_{k1})$$

and get the actual route by looking at the arg min, within $O(n^2 2^n)$ time, which is faster than $O(n!)$.
Another application is on $0 - 1$ knapsack problem, where we want to

$$\max c^{\mathrm{T}} x \text{s.t.} a^{\mathrm{T}} x \leq b, x_i \in \{0, 1\}, i = 1, \ldots, n$$

with a, b, c integral and $a, b, c > 0$.
Define $C_j(w) = \max c_1 x_1 + \cdots + c_j x_j \text{s.t.} a_1 x_1 + \cdots + a_j x_j \leq w, x_i \in \{0, 1\}, i = 1, \ldots, j$. Then note $C_n(b)$ gives the optimal solution of the $0 - 1$ knapsack problem. Furthermore, we have recursion

$$C_{j+1}(w) = \max\{C_k(w), C_j(w - a_{j+1}) + c_{j+1}\}$$

and initial conditions $C_0(w) = 0$ for all w, $C_j(0) = 0$ for all j.
This algorithm will run in polynomial time $O(nb)$.
We can also recover which item to take easily.
Note, the greedy solution, which takes maximum per-unit-weight value, on $0 - 1$ knapsack does not work optimally.

5.5 ϵ-approximation Algorithms

Given a CO problem, with $A = \{F, c\}$, define optimal objective value $c(x^*) = \min c(x) \text{s.t.} x \in F$. An approximation algorithm finds $\bar{x}$ such that

$$|\frac{c(\bar{x}) - c(x^*)}{\max\{c(\bar{x}), c(x^*)\}}| \leq \epsilon$$

then we call this algoorithm an ϵ-approxiamtion algorithm.
Specifically, the samller ϵ is , the better the algorithm.

Theorem 5.3. If $P \neq NP$, then TSP has no polynomial-time ϵ-approximation algorithm.

Definition 5.1 (Triangle TSP)**.**
Triangle TSP is the TSP with additional constraints:

1. $d_i i = 0$,
2. $d_{ij} = d_{ji}$.
3. $\forall i, j, l, d_{ij} + d_{jk} \geq d_{ik}$.

Theorem 5.4. $\delta TSP \in NPC$.

Theorem 5.5 ($\frac{1}{2}$-Approximation Algorithm for δTSP)**.**
Consider the following algorithm:

Step 0 $G(V, E$ and $D = (d_{ij})$.

Step 1 Find MST T^*.

Step 2 Duplicate T^* to find a Eulerian walk.

Step 3 Find an embedded TSP tour τ by skipping intermediate nodes that was already visited before.

The above algorithm is a $\frac{1}{2}$ approximation algorithm.

The $\epsilon = \frac{1}{2}$ is improved to $\frac{1}{3}$ using the algorithm below.

Definition 5.2 (Weighted Matching)**.**
Suppose $G = (V, E)$ is an undirected graph such that there is an even number of nodes, and the graph is complete.
Weighted matching problem is to find $\frac{|V|}{2}$ edges with minimal cost so that every node in V is the endpoint of an edge.

It is known there is a algorithm that runs in $O(n^4)$ times.

Theorem 5.6 (Christofides' Algorithm)**.**

Step 0 $G = (V, E)$ adn $D = (d_{ij})$

Step 1 Find a minimum spanning tree T^* in G.
Let Q be the set of edges in the tree T^*.

Step 2 Let N be the set of nodes that have odd degree in T^*. For these nodes in N, find a set of edges $M \subset E$ solving the weighted matching problem.

- Consider graph $G' = (V, Q \cup M)$. Use previous algorithm to find the embedded TSP tour τ.

The above algorithm is a $\frac{1}{3}$-approximation algorithm.

5.6 Local Search

The combinatorial optimization problem is of the form: given F, c, $\min c(f) \text{s.t.} f \in F$.
Now we define, for any $f \in F$, define its neighbourhood $N(f) \subset F$.
The local search algorithm is as follows:

Step 0 $f = f_0$.

Step 1 $c(f_{k+1}) = \min c(f) \text{s.t.} f \in N(f_k)$

If $c(f_{k+1}) = c(f_k)$, stop, otherwise repeat Step 1.
This local search is guaranteed to find a local minimum, but not necessarily global minimum.

Definition 5.3 (Max Cut)**.**
Given a graph $G = (V, E)$, partition V into two sets S and T to maximize number of edges between S and T.

Theorem 5.7 (Approximation Algorithm for Max Cut)**.** *We start from any partition of V: (S, T) and perform local improvement steps below till it cannot be performed:*

- *If the cut (edges between S and T) can be increased by moving one node from S to T, or T to S, then we do so.*

This algorithm is a $\frac{1}{2}$-approximation algorithm.

5.7 FPTAS

NPC problems are equivalent under reduction, but behaviour can be different for approximation algorithms.
For $0-1$ knapsack problem, we have a ϵ-approximation algorithms for all ϵ, and the time taken for an ϵ behaves well in ϵ.

Definition 5.4 (Fully Polynomial Time Approximation Sheme)**.**
An FPTAS is a family of algorithms $\{A_\epsilon\}_{\epsilon>0}$ such that

- A_ϵ is an ϵ-approximation for P.
- Computation time is polynomial in L and $\frac{1}{\epsilon}$

where P is the problem and L is the length of bit model of an instance of P.

Recall knapsack problem. Suppose we transform the cost vector as such:

$$c' = \lfloor \frac{c}{K} \rfloor$$

and

$$K = \frac{\epsilon c_m ax}{n}$$

to get a integer instance for cost. We use the introduced algorithm for this modified instance and will we will have

$$A_\epsilon \geq (1 - \epsilon) OPT$$

and computation time is polynomial in n and $\frac{1}{n}$.