Part I

Integer and Combinatorial Programming

Chapter 1

Polyhedral Analysis

1.1 Polyhedral and Dimension

1.1.1 Polyhedral, Hyperplanes and Half-spaces

- A **polyhedron** is a set of the form $\{x \in \mathbb{R}^n | Ax \leq b\} = \{x \in \mathbb{R}^n | a^i x \leq b^i, \forall i \in M\}$, where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$
- A polyhedron $P \subset \mathbb{R}^n$ is **bounded** if there exists a constant K such that $|x_i| < K, \forall x \in P, \forall i \in [1, n]$, in this case the polyhedron is call **polytopes**
- The lower-bound of K is called **diagonal** denoted by d

1.1.2 Open, Close Sets: boundary and interior

- Denote $N_\epsilon = \{y \in \mathbb{R}^n | \|y - x\| < \epsilon\}$ as the **neighborhood** of $x \in \mathbb{R}^n$
- Given $S \subseteq \mathbb{R}^n$, x belongs to the **interior** of S, denoted by $int(S)$ if there is $\epsilon > 0$ such that $N_\epsilon(x) \leq S$
- S is said to be an **open set** iff $S = int(S)$
- x belongs to the **boundary** ∂S if $\forall \epsilon > 0$, $N_\epsilon(x)$ contains at least one point in S and a point not in S
- $x \in S$ belongs to the **closure** of S, denoted $cl(s)$ if $\forall \epsilon > 0$, $N_\epsilon(x) \cap S = \emptyset$ - S is called **closed** iff $S = cl(S)$
- In IP, LP, MIP, etc. we always work with close set. No "<" or ">"

1.1.3 Hyperplane and half-space

- A **hyperplane** is $\{x \in \mathbb{R}^n | a^T x = b\}$
- A **half-space** is $\{x \in \mathbb{R}^n | a^T x \leq b\}$

1.1.4 Dimension of Polyhedral

- A polyhedron P is **dimension** k, denoted $dim(P) = k$, if the maximum number of affinely independent points in P is $k + 1$
- A polyhedron $P \subseteq \mathbb{R}^n$ is **full-dimensional** if $dim(P) = n$
- Let:
- $M = \{1, 2, ..., m\}$
- $M^= = \{i \in M | a_i x = b_i, \forall x \in P\}$, i.e. the equality set
- $M^\leq = M \setminus M^=$, i.e. the inequality set
- Let $(A^=, b^=)$, $(A^\leq, b^\leq)$ be the corresponding rows of (A, b)
- If $P \subseteq \mathbb{R}^n$, then $dim(P) = n - rank(A^=, b^=)$
- To proof a constraint $(A^=, b^=)$ is an equality constraint, we need to proof all point in the closure of P satisfied the constraint, to proof it is not an equality constraint, we need to find one point that is not in the hyperplane.

1.1.5 Dimension and Rank

- $x \in P$ is called an **inner point** of P if $a^i x < b_i, \forall i \in M^\leq$
- $x \in P$ is called an **interior point** of P if $a^i x < b_i, \forall i \in M$

- Every nonempty polyhedron has at least one inner point
- A polyhedron has an interior point iff P is full-dimensional, i.e., there is no equality constraint

1.2 Face and Facet

1.2.1 Valid Inequalities and Faces

- The inequality denoted by (π, π_o) is called a **valid inequality** for P if $\pi x \leq \pi_0, \forall x \in P$
- Note that (π, π_0) is a valid inequality iff P lies in the half-space $\{x \in \mathbb{R}^n | Ax \leq b\}$
- If (π, π_0) is a valid inequality for P and $F = \{x \in P | \pi x = x_0\}$, F is called a **facet** of P and we say that (π, π_0)

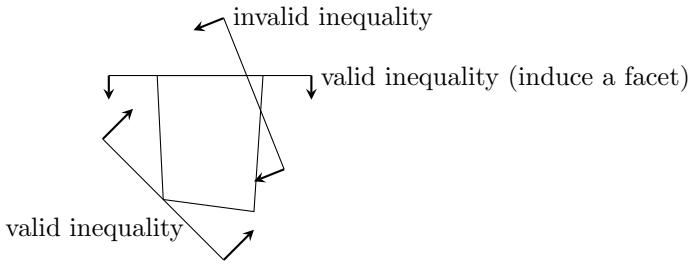

Figure 1.1: Example of valid/invalid inequality

represents or **defines** F
- A face is said to be **proper** if $F \neq \emptyset$ and $F \neq P$
- The face represented by (π, π_0) is nonempty iff $\max\{\pi x | x \in P\} = \pi_0\}$
- If the face F is nonempty, we say it **supports** P
- Let P be a polyhedron with equality set $M^=$. If

$$F = \{x \in P | \pi^T x = \pi_0\} \tag{1.1}$$

is not empty, then F is a polyhedron. Let

$$M^= \subseteq M_F^=, M_F^\leq = M \setminus M_F^= \tag{1.2}$$

then

$$F = \{x | a_i^T x = b_i, \forall i \in M_F^=, a_i^T x \leq b_i, \forall i \in M_f^\leq\} \tag{1.3}$$

1.2.2 Facet

- A face F is said to be a **facet** of P if $dim(F) = dim(P) - 1$
- Facets are all we need to describe polyhedral
- If F is a facet of P, then in any description of P, there exists some inequality representing F
- Every inequality that represents a face that is not a facet is unnecessary in the description of P - Every full-dimensional polyhedron P has a unique (up to scalar multiplication) representation that consists of one inequality representing each facet of P
- If $dim(P) = n - k$ with $k > 0$, then P is described by a maximal set of linearly independent rows of $(A^=, b^=)$, as well as one inequality representing each facet of P

1.2.3 Proving Facet

To prove an inequality $\sum_i a_i x_i \leq b_i$ is facet inducing for a D dimensional polyhedral, we need to prove there are D affinely independent vectors in $\sum_i a_i x_i = b_i$

1.2.4 Domination

$\Pi x \leq \Pi_0$ dominates $Mx \leq M_0$ if

$$\begin{cases} \Pi \geq \mu M, \mu > 0 \\ \Pi_0 \leq \mu M_0, \mu > 0 \\ (\Pi, \Pi_0) \neq (M, M_0) \end{cases} \tag{1.4}$$

Chapter 2

Branch and Bound

2.1 LP based Branch and Bound

2.1.1 Idea of Divide and Conquer

For each iteration, divide the feasible region of LP into two feasible parts and an infeasible part, solve the LP in those parts.

