# Notes for Operations Research & More

Lan Peng, PhD Student

Department of Industrial and Systems Engineering
University at Buffalo, SUNY
lanpeng@buffalo.edu

December 27, 2019

December 27, 2019

# Contents

# Part I

# Linear Programming

# Chapter 1

# Formulation

## 1.1 Problem Manipulation

### 1.1.1 Inequalities and Equalities

An inequality can be transformed into an equation by adding or subtracting the nonnegative slack or surplus variable

$$\sum_{j=1}^{n} a_{ij}x_j \geq b_i \Rightarrow \sum_{j=1}^{n} a_{ij}x_j - x_{n+1} = b_i \tag{1.1}$$

or

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i \Rightarrow \sum_{j=1}^{n} a_{ij}x_j + x_{n+1} = b_i \tag{1.2}$$

Although it is not the practice, equality can be transformed into inequality too

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \Rightarrow \begin{cases} \sum_{j=1}^{n} a_{ij}x_j \leq b_i \\ \sum_{j=1}^{n} a_{ij}x_j \geq b_i \end{cases} \tag{1.3}$$

Also, in linear programming, we only care about close set, so we will not have $<, >$ in the formulation, we can use the following

$$\sum_{j=1}^{n} a_{ij}x_j > b_i \Rightarrow \sum_{j=1}^{n} a_{ij}x_j \geq b_i + \epsilon \tag{1.4}$$

where $\epsilon$ is a small number.

### 1.1.2 Minimization and Maximization

To convert a minimization problem into a maximization problem, we can use the following to define a new objective function

$$\min \sum_{j=1}^{n} c_j x_j = -\max \sum_{j=1}^{n} c_j x_j \tag{1.5}$$

### 1.1.3 Standard Form and Canonical Form

Standard Form

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j x_j && (1.6) \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} && (1.7) \\ & \mathbf{x} \geq \mathbf{0} && (1.8) \end{aligned}$$

Canonical Form

$$\min \quad \sum_{j=1}^{n} c_j x_j \tag{1.9}$$

$$\text{s.t.} \quad \mathbf{A}\mathbf{x} \le \mathbf{b} \tag{1.10}$$

$$\mathbf{x} \ge \mathbf{0} \tag{1.11}$$

## 1.2 Typical Problems

## 1.3 Formulation Skills

### 1.3.1 Absolute Value

Consider the following model statement:

$$\min \quad \sum_{j\in J} c_j |x_j|, \quad c_j > 0 \tag{1.12}$$

$$\text{s.t.} \quad \sum_{j\in J} a_{ij} x_j \gtreqless b_i, \quad \forall i \in I \tag{1.13}$$

$$x_j \quad \text{unrestricted}, \quad \forall j \in J \tag{1.14}$$

Modeling:

$$\min \quad \sum_{j\in J} c_j (x_j^+ + x_j^-), \quad c_j > 0 \tag{1.15}$$

$$\text{s.t.} \quad \sum_{j\in J} a_{ij} (x_j^+ - x_j^-) \gtreqless b_i, \quad \forall i \in I \tag{1.16}$$

$$x_j^+, x_j^- \ge 0, \quad \forall j \in J \tag{1.17}$$

### 1.3.2 A Minimax Objective

Consider the following model statement:

$$\min \quad \max_{k\in K} \sum_{j\in J} c_{kj} x_j \tag{1.18}$$

$$\text{s.t.} \quad \sum_{j\in J} a_{ij} x_j \gtreqless b_i, \quad \forall i \in I \tag{1.19}$$

$$x_j \ge 0, \quad \forall j \in J \tag{1.20}$$

Modeling:

$$\min \quad z \tag{1.21}$$

$$\text{s.t.} \quad \sum_{j\in J} a_{ij} x_j \gtreqless b_i, \quad \forall i \in I \tag{1.22}$$

$$\sum_{j\in J} c_{kj} x_j \le z, \quad \forall k \in K \tag{1.23}$$

$$x_j \ge 0, \quad \forall j \in J \tag{1.24}$$

### 1.3.3 A Fractional Objective

Consider the following model statement:

$$\min \quad \frac{\sum_{j\in J} c_j x_j + \alpha}{\sum_{j\in J} d_j x_j + \beta} \tag{1.25}$$

$$\text{s.t.} \quad \sum_{j\in J} a_{ij} x_j \gtreqless b_i, \quad \forall i \in I \tag{1.26}$$

$$x_j \geq 0, \quad \forall j \in J \tag{1.27}$$

Modeling:

$$\min \quad \sum_{j\in J} c_j x_j t + \alpha t \tag{1.28}$$

$$\text{s.t.} \quad \sum_{j\in J} a_{ij} x_j \gtreqless b_i, \quad \forall i \in J \tag{1.29}$$

$$\sum_{j\in J} d_j x_j t + \beta t = 1 \tag{1.30}$$

$$t > 0 \tag{1.31}$$

$$x_j \geq 0, \quad \forall j \in J \tag{1.32}$$

$$(t = \frac{1}{\sum_{j\in J} d_j x_j + \beta}) \tag{1.33}$$

For the following statement:

$$\min \quad z^P = \frac{\mathbf{c}^\top \mathbf{x} + d}{\mathbf{e}^\top \mathbf{x} + f} \tag{1.34}$$

$$\text{s.t.} \quad \mathbf{G}\mathbf{x} \leq \mathbf{h} \tag{1.35}$$

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1.36}$$

Modeling

$$\min \quad z^R = \mathbf{c}^\top \mathbf{y} + dz \tag{1.37}$$

$$\text{s.t.} \quad \mathbf{G}\mathbf{y} - \mathbf{h}z \leq 0 \tag{1.38}$$

$$\mathbf{A}\mathbf{y} - \mathbf{b}z = 0 \tag{1.39}$$

$$\mathbf{e}^\top \mathbf{y} + fz = 1 \tag{1.40}$$

$$z \geq 0 \tag{1.41}$$

### 1.3.4 A Range Constraint

Consider the following model statement:

$$\min \quad \sum_{j\in J} c_j x_j \tag{1.42}$$

$$\text{s.t.} \quad d_i \leq \sum_{j\in J} a_{ij} x_j \leq e_i, \quad \forall i \in I \tag{1.43}$$

$$x_j \geq 0, \quad \forall j \in J \tag{1.44}$$

Modeling:

$$\min \quad \sum_{j\in J} c_j x_j, \quad c_j > 0 \tag{1.45}$$

$$\text{s.t.} \quad u_i + \sum_{j\in J} a_{ij} x_j = e_i, \quad \forall i \in I \tag{1.46}$$