In this iteration, the original feasible region have been partition into three parts, where S_2 is infeasible for IP

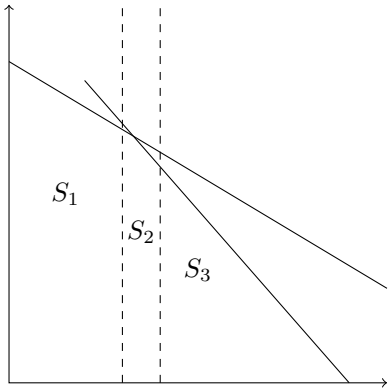

Figure 2.1: Divide and Conquer

because there is not integer point in it. We continue the iteration for S_1 and S_2. Each partition is suppose to give a new upper bound / lower bound and reduce the infeasible space.
If the temp optimal integer in S_1 is larger than the LP relaxation in S_3, we can cut S_3.
For each iteration, we use dual simple method, for the following two reasons:
- We can process new constraint very fast
- Always gives us a valid bound.

2.1.2 Relation Between LP Relaxation and IP

Let

$$Z_{IP} = \max_{x \in S} cx, \quad \text{where } s \text{ is a set of integer solutions} \tag{2.1}$$

$$Z_{LP} = \max cx, \quad \text{the LP relaxation of IP} \tag{2.2}$$

then

$$Z_{IP} = \max_{1 \le i \le k} \{\max_{x \in S_i} cx\} \tag{2.3}$$

$$\text{iff} \quad S = \bigcup_{1 \le i \le k} S_i \tag{2.4}$$

Notice that S_i don't need to be disjointed.
Important! (For maximization problem)
- Any feasible solution provides a lower bound L, which is also the *Prime Bound*

$$\hat{x} \in S \rightarrow Z_{IP} \ge c\hat{x} \tag{2.5}$$

- After branching, solving the LP relaxation over sub-feasible-region S_i produces an upper bound, which is also the *Dual Bound*, on each sub-problem
- If $u(S_i) \le L$, remove S_i
- LP can produce the first upper bound, but there might be possible to find other upper bound with other method (e.g. Lagrangian relaxation)

2.1.3 LP feasibility and IP(or MIP) feasibility

Solve the LP relaxation, one of the following things can happen
- LP is infeasible $\rightarrow$ MIP is infeasible
- LP is unbounded $\rightarrow$ MIP is infeasible or unbounded
- LP has optimal solution $\hat{x}$ and $\hat{x}$ are integer ($\hat{x} \in S$), $\rightarrow Z_{IP} = Z_{LP}$
- LP has optimal solution $\hat{x}$ and $\hat{x}$ are not integer ($\hat{x} \notin S$), now defines a new upper bound, $Z_{LP} \ge Z_{IP}$

If the first three happens, stop, if the fourth happens, we branch and recursively solve the sub-problems.

2.2 Terminology in Branch and Bound

- If we picture the sub-problems, they will form a **search tree** (typically a binary tree)
- Each node in the search tree is a **sub-problem**
- Eliminating a node is called **pruning**, we also stop considering its children
- A sub-problem that has not being processed is called a **candidate**, we keep a list of candidates

2.3 Bounding

Notice! this section is for maximization, if it is for minimization, reverse upper bound and lower bound.

2.3.1 Upper Bound

- Upper bound it the Prime bound. which means it has to be a feasible solution
- Some methods to get an upper bound:
- Rounding
- Heuristic
- Meta-heuristic

2.3.2 Lower Bound

- Lower Bound is the Dual bound,we can use LP relaxation to get it
- The tighter the better, LP is better

Algorithm 1 Branch and Bound

find a feasible solution as the initial Lower bound L
put the original LP relaxation in candidate list S
while $S \neq \emptyset$ **do**
    select a problem $\hat{S}$ from S
    solve the LP relaxation of $\hat{S}$ to obtain $u(\hat{S})$
    if LP is infeasible **then**
        $\hat{S}$ pruned by infeasibility
    else if LP is unbounded **then**
        $\hat{S}$ pruned by unboundness or infeasibility
    else if LP $u(\hat{S}) \leq L$ **then**
        $\hat{S}$ pruned by bound
    else if LP $u(\hat{S}) > L$ **then**
        if $\hat{x} \in S$ **then**
            $u(\hat{S})$ becomes new L, $L = u(\hat{S})$
        else if $\hat{x} \notin S$ **then**
            branch and add the new sub-problems to S
            if LP $u(\hat{S})$ is at current best upper bound **then**
                set $U = u(\hat{S})$
if Lower bound exists **then**
    find the optimal at L
else
    Infeasible

2.4 Branch and Bound Algorithm

2.5 The goal of Branching

- Divide the problem into easier sub-problems
- We want to chose the branching variables that minimize the sum of the solution times of the sub-problems
- If after branching the $u(S_i)$ changes a lot,
- I can find a good L first
- The branch may get worse than the current bound first
- Instead of solving the potential two branches for all candidates to optimality, solve a few iterations of the dual simplex, each iteration of pivoting yields an upper bound.

2.6 Choose Branching Variables

2.6.1 The Most Violated Integrality constraint

Pick the j of which $x_j - \lfloor \hat{x_j} \rfloor$ is closer to 0.5

2.6.2 Strong Branching

Select a few candidates (K), create all sub-problems for each of these variables, run a few dual simplex iterations to see the improved bounds, select the variable with the best bounds.
for variable $x_j \in K$, we branch and do a few iterations to find two reductions of gaps, i.e. D_j^+ and D_j^-,

2.6.3 pseudo-cost Branching

Pseudo-cost is an estimate of per-unit change in the objective function, for each variable

$$\begin{cases} P_j^+, & \text{bound reduction if rounded up} \\ P_j^-, & \text{bound reduction if rounded down} \end{cases} \tag{2.6}$$

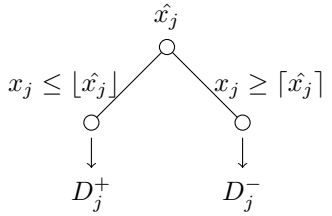

Figure 2.2: Strong Branching

define $f_j = x_j - \lfloor x_j \rfloor$

$$\begin{cases} D_j^+ = P_j^+(1 - f_j) \\ D_j^- = P_j^- f_j \end{cases} \tag{2.7}$$

2.7 Choose the Node to Branch

2.7.1 Update After Branching

For those variables in K find the
- $\max\{\min\{D_j^+, D_j^-\}\}$, or
- $\max\{\max\{D_j^+, D_j^-\}\}$, or
- $\max\{\frac{D_j^+ + D_j^-}{2}\}$, or
- $\max\{\alpha_1 \min\{D_j^+, D_j^-\} + \alpha_2 \max\{D_j^+, D_j^-\}\}$

to branch.

2.7.2 Branch on Important Variables First

Branch on variables that affects many decisions.

2.7.3 Some Search Strategy

- Best Bound First: select the node with the largest bound (good for closing the gap)
- Deep First: Good for finding Lower bound and easier to do dual simplex
- Mix: Start with "Deep First" until we find a good bound and do "Best Bound First"

2.8 Types of Branching

2.8.1 Traditional Branching

For $\hat{x} \notin S$, $\exists j \in N$ such that

$$\hat{x_j} - \lfloor \hat{x_j} \rfloor > 0 \tag{2.8}$$

Create two sub-problems

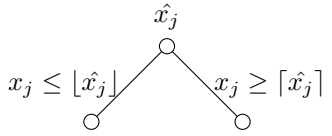

Figure 2.3: Traditional Branching

2.8.2 Constraint Branching

Use parallel constraints to branch, e.g.

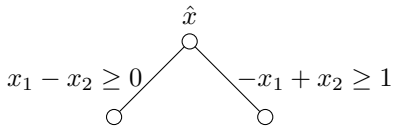

Figure 2.4: Traditional Branching

2.8.3 SOS

For SOS1, For SOS2 (using the first definition),

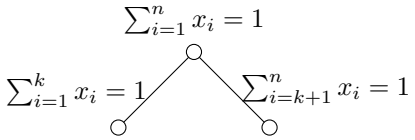

Figure 2.5: Traditional Branching

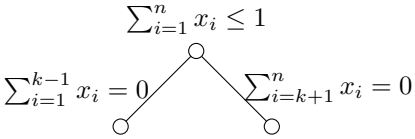

Figure 2.6: Traditional Branching

2.8.4 GUB

This is where $x_i \in \{0, 1\}$, at most one variable can be 1,

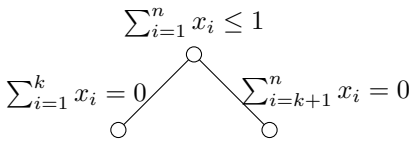

Figure 2.7: Traditional Branching

2.8.5 Ryan-Foster

Ryan-Foster is for Set covering problem. The typical model is

$$\min \quad \sum_{i \in C} x_i \tag{2.9}$$

$$\text{s.t.} \quad \sum_{i \in C} a_{ij} x_i \geq 1, \quad \forall j \in U \tag{2.10}$$

$$x_i \in \{0, 1\}, \quad \forall i \in C \tag{2.11}$$

Observation For any fractional solution, there are at least two elements (i, j) so that i and j are both partially covered by the same set S, but there is another set T that only covers i

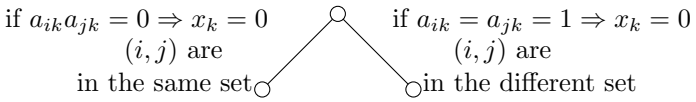

Figure 2.8: Traditional Branching

Chapter 3

Branch and Cut

3.1 Separation Algorithm

Basic idea is to separate the feasible region so that the current "solution" (which is an fractional solution) is not included in the feasible region.

3.1.1 Vertices Packing

The current solution is $\bar{x} \in [0,1]^n$, we have two options to do the separation:
Option 1 - find the maximum clique:
(This approach is as hard as the original problem)
denote

$$y_i = \begin{cases} 1, \text{ if } i \in C \\ 0, \text{ otherwise} \end{cases} \tag{3.1}$$

Find the maximum clique via:

$$\max \quad \sum \bar{x_i} y_i \tag{3.2}$$

$$\text{s.t.} \quad y_i + y_j \leq 1, \forall \{i,j\} \notin E \tag{3.3}$$

Option 2 - Heuristic:

Algorithm 2 Heuristic method to find a clique

1: find $v = \text{argmax}_{i \in V}\{\bar{x_i}\}, C = \{v\}$
2: **while** $u \in \text{argmax}_{i \in \cap_{i \notin C} N_{(i,j)} \notin C}\{\bar{x_1}\}$ exists **do**
3: C.add(u)
4: return C

If $\sum_{i \in C} \bar{x_i} > 1$ then add cut $\sum_{i \in C} x_i \leq 1$

3.1.2 TSP

When we have a solution, i.e. $\bar{x}$, perform the sub-tour searching algorithm, if there exists any sub-tour, add the corresponded constraint. That is the separation.