$$x_j \geq 0, \quad \forall j \in J \tag{1.47}$$

$$0 \leq u_i \leq e_i - d_i, \quad \forall i \in I \tag{1.48}$$

# Chapter 2

# Simplex Method

## 2.1 Basic Feasible Solutions and Extreme Points

**Definition 2.1.1** (Basic Feasible Solutions)**.** Consider the system $\{\mathbf{A_{m\times n}x = b_m, b_m \geq 0}\}$, suppose $rank(\mathbf{A}, \mathbf{b}) = rank(\mathbf{A}) = m$, we can rearrange the columns of $\mathbf{A}$ so that we have a partition of $\mathbf{A}$. Let $\mathbf{A} = \begin{bmatrix}\mathbf{B} & \mathbf{N}\end{bmatrix}$ where $\mathbf{B}$ is an $m \times m$ invertible matrix, and $\mathbf{N}$ is an $m \times (n-m)$ matrix. The solution $\mathbf{x} = \begin{bmatrix}\mathbf{x_B} \\ \mathbf{x_N}\end{bmatrix}$ to the equation $\mathbf{Ax = b}$, where $\mathbf{x_B} = \mathbf{B}^{-1}\mathbf{b}$ and $\mathbf{x_N} = \mathbf{0}$ is called **basic solution** of system. If $\mathbf{x_B} \geq \mathbf{0}$, it is called **basic feasible solution** or **B.F.S.**. If $\mathbf{x_B} > \mathbf{0}$ it is called **non-degenerate basic feasible solution**. For $\mathbf{x_B} \geq \mathbf{0}$, if some $x_j = 0$, those components are called **degenerated basic feasible solution**. $\mathbf{B}$ is called the **basic matrix**, $\mathbf{N}$ is called **nonbasic matrix**

**Theorem 2.1.** $\mathbf{x}$ *is an extreme point* $\iff$ $\mathbf{x}$ *is a basic feasible solution.*

*Proof.* This proof is lack of details. Denote $\mathcal{S}$ as feasible region. FIXME
($\Rightarrow$) First, Let $\mathbf{x}$ be a B.F.S., Suppose $\mathbf{x} = \lambda\mathbf{u} + (1-\lambda)\mathbf{v}$, for $\mathbf{u}, \mathbf{v} \in \mathcal{S}, \lambda \in (0, 1)$. Let $I = \{i : x_i > 0\}$ be the set of index where the inequality constraint are not tight. Then for $i \notin I$, $x_i = 0$, which implies $u_i = v_i = 0$. $\mathbf{u}, \mathbf{v} \in \mathcal{S} \Rightarrow \mathbf{Au = Av = b} \Rightarrow \mathbf{A(u - v) = 0} \Rightarrow \sum_{i=1}^{n}(u_i - v_i)a_i = 0$.

-

- if $i \notin I$ then $x_i = 0$, which implies $u_i = v_i = 0$ - $\because$ $\mathbf{Au = Av = b}$, $\therefore$ $\mathbf{A(u-v) = 0} \Rightarrow \sum_{i=1}^{n}(u_i - v_i)a_i = 0$, $\because u_i = v_i = 0$, for $i \notin I$, it implies $u_i = v_i$ for $i \in I$, Hence $u = v$, $x$ is E.P.

($\Leftarrow$) Second, suppose $\mathbf{x}$ is not B.F.S., i.e. $\{a_i : i \in I\}$ are linearly dependent.
Then there $\exists \mathbf{u} \neq \mathbf{0}, u_i = 0, i \notin I$ such that $\mathbf{Au = 0}$.
Hence, for a small $\epsilon$, $\mathbf{x} = \frac{1}{2}(\mathbf{x} + \epsilon\mathbf{u}) + \frac{1}{2}(\mathbf{x} - \epsilon\mathbf{u})$, $\mathbf{x}$ is not E.P. □

## 2.2 Simplex Method

### 2.2.1 Key to Simplex Method

#### Cost Coefficient

The cost coefficient can be derived from the following

$$\begin{aligned} z &= cx && (2.1)\\ &= c_B x_B + c_N x_N && (2.2)\\ &= c_B(B^{-1}b - B^{-1}Nx_N) + c_N x_N && (2.3)\\ &= c_B B^{-1}b - \sum_{j\in N}(c_B B^{-1}a_j - c_j)x_j && (2.4)\\ &= c_B B^{-1}b - \sum_{j\in N}(z_j - c_j)x_j && (2.5) \end{aligned}$$

We denote $z_0 = c_B B^{-1} b$, $z_j = c_B^{-1} a_j$, $\bar{b} = B^{-1} b$ and $y_j = B^{-1} a_j$ for all nonbasic variables.
The formulation can be transformed into

$$\begin{aligned} \min \quad & z = z_0 - \sum_{j \in N} (z_j - c_j) x_j && (2.6) \\ \text{s.t.} \quad & \sum_{j \in N} y_j x_j + x_B = \bar{b} && (2.7) \\ & x_j \geq 0, j \in N && (2.8) \\ & x_B \geq 0 && (2.9) \end{aligned}$$

In the above formulation, $z_j - c_j$ is the cost coefficient. If $\exists j$ and $z_j - c_j > 0$, it means the objective function can still be optimized. (If $\forall j$, $z_j - c_j \leq 0$, then $z \geq z_0$ for any feasible solution, $z$ is the optimal solution)

**Pivot**

After finding the most violated $z_j - c_j$, we find a variable, say $x_k$, where $z_k - c_k = \min\{z_j - c_j\}$ to be the variable leaving the basis.
If there are degenerated variables, we can perform different method to choose variable to enter basis.

**Minimum Ratio**

$$x_{B_i} = \bar{b}_i - y_{ik} x_k \geq 0 \tag{2.10}$$

Therefore we have the minimum ratio rule

$$x_k = \min_{i \in B} \{ \frac{\bar{b}_i}{y_{ik}}, y_{ik} > 0 \} \tag{2.11}$$

If for the that column all $y_{ik} \leq 0$, unbounded.

### 2.2.2 Simplex Method Algorithm

The pseudo-code of Simplex Method is given as following:

## 2.3 Tableau Method for Simplex Method

The followin is an example of using tableau to solve simplex method. Initial tableau:

| | $z$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | RHS |
|---|---|---|---|---|---|---|---|
| $z$ | 1 | 1 | 3 | 0 | 0 | 0 | 0 |
| $x_3$ | 0 | 1 | -2 | 1 | 0 | 0 | 0 |
| $x_4$ | 0 | -2 | $\boxed{1}$ | 0 | 1 | 0 | 4 |
| $x_5$ | 0 | 5 | 3 | 0 | 0 | 1 | 15 |

(2.12)

Last tableau:

| | $z$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | RHS |
|---|---|---|---|---|---|---|---|
| $z$ | 1 | 0 | 0 | 0 | $-\frac{12}{11}$ | $-\frac{7}{11}$ | $-\frac{153}{11}$ |
| $x_3$ | 0 | 0 | 1 | 1 | $\frac{13}{11}$ | $\frac{3}{11}$ | $\frac{97}{11}$ |
| $x_2$ | 0 | 1 | 0 | 0 | $\frac{5}{11}$ | $\frac{2}{11}$ | $\frac{50}{11}$ |
| $x_1$ | 1 | 0 | 0 | 0 | $-\frac{3}{11}$ | $\frac{1}{11}$ | $\frac{3}{11}$ |

(2.13)

- The optimal basic variables are $x_3$, $x_2$, $x_1$. The optimal basis is the columns in the initial tableau with correspond columns

$$B = \begin{pmatrix} \frac{13}{11} & \frac{3}{11} & \frac{97}{11} \\ \frac{5}{11} & \frac{2}{11} & \frac{50}{11} \\ -\frac{3}{11} & \frac{1}{11} & \frac{3}{11} \end{pmatrix} \tag{2.14}$$

**Algorithm 1** Simplex Method

```
Require: Given a basic feasible solution with basis $B$
Ensure: Optimal objective value $\min z = cx$
 1: Set $\mathbf{B}$ for basic variables, $\mathbf{N}$ for nonbasic variables
 2: $\mathbf{B} \leftarrow$ all slack variables
 3: $\mathbf{N} \leftarrow$ all variables excepts slack variables
 4: for $\forall j$ do
 5:     $z_j = c_B B^{-1} a_j = 0$
 6: end for
 7: while $\exists z_j - c_j > 0$ do
 8:     $z_j = wa_j - c_j = c_B B^{-1} a_j - c_j$
 9:     $z_k - c_k = \max_{j \in \mathbf{N}} \{z_j - c_j\}$
10:     $y_k = B^{-1} a_k$
11:     if $\exists y_{ik} > 0$ then
12:         $\theta_r = \min_{i \in \mathbf{B}} \{\theta_i = \frac{\bar{b}_i}{y_{ik}} : y_{ik} > 0\}$
13:         $\mathbf{B} \leftarrow \mathbf{B} \backslash \{k\}$
14:         $\mathbf{N} \leftarrow \mathbf{N} \cup \{k\}$
15:         $\mathbf{B} \leftarrow \mathbf{B} \cup \{r\}$
16:         $\mathbf{N} \leftarrow \mathbf{N} \backslash \{r\}$
17:     else
18:         Unbounded
19:     end if
20: end while
21: $x_B^* = B^{-1} b = \bar{b}$
22: $x_N = 0$
23: $z^* = c_B B^{-1} b = c_B \bar{b} \mathbf{a_{B_k}}$
```

- From the initial tableau, we can see the initial basis is built from slack variables $x_3$, $x_4$, $x_5$. The $B^{-1}$ is the correspond columns in final tableau.

$$B = \begin{pmatrix} 1 & -2 & 1 \\ 0 & 1 & -2 \\ 0 & 3 & 5 \end{pmatrix} \tag{2.15}$$

- The optimal basic variables are $x_3$, $x_2$, $x_1$. Find $c_B$ in the initial tableau.

$$c_B = \begin{pmatrix} 0 \\ 3 \\ 1 \end{pmatrix} \tag{2.16}$$

- Find $w = c_B B^{-1}$ from the final tableau, correspond to the slack variable.

$$w = c_B B^{-1} = \begin{pmatrix} 0 \\ -\frac{12}{11} \\ -\frac{7}{11} \end{pmatrix} \tag{2.17}$$

## 2.4 Artificial Variable

If some of the constraint is not in $\sum_{i=1}^{n} a_i x_i \leq 0$ form, we cannot add a positive slack variable. In this case, we add an artificial variable other than slack variable.

$$\sum_{i=1}^{n} a_i x_i \geq (or =) 0 \Rightarrow \sum_{j=1}^{n} a_i x_i + x_a = 0 \tag{2.18}$$

Notice that in an optimal solution, $x_a = 0$, otherwise it is not valid.
Artificial variables are only a tool to get the simplex method started.

### 2.4.1 Two-Phase Method

**Two-Phase Method**

For **Phase I:**
Solve the following program start with a basic feasible solution $x = 0, x_a = b$, i.e., the artificial variable forms the basis.

$$\begin{aligned} \min \quad & 1x_a && (2.19) \\ \text{s.t.} \quad & Ax + x_a = b && (2.20) \\ & x \geq 0 && (2.21) \\ & x_a \geq 0 && (2.22) \end{aligned}$$

If the optimal $1x_a \neq 0$, infeasible, stop. Otherwise proceed Phase II. For **Phase II:**
Remove the columns of artificial variables, replace the objective function with the original objective function, proceed to solve simplex method.

**Discussion**

**Case A:** $x_a \neq 0$
Infeasible.
**Case B.1:** $x_a = 0$ and all artificial variables are out of the basis
At the end of Phase I, we derive

| $x_0$ | $x_B$ | $x_N$ | $x_a$ | RHS |
|---|---|---|---|---|
| 1 | 0 | 0 | -1 | 0 |
| 0 | $I$ | $B^{-1}N$ | $B^{-1}$ | $B^{-1}b$ |

(2.23)

We can discard $x_a$ columns, (or we can leave it because it keeps track of $B^{-1}$), and then we do the Phase II

| $z$ | $x_B$ | $x_N$ | $RHS$ |
|---|---|---|---|
| 1 | 0 | $c_BB^{-1}N - c_N$ | $c_BB^{-1}b$ |
| 0 | $I$ | $B^{-1}N$ | $B^{-1}b$ |

(2.24)

**Case B.2:** Some artificial variables are in the basis at zero values
This is because of degeneracy. We pivot on those artificial variables, once they leave the basis, eliminate them.