3.2 Optional v.s. Essential Inequalities

3.2.1 Valid (Optional) Inequalities

See Figure 3.1

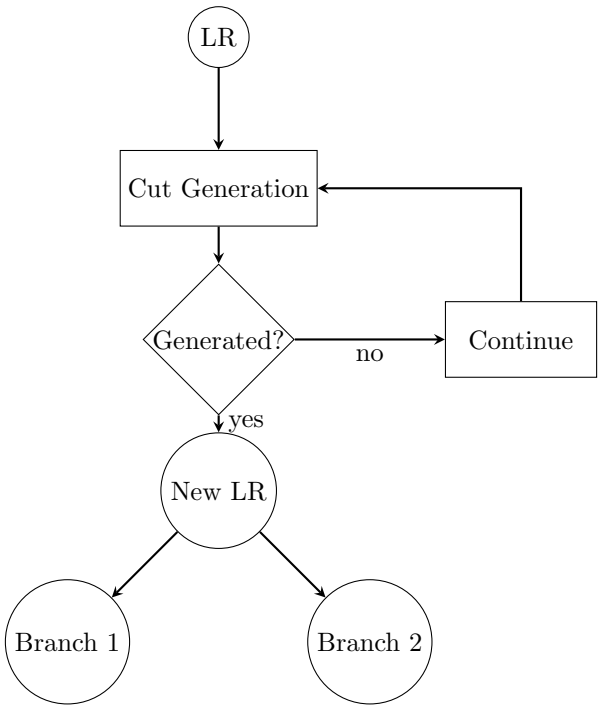

Figure 3.1: Branch and Cut for Optional Inequality

3.2.2 Essential Inequalities (Lazy Cuts)

See Figure 3.2

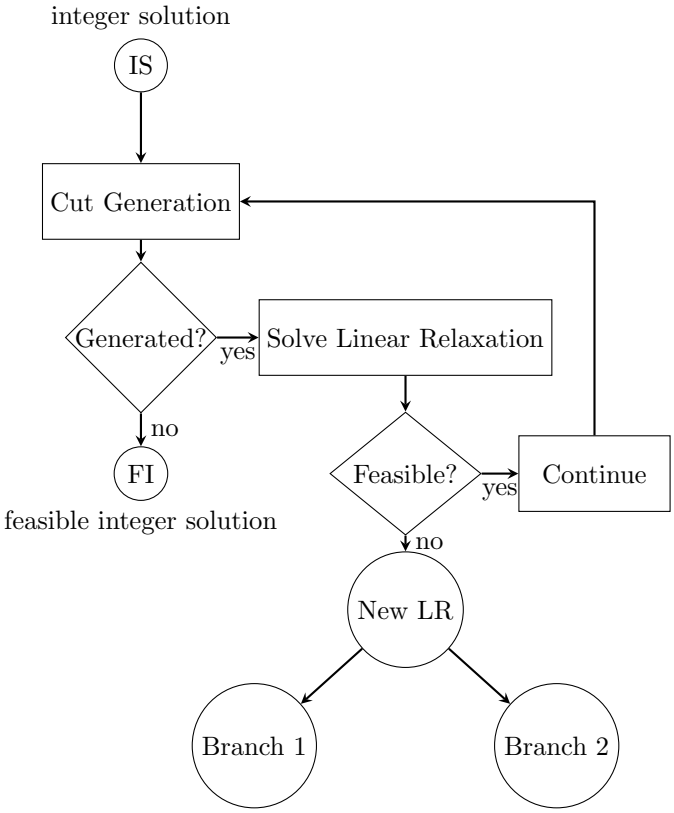

Figure 3.2: Branch and Cut for Essential Inequality

3.3 Chvatal-Gomory Cut

3.3.1 Chvatal-Gomory Rounding Procedure

For $x = P \cap \mathbb{Z}^n_+$, where $P = \{x \in \mathbb{R}^n_+ | Ax \leq b\}$, A is an mxn matrix with columns $\{a_1, ..., a_n\}$ and $u \in \mathbb{R}^n_+$
- The inequality

$$\sum_{j=1}^{n} ua_j x_j \leq ub \tag{3.4}$$

is valid
- Therefore the inequality

$$\sum_{j=1}^{n} \lfloor ua_j \rfloor x_j \leq ub \tag{3.5}$$

is valid
- Furthermore, the inequality

$$\sum_{j=1}^{n} \lfloor ua_j \rfloor x_j \leq \lfloor ub \rfloor \tag{3.6}$$

is valid.