### 2.4.2 Big M Method

### 2.4.3 Single Artificial Variable

## 2.5 Revised Simplex Method

### 2.5.1 Key to Revised Simplex Method

The procedure of Simplex Method is (almost) exactly the same as original simplex method. However, notice that we don't need to use $N$ so for the revised simplex method, we don't calculate any matrix related to $N$
The original matrix:

| $z$ | $x_B$ | $x_N$ | $RHS$ |
|---|---|---|---|
| 1 | 0 | $c_BB^{-1}N - c_N$ | $c_BB^{-1}b$ |
| 0 | $I$ | $B^{-1}N$ | $B^{-1}b$ |

(2.25)

The revised matrix:

| Basic Inverse | RHS |
|---|---|
| $w = c_BB^{-1}$ | $c_B\bar{b} = c_BB^{-1}b$ |
| $B^{-1}$ | $\bar{b} = B^{-1}b$ |

(2.26)

For each pivot iteration, calculate $z_j - c_j = wa_j - c_j = c_B B^{-1} a_j - c_j, \forall j \in N$, pivot rules are the same as simplex method, each time find a variable $x_k$ to enter basis

| $B^{-1}$ | RHS |
|---|---|
| $w$ | $c_B \bar{b}$ |
| $B^{-1}$ | $\bar{b}$ |

| $x_k$ |
|---|
| $z_k - c_k$ |
| $y_k$ |

(2.27)

Do the minimum ratio rule to find the variable $x_r$ to leave the basis

| $B^{-1}$ | RHS |
|---|---|
| $w$ | $c_B \bar{b}$ |
|  | $\bar{b}_1$ |
|  | $\bar{b}_2$ |
|  | ... |
| $B^{-1}$ | $\bar{b}_r$ |
|  | ... |
|  | $\bar{b}_m$ |

| $x_k$ |
|---|
| $z_k - c_k$ |
| $y_{1k}$ |
| $y_{2k}$ |
| ... |
| $y_{rk}$(pivot at here) |
| ... |
| $y_{mk}$ |

(2.28)

### 2.5.2 Comparison between Simplex and Revised Simplex

**Advantage of Revised Simplex**

- Save storage memory
- Don't need to calculate N (including $B^{-1}N$ and $c_B B^{-1} N$)
- More accurate because round up errors will not be accumulated

**Disadvantage of Revised Simplex**

- Need to calculate $wa_j$ for all $j \in N$ (in fact don't need to calculated it for the variable just left the basis)

**Computation Complexity**

| Method | Type | Operations |
|---|---|---|
| Simplex | $\times$ | $(m+1)(n-m+1)$ |
|  | $+$ | $m(n-m+1)$ |
| Revised Simplex | $\times$ | $(m+1)^2 + m(n-m)$ |
|  | $+$ | $m(m+1) + m(n-m)$ |

(2.29)

**When to use?**

- When $m >> n$, do revise simplex method on the dual problem
- When $m \simeq n$, revise simplex method is not as good as simplex method
- When $m << n$ perfect for revise simplex method.

### 2.5.3 Decomposition of B inverse

Let $B = \{a_{B_1}, a_{B_2}, ..., a_{B_r}, ..., a_{B_m}\}$ and $B^{-1}$ is known. If $a_{B_r}$ is replaced by $a_{B_k}$, then $B$ becomes $\bar{B}$. Which means $a_{B_r}$ enters the basis and $a_{B_k}$ leaves the basis.
Then $\bar{B}^{-1}$ can be represent by $B^{-1}$. Noting that $a_k = By_k$ and $a_{B_i} = Be_i$, then

$$\begin{aligned} \bar{B} &= (a_{B_1}, a_{B_2}, ..., a_{B_{r-1}}, a_k, a_{B_{r+1}}, a_m) && (2.30) \\ &= (Be_1, Be_2, ..., Be_{r-1}, By_k, Be_{r+1}, ..., Be_m) && (2.31) \\ &= BT && (2.32) \end{aligned}$$

where $T$ is

$$T = \begin{bmatrix} 1 & 0 & ... & 0 & y_{1k} & 0 & ... & 0 \\ 0 & 1 & ... & 0 & y_{2k} & 0 & ... & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & ... & 1 & y_{r-1,k} & 0 & ... & 0 \\ 0 & 0 & ... & 0 & y_{rk} & 0 & ... & 0 \\ 0 & 0 & ... & 0 & y_{r+1,k} & 1 & ... & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & ... & 0 & y_{mk} & 0 & ... & 1 \end{bmatrix} \tag{2.33}$$

and

$$E = T^{-1} = \begin{bmatrix} 1 & 0 & ... & 0 & \frac{-y_{1k}}{y_{rk}} & 0 & ... & 0 \\ 0 & 1 & ... & 0 & \frac{-y_{2k}}{y_{rk}} & 0 & ... & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & ... & 1 & \frac{-y_{r-1,k}}{y_{rk}} & 0 & ... & 0 \\ 0 & 0 & ... & 0 & \frac{1}{y_{rk}} & 0 & ... & 0 \\ 0 & 0 & ... & 0 & \frac{-y_{r+1,k}}{y_{rk}} & 1 & ... & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & ... & 0 & \frac{-y_{mk}}{y_{rk}} & 0 & ... & 1 \end{bmatrix} \tag{2.34}$$

For each iteration, i.e. one variable enters the basis and one leaves the basis, $\bar{B}^{-1} = T^{-1}B^{-1} = EB^{-1}$. Given that the first iteration starts from slack variables, the first basis $B_1$ is $I$, then we have

$$B_t^{-1} = E_{t-1}E_{t-2}\cdots E_2E_1I \tag{2.35}$$

Using $E$ in calculation can simplify the product of matrix where

$$cE = c_1, c_2, ..., c_m \begin{bmatrix} 1 & 0 & ... & g_1 & ... & 0 \\ 0 & 1 & ... & g_2 & ... & 0 \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & ... & g_m & ... & 1 \end{bmatrix} \tag{2.36}$$

$$= (c_1, c_2, ..., c_{r-1}, cg, c_{r+1}, ..., c_m) \tag{2.37}$$

and

$$Ea = \begin{bmatrix} 1 & 0 & ... & g_1 & ... & 0 \\ 0 & 1 & ... & g_2 & ... & 0 \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & ... & g_m & ... & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} \tag{2.38}$$

$$= \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{r-1} \\ 0 \\ a_{r+1} \\ \vdots \\ a_m \end{bmatrix} + a_r \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_{r-1} \\ g_r \\ g_{r+1} \\ \vdots \\ g_m \end{bmatrix} \tag{2.39}$$

$$= \bar{a} + a_r g \tag{2.40}$$

Then we can calculate $w$, $y_k$ and $\bar{b}$

$$w = c_B B^{-1} = c_B E_{t-1}E_{t-2}...E_2E_1 \tag{2.41}$$

$$y_k = B^{-1}a_k = E_{t-1}E_{t-2}...E_2E_1a_k \tag{2.42}$$

$$\bar{b} = B_{t+1}^{-1}b = E_tE_{t-1}E_{t-2}...E_2E_1b \tag{2.43}$$

## 2.6 Simplex for Bounded Variables

### 2.6.1 Bounded Variable Formulation

$$\begin{aligned} \min \quad & cx && (2.44)\\ \text{s.t.} \quad & Ax = b && (2.45)\\ & l \le x \le b && (2.46) \end{aligned}$$

Reason why we don't the following formulation

$$\begin{aligned} \min \quad & cx && (2.47)\\ \text{s.t.} \quad & Ax = b && (2.48)\\ & x - Ix_l = l && (2.49)\\ & x + Ix_u = u && (2.50)\\ & x \ge 0 && (2.51)\\ & x_l \ge 0 && (2.52)\\ & x_u \ge 0 && (2.53) \end{aligned}$$

is that this formulation increase the number of variable from $n$ to $3n$, and the number of constraint from $m$ to $m + 2n$, the size in increase significantly.

### 2.6.2 Basic Feasible Solution

Consider the system $Ax = b$ and $l \le x \le b$, where $A$ is a $m \times n$ matrix of rank $m$, the solution $\bar{x}$ is a **basic feasible solution** if $A$ can be partition into $[B, N_l, N_u]$ where the solution $x$ can be partition into $x = (x_B, x_{N_l}, x_{N_u})$, in which $\bar{x}_{N_l} = l_{N_l}$ and $\bar{x}_{N_u} = u_{N_u}$, therefore

$$\bar{x}_B = B^{-1}b - B^{-1}N_l x_{N_l} - B^{-1}N_u x_{N_u} \tag{2.54}$$

Furthermore, similar to definition of nonnegative variables, if $l_B \le x_B \le u_B$, $x_B$ is a basic feasible solution, if $l_B < x_B < u_B$, $x_B$ is a non-degenerate basic feasible solution.

### 2.6.3 Improving Basic Feasible Solution

The basic variables and the objective function can be derived as following:

$$\begin{aligned} x_B &= B^{-1}b - B^{-1}N_l x_{N_l} - B^{-1}N_u x_{N_u} && (2.55)\\ z &= c_B x_B + c_{N_l} x_{N_l} + c_{N_u} x_{N_u} && (2.56)\\ &= c_B(B^{-1}b - B^{-1}N_l x_{N_l} - B^{-1}N_u x_{N_u}) && (2.57)\\ &\quad + c_B x_B + c_{N_l} x_{N_l} + c_{N_u} x_{N_u} && (2.58)\\ &= c_B B^{-1} b + (c_{N_l} - c_B B^{-1} N_l) x_{N_l} && (2.59)\\ &\quad + (c_{N_u} - c_B B^{-1} N_u) x_{N_u} && (2.60)\\ &= c_B B^{-1} b - \sum_{j \in J_1} (z_j - c_j) x_j - \sum_{j \in J_2} (z_j - c_j) x_j && (2.61) \end{aligned}$$

$J_1$ is the set of variables at lower bound, $J_2$ is the set of the variables at upper bound.
Notice that the right-hand-side no longer provide $c_B B^{-1} b$ and $B^{-1}b$. For the variable entering the basis, find the variable with

$$\max\{\max_{j \in J_1}\{z_j - c_j\}, \max_{j \in J_2}\{c_j - z_j\}\} \tag{2.62}$$

to enter the basis
$\boxed{\textbf{Tip:}}$ "Most violated rule"

The minimum ratio rule is revised for bounded simplex

$$\Delta = \min\{\gamma_1, \gamma_2, u_k - l_k\} \tag{2.63}$$

$$\gamma_1 = \begin{cases} \min_{r\in J_1}\{\frac{\bar{b}_r - l_{B_r}}{y_{rk}} : y_{rk} > 0\} \\ \min_{r\in J_2}\{\frac{\bar{b}_r - l_{B_r}}{-y_{rk}} : y_{rk} < 0\} \\ \infty \end{cases} \tag{2.64}$$

$$\gamma_2 = \begin{cases} \min_{r\in J_1}\{\frac{u_{B_r} - \bar{b}_r}{-y_{rk}} : y_{rk} < 0\} \\ \min_{r\in J_2}\{\frac{u_{B_r} - \bar{b}_r}{y_{rk}} : y_{rk} > 0\} \\ \infty \end{cases} \tag{2.65}$$

**Tip:**
Use $l \leq x + \Delta \leq u$ to test the range of $\delta$, if it hits lower bound, it is called $\gamma_1$, if it hits upper bound, it is called $\gamma_2$.

## 2.7 Degeneracy and Cycling

### 2.7.1 Degeneracy

#### Degeneracy in Simplex Method

If the basic variable $x_B$ is not strictly $> 0$, i.e. if some basic variable equals to 0, we call it degenerate.

#### Degeneracy for Bounded Variables

If some basic variables are at their upper bound or lower bound, we call it degenerate.

### 2.7.2 Cycling

In the degenerate case, pivoting by the simplex rule does not always give a strict decrease in the objective function value, because it may have $b_r = 0$. It is possible that the tableau may repeat if we use the simplex rule.
Geometrically speaking, it means that at the same point - extreme point - it corresponds to more than one feasible solutions, so when we are pivoting, we stays at the same place.
In computer algorithm, we rarely care about cycling because the data in computer is not precise, it is very hard to get into cycling.

### 2.7.3 Cycling Prevent

#### Lexicographic Rule

- For entering variable, same as simplex rule
- For leaving variable, if there is a tie, choose the variable with the smallest $\frac{y_{r1}}{y_{rk}}$.

#### Bland's Rule

- For entering variable, choose the variable with smallest index where $z_j - c_j \leq 0$
- For leaving variable, if there is a tie, choose the variable with smallest index.

#### Successive Ratio Rule

- Select the pivot column as any column $k$ where $z_k - c_k \leq 0$
- Given $k$, select the pivot row $r$ as the minimum successive ratio row associated with column $k$.