3.3.2 Gomory Cutting Plane

For a IP problem

$$\begin{aligned} \max \quad & cx && (3.7)\\ \text{s.t.} \quad & Ax = b && (3.8)\\ & x \in \mathbb{B}^n && (3.9) \end{aligned}$$

let $\bar{x}$ be an optimal basic solution for the LR of P.

$$\bar{x} = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix} = \begin{bmatrix} x_B \\ x_N \end{bmatrix} \tag{3.10}$$

We have

$$\begin{aligned} & Bx_B + Nx_N = b && (3.11)\\ \Rightarrow \quad & x_B + B^{-1}Nx_N = B^{-1}b && (3.12)\\ \Rightarrow \quad & x_B + [\bar{a}_1, \bar{a}_2, ...]x_N = \bar{b} && (3.13)\\ \Rightarrow \quad & x_i + \sum_{j \in NB} \bar{a}_{ij} x_j = \bar{b}_i \quad \text{(for the } i\text{th row)} && (3.14) \end{aligned}$$

Assume that $x_i \in \{0, 1\}$, use CG-Procedure

$$x_i + \sum_{j \in NB} \lfloor \bar{a}_{ij} \rfloor x_j \leq \lfloor \bar{b}_i \rfloor \tag{3.15}$$

is a valid constraint for P, furthermore,

$$(\bar{b}_i - \sum_{j \in NB} \bar{a}_{ij} x_j) + \sum_{j \in NB} \lfloor \bar{a}_{ij} \rfloor x_j \leq \lfloor \bar{b}_i \rfloor \tag{3.16}$$

Move the item, we get a new Gomory Cutting Plane

$$\sum_{j \in NB} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor) x_j \geq \bar{b}_i - \lfloor \bar{b}_i \rfloor \tag{3.17}$$

Add this inequality to the LR, use the dual simplex method to do one pivot, we get a new solution. Use Gomory cutting plane iteratively and we can find the optimal solution for IP.

Chapter 4

Packing and Matching

4.1 Vertices Packing and Matching Formulation

Given a graph $G = (V, E)$, with $|V| = n$. A vertices packing solution is that no two neighboring vertices can be chosen at the same time.

$$PACK(G) = \{x \in \mathbb{B}^n | x_i + x_j \leq 1, \forall (i,j) \in E\} \tag{4.1}$$

Example. The following is an example:
The PACK of this graph is

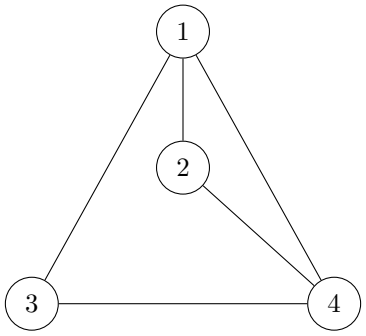

Figure 4.1: Example of vertices packing problem

$$PACK = conv\left(\begin{pmatrix}0\\0\\0\\0\end{pmatrix}, \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\\1\end{pmatrix}, \begin{pmatrix}0\\1\\1\\0\end{pmatrix}\right) \tag{4.2}$$

Given a graph $G = (V, E)$, denote $\delta(i)$ as the set of all the edges introduced to vertice $i \in V$. A matching solution is that no two edges introduced to the same vertice can be chosen at the same time.

$$MATCH(G) = \{\sum_{e \in \delta(i)} x_e \leq 1 | i \in V\} \tag{4.3}$$

4.2 Dimension of PACK(G)

The dimension of PACK, i.e. $dim(PACK(G))$ is (full-dimensional)

$$dim(PACK(G)) = |V| \tag{4.4}$$

To prove that $dim(PACK(G)) = |V|$, we need to find $|V| + 1$ affinely independent vectors.

Proof.

$$rank\left(\begin{bmatrix}0 & I_{|V|}\\ 1 & 1\end{bmatrix}\right) = |V| + 1 \tag{4.5}$$

Therefore, in PACK, $rank(A^{=}, b^{=}) = 0$ □

4.3 Clique

- A **clique** is a subset of a graph that in the clique every two vertices linked with each other (complete sub-graph).
- A **maximum clique** is a clique that any other vertice can not form a clique with all the points in this clique.

4.4 Inequalities and Facets of conv(VP)

Example:

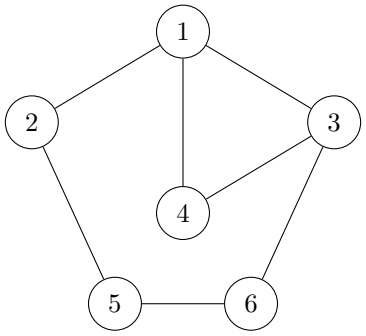

Figure 4.2: Example

4.4.1 Type 1 (Nonnegative Constraints)

$x_i \geq 0$ induce facets.

Proof:

$$rank\left(\begin{bmatrix}0 & 0\\ 0 & I_{|V|}\end{bmatrix}\right) = |V| + 1 \tag{4.6}$$

4.4.2 Type 2 (Neighborhood Constraints)

$x_i + x_j \leq 1$ is a valid constraint, but it **DOES NOT** always induce facet.

4.4.3 Type 3 (Odd Hole)

H is an odd hole if it contains circle of k nodes, such that k is odd and there is no cords. e.g. {1, 2, 5, 6, 3}. Then, the following inequality is valid,

$$\sum_{i \in H} x_i \leq \frac{|H| - 1}{2} \tag{4.7}$$

Odd Hole inequality **DOES NOT** always induce facets.
This inequality can be derived from Gomory cut.

4.4.4 Type 4 (Maximum Clique)

C is a maximum clique, then the following inequality is valid and induce a facet,

$$\sum_{i \in C} x_i \leq 1 \tag{4.8}$$

Proof:
First, if $C = V$

$$rank\left([I]\right) = |C| = |V| \tag{4.9}$$

Second, if C is a subset of V, for each vertice in $V \setminus C$, there should be at least one vertice in C that is not linked with it. Therefore for each vertice in C we can find a packing.