In other words, for pivot columns where there is no tie in the usual minimum ratio, the successive ratio rule reduces to the simplex rule

## 2.8 Dual Simplex Method

Maintain dual feasibility, i.e. primal optimality, and complementary slackness and work towards primal feasibility.
$\boxed{\textbf{Tip:}}$ The RHS become new $z_j - c_j$, the old $z_j - c_j$ become new RHS. We are actually solving the dual problem.

## 2.9 As a Search Algorithm

### 2.9.1 Improving Search Algorithm

A simplex method is a search algorithm, for each iteration it finds a not-worse solution, which can be represented as:

$$x^t = x^{t-1} + \lambda_{t-1} d^{t-1} \tag{2.66}$$

Where
- $x^t$ is the solution of the $t$th iteration
- $\lambda_t$ is the step length of $t$th iteration
- $d^t$ is the direction of the $t$th iteration
For each iteration, it contains three steps:
- Optimality test
- Find direction
- Find the step length

### 2.9.2 Optimality Test

$$\begin{aligned} z &= cx && (2.67)\\ &= \begin{bmatrix} c_B & c_N \end{bmatrix} \begin{bmatrix} x_B \\ x_N \end{bmatrix} && (2.68)\\ &= c_B x_B + c_N x_N && (2.69)\\ \text{and } \because & Ax = b && (2.70)\\ & \therefore B x_B + N x_N = b, x_B \geq 0, x_N \geq 0 && (2.71)\\ & \therefore x_B = B^{-1} b - B^{-1} N x_N && (2.72)\\ z &= c_B B^{-1} b - c_B B^{-1} N x_N + c_N x_N && (2.73) \end{aligned}$$

for current solution $\hat{x} = \begin{bmatrix} \hat{x_B} \\ 0 \end{bmatrix}$, $\hat{z} = c_B B^{-1} b$, then

$$z - \hat{z} = \begin{bmatrix} 0 & c_N - c_B B^{-1} N \end{bmatrix} \begin{bmatrix} x_B \\ x_N \end{bmatrix} \tag{2.74}$$

The $c_N - c_B B^{-1} N$ is the reduced cost, for a minimized problem, if $c_N - c_B B^{-1} N > 0$ means $z - \hat{z} \geq 0$, it reaches the optimality because we cannot find a solution less than $\hat{z}$.

### 2.9.3 Find Direction

Suppose we choose $x_k$ as a candidate to pivot into Basis

$$x = \begin{bmatrix} B^{-1} b - B^{-1} a_k x_k \\ 0 + e_k x_k \end{bmatrix} = \begin{bmatrix} B^{-1} b \\ 0 \end{bmatrix} + \begin{bmatrix} -B^{-1} a_k \\ e_k \end{bmatrix} x_k \tag{2.75}$$

In this form, we can see: $x$ is the result after $t$th iteration, $\begin{bmatrix} B^{-1} b \\ 0 \end{bmatrix}$ is the result after $(t-1)$th iteration. $\begin{bmatrix} -B^{-1} a_k \\ e_k \end{bmatrix}$ is the iteration direction, $x_k$ is the step length.
The only requirement of $x_k$ is $r_k < 0$ where $r_k = c_k - z_k$ is reduce cost, which is the $k$th entry of $c_N - c_B B^{-1} N$.
Generally speaking, we usually take $r_k = \min\{c_j - z_j\}$ (which in fact can not guarantee the efficient of the algorithm.)

### 2.9.4 Find the Step Length

We need to guarantee the non-negativity, so for each iteration, we need to make sure $x \geq 0$. Which means

$$B^{-1}b - B^{-1}a_k x_k \geq 0 \tag{2.76}$$

Denote $B^{-1}b$ as $\bar{b}$, denote $B^{-1}a_k$ as $y_k$
If $y_k \leq 0$, we can have $x_k$ as large as infinite, which means unboundedness.
If $y_k > 0$ now we can use the minimum ratio to guarantee non-negativity.
**Remember** hit the bound, basic variable leave the basis and become non-basic variable.

### 2.9.5 Simplex Method Algorithm

### 2.9.6 Simplex Method Tableau

### 2.9.7 Simplex Method as a Search Algorithm

# Chapter 3

# Duality, Sensitivity and Relaxation

## 3.1 Duality

### 3.1.1 Dual Formulation

For any prime problem

$$\begin{aligned} \min \quad & cx && (3.1)\\ \text{s.t.} \quad & Ax \geq b && (3.2)\\ & x \geq 0 && (3.3) \end{aligned}$$

we can have a dual problem

$$\begin{aligned} \max \quad & wb && (3.4)\\ \text{s.t.} \quad & wA \leq c && (3.5)\\ & w \geq 0 && (3.6) \end{aligned}$$

### 3.1.2 Mixed Forms of Duality

For the following prime problem

$$\begin{aligned} \text{P(or D)} \quad \min \quad & c_1x_1 + c_2x_2 + c_3x_3 && (3.7)\\ \text{s.t.} \quad & A_{11}x_1 + A_{12}x_2 + A_{13}x_3 \geq b_1 && (3.8)\\ & A_{21}x_1 + A_{22}x_2 + A_{23}x_3 \leq b_2 && (3.9)\\ & A_{31}x_1 + A_{32}x_2 + A_{33}x_3 = b_3 && (3.10)\\ & x_1 \geq 0 && (3.11)\\ & x_2 \leq 0 && (3.12)\\ & x_3 \quad \text{unrestricted} && (3.13) \end{aligned}$$

The dual of the problem

$$\begin{aligned} \text{D(or P)} \quad \max \quad & w_1b_1 + w_2b_2 + w_3b3 && (3.14)\\ \text{s.t.} \quad & w_1A_{11} + w_2A_{21} + w_3A_{31} \leq c_1 && (3.15)\\ & w_1A_{12} + w_2A_{22} + w_3A_{32} \geq c_2 && (3.16)\\ & w_1A_{13} + w_2A_{23} + w_3A_{33} = c_3 && (3.17)\\ & w_1 \geq 0 && (3.18)\\ & w_2 \leq 0 && (3.19)\\ & w_3 \quad \text{unrestricted} && (3.20) \end{aligned}$$

In sum, the relation between primal and dual problems are listed as following

| | Minimization | | Maximization | |
|---|---|---|---|---|
| Var | $\geq 0$ | $\longleftrightarrow$ | $\leq 0$ | Cons |
| | $\leq 0$ | $\longleftrightarrow$ | $\geq 0$ | |
| | Unrestricted | $\longleftrightarrow$ | $=$ | |
| Cons | $\geq 0$ | $\longleftrightarrow$ | $\geq 0$ | Var |
| | $\leq 0$ | $\longleftrightarrow$ | $\leq 0$ | |
| | $=$ | $\longleftrightarrow$ | Unrestricted | |

### 3.1.3 Dual of the Dual is the Primal

For a primal problem (P)

$$(P) \quad \min \quad cx \tag{3.21}$$
$$\text{s.t.} \quad Ax \geq b \tag{3.22}$$
$$x \geq 0 \tag{3.23}$$

The dual problem (D) is

$$(D) \quad \max \quad wb \tag{3.24}$$
$$\text{s.t.} \quad wA \leq c \tag{3.25}$$
$$w \geq 0 \tag{3.26}$$

Rewrite the dual

$$\min \quad -b^T w^T \tag{3.27}$$
$$\text{s.t.} \quad -A^T w^T \geq -c^T \tag{3.28}$$
$$w^T \geq 0 \tag{3.29}$$

Find the dual of this problem

$$\max \quad x^T(-c^T) \tag{3.30}$$
$$\text{s.t.} \quad x^T(-A^T) \leq (-b^T) \tag{3.31}$$
$$x^T \geq 0 \tag{3.32}$$
$$\tag{3.33}$$

Rewrite the dual of the dual

$$(P) \quad \min \quad cx \tag{3.34}$$
$$\text{s.t.} \quad Ax \geq b \tag{3.35}$$
$$x \geq 0 \tag{3.36}$$

### 3.1.4 Primal-Dual Relationships

#### Weak Duality Property

Let $x_0$ be any feasible solution of a primal minimization problem,

$$Ax_0 \geq b, \quad x_0 \geq 0 \tag{3.37}$$

Let $x_0$ be any feasible solution of a dual maximization problem,

$$w_0 A \leq c, \quad w_0 \geq 0 \tag{3.38}$$

Therefore, we have

$$cx_0 \geq w_0 A x_0 \geq w_0 b \tag{3.39}$$

which is called the weak duality property. This property is for any feasible solution in the primal and dual problem. Therefore, any feasible solution in the maximization problem gives the lower bound of its dual problem, which is a minimization problem, vice versa. We use this to give the bounds in using linear relaxation to solve IP problem.

**Fundamental Theorem of Duality**

With regard to the primal and dual LP problems, one and only one of the following can be true.
- Both primal and dual has optimal solution $x^*$ and $w^*$, where $cx^* = w^*b$
- One problem has an unbounded optimal objective value, the other problem must be infeasible
- Both problems are infeasible.

**Strong Duality Property**

From KKT condition, we know that in order to make $x^*$ the optimal solution, the following condition should be met.
- Primal Optimal: $Ax^* \geq b$, $x^* \geq 0$
- Dual Optimal: $w^*A \leq c$, $w^* \geq 0$
- Complementary Slackness:

$$\begin{cases} w^*(Ax^* - b) = 0 \\ (c - w^*A)x^* = 0 \end{cases} \tag{3.40}$$

The first condition means the primal has an optimal solution, the second condition means the dual has an optimal solution. The third condition means $cx^* = w^*b$, which is also called **strong duality property**
$\boxed{\textbf{Tip:}}$ $w$ in the dual problem is the same as the $w = c_B B^{-1}$ in primal problem.

**Complementary Slackness Theorem**

Let $\boldsymbol{x^*}$ and $\boldsymbol{w^*}$ be any feasible solutions, they are optimal iff

$$(c_j - \boldsymbol{w^* a_j})x_j^* = 0, \quad j = 1, ..., n \tag{3.41}$$

$$w_i^*(\boldsymbol{a^i x^*} - b_i) = 0, \quad i = 1, ..., m \tag{3.42}$$

In particular

$$x_j^* > 0 \Rightarrow \boldsymbol{w^* a_j} = c_j \tag{3.43}$$

$$\boldsymbol{w^* a_j} < c_j \Rightarrow x_j^* = 0 \tag{3.44}$$

$$w_i^* > 0 \Rightarrow \boldsymbol{a^i x^*} = b_i \tag{3.45}$$

$$\boldsymbol{a^i x^*} > b_i \Rightarrow w_i^* = 0 \tag{3.46}$$

It means, if in optimal solution a variable is positive (has to be in the basis), the correspond constraint in the other problem is tight. If the constraint in one problem is not tight, the correspond variable in the other problem is zero.

**Use Dual to Solve the Primal**

in the dual problem, we solved some $w$ which is positive, we can know that the correspond constraint in primal is tight, furthermore we can solve the basic variables from those tight constraints, which becomes equality and we can solve it using Gaussian-Elimination.

### 3.1.5 Shadow Price

**Shadow Price under Non-degeneracy**

Let $B$ be an optimal basis for primal problem and the optimal solution $x^*$ is non-degenerated.

$$z = c_B B^{-1} b - \sum_{j \in N}(z_j - c_j)x_j = w^* b - \sum_{j \in N}(z_j - c_j)x_j \tag{3.47}$$

therefore

$$\frac{\partial z^*}{\partial b_i} = c_B B_i^{-1} = w_i^* \tag{3.48}$$

$w^*$ is the shadow prices for the right-hand-side vectors. We can also regard it as the **incremental cost** of producing one more unit of the $i$th product. Or $w^*$ is the **fair price** we would pay to have an extra unit of the $i$th product.

**Shadow Price under Degeneracy**

For shadow price under degeneracy, the $w^*$ may not be the true shadow price, for it may not be the right basis. In this case, the partial differentiation may not be valid, for component $b_i$, if $x_i = 0$ and $x_i$ is a basic variable, we can't find the differentiation.

## 3.2 Sensitivity

### 3.2.1 Change in the Cost Vector

**Case 1: Nonbasic Variable**

$c_B$ is not affected, $z_j = c_B B^{-1} a_j$ is not changed, say nonbasic variable cost coefficient $c_k$ changed into $c_k^{'}$. For now $z_k - c_k \leq 0$, if $z_k - c_k^{'}$ is positive, $x_k$ must into the basis, the optimal value changed.Otherwise stays at the same.

**Case 2: Basic Variable**

If $c_{B_t}$ is replaced by $c_{B_t}^{'}$, then $z_j^{'} - c_j$ is

$$z_j^{'} - c_j = c_B^{'} B^{-1} a_j - c_j = (z_j - c_j) - (c_{B_t}^{'} - c_{B_t}) B^{-1} a_{B_t} \tag{3.49}$$

for $j = k$, it is a basic variable, therefore original $z_k - c_k = 0$, $B^{-1} a_k = 1$. Hence $z_k^{'} - c_k = c_k^{'} - c_k \Rightarrow z_k^{'} - c_k^{'} = 0$. The basis stays the same. The optimal solution updated as $c_B^{'} B^{-1} b = c_B B^{-1} b + (c_{B_t}^{'} - c_{B_t}) B^{-1} b_{B_t}$.