4.5 Gomory Cut in Set Covering

Consider a graph $G = (V, E)$, the covering problem is

$$\sum_{e \in \delta(i)} x_e \leq 1, i \in V, x_e \in \{0, 1\}, e \in E \tag{4.10}$$

For $T \subset V$, denote $\delta(i)$ as all edges induce to $i \in V$, denote $E(T) \subset E$ as all the edges linked between $(i, j), i \in T, j \in T$, therefore we have

$$\sum_{i \in T} \sum_{e \in \delta(i)} x_e \leq |T| \tag{4.11}$$

For edges linking $i \in T, j \in T$, count them twice, for edges linking $i \in T, j \notin T$, count them once.We can have a new constraint

$$2 \sum_{e \in E(T)} x_e + \sum_{e \in \delta(V \setminus T, T)} x_e \leq |T| \tag{4.12}$$

Perform the Gomory Cut, the following constraint is a valid:

$$\sum_{e \in E(T)} x_e \leq \lfloor \frac{|T|}{2} \rfloor \tag{4.13}$$

Chapter 5

Traveling Salesman Problem

5.1 TSP Formulation (Asymmetric)

Consider a Graph $G = \{A, N\}$
Denote:

$$x_{ij} = \begin{cases} 1, & \text{if goes from } i \text{ to } j \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

Dantzig-Fulkerson-Johnson Formulation:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{5.2}$$

$$\sum_{j\in N,(i,j)\in A} x_{ij} = 1 \tag{5.3}$$

$$\sum_{i\in N,(i,j)\in A} x_{ij} = 1 \tag{5.4}$$

$$\sum_{j\notin S,i\in S,(i,j)\in A} x_{ij} = 1 \text{ or } \sum_{i,j\in S,(i,j)\in A} x_{ij} \leq |S| - 1 \tag{5.5}$$

$$\forall S \subset N, S \neq \emptyset, 2 \leq |S| \leq n-1 \tag{5.6}$$

Miller-Tucker-Zemlin Formulation:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{5.7}$$

$$\sum_{j\in N,(i,j)\in A} x_i j = 1 \tag{5.8}$$

$$\sum_{i\in N,(i,j)\in A} x_i j = 1 \tag{5.9}$$

$$u_i - u_j + nx_{ij} \leq n-1 \quad i,j \in 2,...,n,(i,j) \in A \tag{5.10}$$

$$u_1 = 1 \tag{5.11}$$

$$2 \leq u_i \leq n, i \in N, i > 1 \tag{5.12}$$

5.2 Sub-tour Searching Algorithm

In the graph $G = (N, A)$, let $\bar{G} = (N, \bar{A})$ be the connected components of graph, where

$$\bar{G} = (G, \bar{A}), \bar{A} = \{(i,j) \in A | \bar{x_{ij}} = 1\} \tag{5.13}$$

denote

$$\bar{FS}(i) = \{(i,j) \in \bar{A}\} \tag{5.14}$$

Then the algorithm to find all sub-tour is the following:

Algorithm 3 Sub-tour Searching Algorithm

```
1:  K = ∅
2:  d_i = 0, ∀i ∈ N
3:  for i ∈ N do
4:      C = ∅
5:      Q = ∅
6:      if d_i == 0 then
7:          d_i = 1
8:          C = C ∪ {i}
9:          Q.append(i)
10:         while Q ≠ ∅ do
11:             v = Q.pop()
12:             for u ∈ FS(v) do
13:                 if d_u == 0 then
14:                     d_u = 1
15:                     C = C ∪ {u}
16:                     Q.append(u)
17:     K = K ∪ C
```

Chapter 6

Knapsack Problem

6.1 Knapsack Problem Formulation

Consider the knapsack set KNAP

$$conv(KNAP) = conv(\{x \in \mathbb{B}^n | \sum_{j \in N} a_j x_j \leq b\}) \tag{6.1}$$

in where
- $N = \{1, 2, ..., n\}$
- With out lost of generality, assume that $a_j > 0, \forall j \in N$ and $a_j < b, \forall j \in N$

6.2 Valid Inequalities for a Relaxation

For $P = \{x \in \mathbb{B}^n | Ax \leq b\}$, each row can be regard as a Knapsack problem, i.e. for row i

$$P_i = \{x \in \mathbb{B}^n | a_i^T x \leq b_i\} \tag{6.2}$$

is a relaxation of P, therefore,

$$P \subseteq P_i, \forall i = 1, 2, ..., m \tag{6.3}$$

$$P \subseteq \cap_{i=1}^m P_i \tag{6.4}$$

So any inequality valid for a relaxation of an IP is also valid for IP itself.

6.3 Cover and Extended Cover

A set $C \subseteq N$ is a cover if $\sum_{j \in C} a_j > b$, a cover C is minimal cover if

$$C \subseteq N | \sum_{j \in C} a_j > b, \sum_{j \in C \setminus k} a_j < b, \forall k \in C \tag{6.5}$$

For a cover C, we can have the cover inequality

$$\sum_{j \in C} x_j \leq |C| - 1 \tag{6.6}$$

The inequality is trivial considering the pigeonhole principle.
$C \subseteq N$ is a minimal cover, then $E(C)$ is defined as following:

$$E(C) = C \cup \{j \in N | a_j \geq a_i, \forall i \in C\} \tag{6.7}$$

is called an extended cover. Then we have,

$$\sum_{i \in E(C)} x_i \leq |C| - 1 \text{ dominates } \sum_{i \in C} x_i \leq |C| - 1 \tag{6.8}$$

and

$$\sum_{i\in E(C)} x_i \le |C|-1 \text{ dominates } \sum_{i\in E(C)} x_i \le |E(C)|-1 \tag{6.9}$$

Hereby we need to prove that $\sum_{i\in E(C)} x_i \le |C|-1$ is valid, by contradiction.

Proof:??? Suppose $x^R \in KNAP$, R is a feasible solution, Where

$$x_j^R = \begin{cases} 1, & \text{if } j \in R \\ 0, & \text{otherwise} \end{cases} \tag{6.10}$$

Then

$$\sum_{j\in E(C)} x_j^R \ge |C| \Rightarrow |R \cap E(C)| \ge |C| \tag{6.11}$$

therefore

$$\sum_{j\in R} a_j \ge \sum_{j\in R\cap E(C)} a_j \ge \sum_{j\in C} a_j > b \tag{6.12}$$

which means R is a cover, it is contradict to $\sum_{j\in E(C)} x_j^R \ge |C|$ so $x^R \notin KNAP$

6.4 Dimension of KNAP

$conv(KNAP)$ is full dimension, i.e. $dim(conv(KNAP)) = n$.

Proof: $0, e_j, \forall j \in N$ are $n+1$ affinely independent points in $conv(KNAP)$

6.5 Inequalities and Facets of conv(KNAP)

6.5.1 Type 1 (Lower Bound and Upper Bound Constraints):

- $x_k \ge 0$ is a facet of $conv(KNAP)$

Proof: $0, e_j, \forall j \in N \setminus k$ are n affinely independent points that satisfied $x_k = 0$

- $x_k \le 1$ is a facet iff $a_j + a_k \le b, \forall j \in N \setminus k$

Proof: $e_k, e_j + e_k, \forall j \in N \setminus k$ are n affinely independent points that satisfied $x_k = 1$

6.5.2 Type 2 (Extended Cover)

Order the variables so that $a_1 \ge a_2 \ge \cdots \ge a_n$, therefore $a_1 = a_{max}$

Let C be a cover with $\{j_1, j_2, \ldots, j_r\}$ where $j_1 < j_2 < \cdots < j_r$ so that $a_{j_1} \ge a_{j_2} \ge \cdots \ge a_{j_r}$

Let $p = \min\{j | j \in N \setminus E(C)\}$

Then

$$\sum_{j\in E(C)} x_j \le |C|-1 \tag{6.13}$$

is a facet of $conv(KNAP)$ if

- $C = N$

Proof:

$$R_k = C \setminus k, \forall k \in C = N \setminus k, \forall k \in N \tag{6.14}$$

have $|N|$ affinely independent vectors

- $E(C) = N$ and $\sum_{j\in C\setminus\{j_1,j_2\}} a_j + a_{max} \le b$

Proof: (j_1, j_2 are two heaviest elements in C)

$$S_k = C \setminus \{j_1, j_2\} \cup \{k\}, \forall k \in E(C) \setminus C \tag{6.15}$$

$R_k \cup S_k$ have $|C| + |E(C) \setminus C| = |E(C)| = |N|$ affinely independent vectors

- $C = E(C)$ and $\sum_{j\in C\setminus j_1} a_j + a_p \le b$)

Proof: (j_1 is the heaviest element in C, k is the lightest element outside extended cover)

$$T_k = C \setminus j_i \cup \{k\}, \forall k \in N \setminus E(C) \tag{6.16}$$

$R_k \cup T_k$ have $|N \setminus E(C)| + |E(C)| = |N \setminus C| + |C| = |N|$ affinely independent vectors
- $C \subset E(C) \subset N$ and $\sum_{j \in C \setminus \{j_1, j_2\}} a_j + a_{max} \leq b$ and $\sum_{j \in C \setminus j_1} a_j + a_p \leq b$
Proof: $S_k \cup T_k$ have $|E(C) \setminus C| + |N \setminus E(C)| = |N|$ affinely independent vectors

6.6 Lifting

6.6.1 Up Lifting

For KNAP problem

$$KNAP = \{x \in \mathbb{B}^n | \sum_j a_j x_j \leq b\} \tag{6.17}$$

For $P = conv(KNAP)$ denote

$$P_{k_1,k_2,...,k_m} \tag{6.18}$$
$$= conv(KNAP \cap \{x \in \mathbb{B}^n | x_{k_1} = x_{k_2} = \cdots = x_{k_m} = 0\}) \tag{6.19}$$

Therefore

$$P_{k_1,k_2,...,k_m} \tag{6.20}$$
$$= conv(KNAP \cap \{x \in \mathbb{B}^n | \sum_{j \in N \setminus \{k_1,k_2,...,k_m\}} a_j x_j \leq b\}) \tag{6.21}$$

The $C = N$ cover inequality for $P_{k_1,k_2,...,k_m}$ implies

$$\sum_{j \in N \setminus \{k_1,k_2,...,k_m\}} x_j \leq n - m - 1 \tag{6.22}$$

is a facet of $P_{k_1,k_2,...,k_m}$
The lifting process is to find a facet for $P_{k_1,k_2,...,k_{m-1}}$ using facet of $P_{k_1,k_2,...,k_m}$, i.e. find α_m for the following constraint to be a facet.

$$\alpha_m x_m + \sum_{j \in N \setminus \{k_1,k_2,...,k_m\}} x_j \leq n - m - 1 \tag{6.23}$$

If $x_m = 0$, $\alpha_m \geq 0$,
If $x_m = 1$, $\alpha_m \leq (n - m + 1) - \gamma$ where

$$\gamma = \max\{\sum_{j \in N \setminus \{k_1,k_2,...,k_m\}} x_j | x \in P_{k_1,k_2,...,k_{m-1}}, x_m = 1\} \tag{6.24}$$
$$= \max\{\sum_{j \in N \setminus \{k_1,k_2,...,k_m\}} x_j | \sum_{j \in N \setminus \{k_1,k_2,...,k_m\}} a_j x_j \leq b - a_m\} \tag{6.25}$$

Then let $\alpha_m = n - m + 1 - \gamma$, we uplifted a constraint. Repeat this procedure for $\{k_1, k_2, ..., k_m\}$ and finally we can find a family of facets for $conv(KNAP)$

6.6.2 Down Lifting

Similar to up lifting, we can perform the lifting in a different way.
Denote

$$P'_{k_1,k_2,...,k_m} \tag{6.26}$$
$$= conv(KNAP \cap \{x \in \mathbb{B}^n | x_{k_1} = x_{k_2} = \cdots = x_{k_m} = 1\}) \tag{6.27}$$

6.7 Separation of a Cover Inequality

$C \in N$ is a cover if $\sum_{i \in C} a_i > b$, let C be a minimal cover

$$\sum_{i \in C} x_i \leq |C| - 1 \tag{6.28}$$

$$\Rightarrow \quad |C| - \sum_{i \in C} x_i = \sum_{i \in C} (1 - x_i) \geq 1 \tag{6.29}$$

$$\tag{6.30}$$

let $\bar{x}$ be a fractional solution of $\{\sum_{i \in N} a_i x_i \leq b, x_i \in [0,1], i \in N\}$, find a cover C of which $\sum_{i \in C}(1 - \bar{x_i}) < 1$
Decision variable:

$$y_i = \begin{cases} 1, & \text{if } i \in C \\ 0, & \text{otherwise} \end{cases} \tag{6.31}$$

$$\min \quad \sum_{i \in N} (1 - \bar{x_i}) y_i = z \tag{6.32}$$

$$\text{s.t.} \quad \sum_{i \in N} a_i y_i \geq b + 1 \tag{6.33}$$

$$y_i \in \{0, 1\}, i \in N \tag{6.34}$$

if $z < 1$, then the cover cut associated with y is violation by $\bar{x}$

Chapter 7

Network Flow Problem

(Network Flow Problem is a special type of IP problem, its linear relaxation is the convex hull of the original problem.)

7.1 Shortest Path Problem

A graph $G = (A, N)$ is a directed graph
Denote:

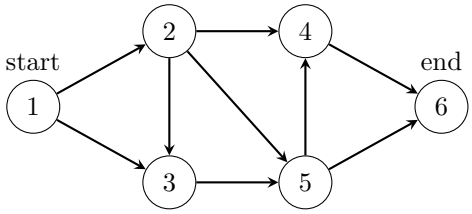

Figure 7.1: Example of directed graph

$$x_{ij} = \begin{cases} 1, & \text{if goes from } i \text{ to } j \\ 0, & \text{otherwise} \end{cases} \tag{7.1}$$

The shortest path problem can be formulated as the following:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{7.2}$$

$$\sum_{i\in N\setminus(\{S\}\cup\{E\}),(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = 0 \tag{7.3}$$

$$\sum_{i=\{S\},(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = 1 \tag{7.4}$$

$$\sum_{i=\{E\},(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = -1 \tag{7.5}$$

$$x_{ij} \in [0,1], (i,j) \in A \tag{7.6}$$

Although initially $x_{ij} \in [0, 1]$, in the optimized solution, $x \in \{0, 1\}$.

7.2 Maximum Flow Problem

$$\min \sum_{(i,j)\in A} F \tag{7.7}$$

$$\sum_{i\in N\setminus(\{S\}\cup\{E\}),(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = 0 \tag{7.8}$$

$$\sum_{i=\{S\},(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = F \tag{7.9}$$

$$\sum_{i=\{E\},(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = -F \tag{7.10}$$

$$l_{ij} \le x_{ij} \le u_{ij}, (i,j) \in A \tag{7.11}$$

In where F means the flow from source to target.

7.3 Minimum Cost Flow

The shortest path problem is a special case of Minimum Cost Flow Problem, which can be formulated as the following:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{7.12}$$

$$\sum_{i\in N\setminus(\{S\}\cup\{E\}),(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = 0 \tag{7.13}$$

$$\sum_{i=\{S\},(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = 1 \tag{7.14}$$

$$\sum_{i=\{E\},(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = -1 \tag{7.15}$$

$$x_{ij} \in [0,1], (i,j) \in A \tag{7.16}$$

7.4 Unimodularity

7.4.1 Unimodular Matrix and Total Unimodular Matrix

- A unimodular matrix M is a squared matrix, where $det(M) = 1$ or -1.
- Total unimodular matrix is a matrix where all its sub-matrices are unimodular matrix.

7.4.2 Importance of Unimodular Matrix

Let $M_{m\times m}$ be a unimodular matrix, if $b \in \mathbb{Z}^m$, the solution for $Mx = b$ is always integer.
Proof: By Cramer's Rule

$$x_i = \frac{detM_i}{detM} \tag{7.17}$$

in which M_i is matrix M replace ith column with b. Therefore $det(M_i)$ is integer. Also, $det(M) \neq 0$, so $det(M) = 1$ or $det(M) = -1$. Proved.

7.4.3 Structures of Total Unimodular Matrix

Structure 1: Matrix M that has only 1, -1, 0 enters and each column has at most 2 non-zeros is a TU matrix if it satisfies the following conditions:
We can split the rows in to tops and bottoms, such that for all columns j having 2 non-zeros
- If the non-zeros have the same sign, then one value should be in top and the other should be in bottom

- If the non-zeros have the different sign, then both of them should be in top or both of them should be in bottom
Structure 2: If all the columns in matrix M has only 0 or consecutive 1s (or -1s), matrix M is a TU matrix

7.4.4 Construct a New Unimodular Matrix

Let F be a unimodular matrix, then

$$\begin{bmatrix} F \\ I \end{bmatrix} \tag{7.18}$$

is a unimodular matrix, also

$$\begin{bmatrix} F & 0 \\ I & I \end{bmatrix} \tag{7.19}$$

is a unimodular matrix.