### 3.2.2 Change in the Right-Hand-Side

If $b$ is replaced by $b^{'}$, then $B^{-1}b$ is replaced by $B^{-1}b^{'}$. If $B^{-1}b^{'} \geq 0$, the basis remains optimal. Otherwise, we perform dual simplex method to continue.

### 3.2.3 Change in the Matrix

**Case 1: Changes in Activity(Variable) Vectors for Nonbasic Columns**

If a nonbasic column $a_j$ is replaced by $a_j^{'}$, then $z_j = c_B B^{-1} a_j$ is replaced by $z_j^{'} = c_B B^{-1} a_j^{'}$, if new $z_j^{'} - c_j \leq 0$, the basis stays optimal basis, the optimal value is the same because $c_B$ stays the same.

**Case 2: Changes in Activity(Variable) Vectors for Basic Columns**

If a basic columns changed, it means $B$ and $B^{-1}$ changed, and every column changed. We can do this in two steps:
- step I: add a new column with $a_j^{'}$
- step II: remove the original column $a_j$

If in step I the new variable can enter basis, i.e. $z_j^{'} - c_j \leq 0$, let it enter the basis and eliminate the original column directly (because at this time the original column leave the basis the nonbasic variable is 0); otherwise, if the new column can not form a new basis, treat $x_j$, the original variable as an artificial variable.

**Add a New Activity(Variable)**

Suppose we add a new variable $x_{n+1}$ and $c_{n+1}$ and $a_{n+1}$ respectively. Calculate $z_{n+1} - c_{n+1}$ to determine if the new variable enters the basis, if not, remains the same optimal solution, otherwise, continue on to find a new optimal solution.

**Add a New Constraint**

This is the basic of Branch-and-Cut/Bound, also, we can perform dual simplex method after we add a new constraint(cut)

## 3.3 Relaxation

### 3.3.1 Why Rounding Can be Bad - IP Example

Rounding can be bad because the optimal of IP can be far away from optimal of LP. For example,

$$\begin{aligned}
\max \quad & z = x_1 + 0.64x_2 && (3.50)\\
\text{s.t.} \quad & 50x_1 + 31x_2 \leq 250 && (3.51)\\
& 3x_1 - 2x_2 \geq -4 && (3.52)\\
& x_1, x_2 \geq 0 \quad \text{(for LP)} && (3.53)\\
& x_1, x_2 \in Z^+ \quad \text{(for IP)} && (3.54)
\end{aligned}$$

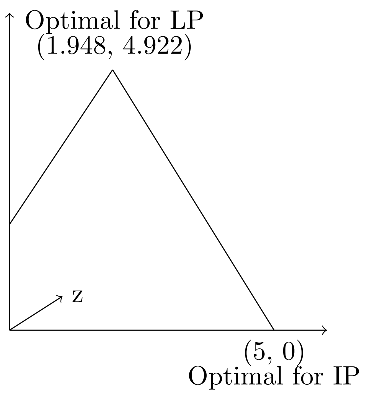


Figure 3.1: Optimal solution for LP / IP

### 3.3.2 Why Rounding Can be Bad - QAP example

Rounding can make the LP useless. For example, for QAP problem, the IP model is

$$\begin{aligned}
\min \quad & z = \sum_{i\in D}\sum_{s\in O} c_i s x_i s + \sum_{i\in D}\sum_{j\in D}\sum_{s\in O}\sum_{t\in O} w_{ij}^{st} y_{ij}^{st} && (3.55)\\
\text{s.t.} \quad & \sum_{i\in D} x_{is} = 1, \quad s \in D && (3.56)\\
& \sum_{s\in O} x_{is} = 1, \quad i \in D && (3.57)\\
& x_{is} \in \{0,1\}, \quad i \in D, s \in O && (3.58)\\
& y_{ij}^{st} \geq x_{is} + x_{jt} - 1, \quad i \in D, j \in D, s \in O, t \in O && (3.59)\\
& y_{ij}^{st} \geq 0, \quad i \in D, j \in D, s \in O, t \in O && (3.60)\\
& y_{ij}^{st} \leq x_{is}, \quad i \in D, j \in D, s \in O, t \in O && (3.61)\\
& y_{ij}^{st} \leq x_{jt}, \quad i \in D, j \in D, s \in O, t \in O && (3.62)
\end{aligned}$$

We can get the optimal solution for LP supposing $\forall i, s \quad x_{is} \in [0,1]$

$$\begin{aligned}
& x_{is} = \frac{1}{|D|}, \quad i \in D, s \in O && (3.63)\\
& y_{ij}^{st} = 0, \quad i \in D, j \in D, s \in O, t \in O && (3.64)
\end{aligned}$$

### 3.3.3 IP and Convex Hull

For IP problem

$$Z_{IP} \quad \max \quad z = cx \tag{3.65}$$
$$\text{s.t.} Ax \leq b \tag{3.66}$$
$$x \in Z^n \tag{3.67}$$

In feasible region $S = \{x \in Z^n, Ax \leq b\}$ , the optimal solution $Z_{IP} = \max\{cx : x \in S\}$.
Denote $conv(S)$ as the convex hull of $S$ then

$$Z_{IP}(S) = Z_{IP}(conv(S)) \tag{3.68}$$

### 3.3.4 Local Optimal and Global Optimal

Let

$$Z_s = \min\{f(x) : x \in S\} \tag{3.69}$$
$$Z_t = \min\{f(x) : x \in T\} \tag{3.70}$$
$$S \subset T \tag{3.71}$$

then

$$Z_t \leq Z_s \tag{3.72}$$

**Notice** that if $x_T^* \in S$ then $x_S^* = x_T^*$, to generalized it,
We have

$$\begin{cases} x_T^* \in \arg\min\{f(x) : x \in T\} \\ x_T^* \in S \end{cases} \tag{3.73}$$
$$\Rightarrow x_T^* \in \arg\min\{f(x) : x \in S\} \tag{3.74}$$

Especially for IP, we can take the LP relaxation as $T$ and the original feasible region of IP as $S$, therefore, if we find an optimal solution from LP relaxation $T$ which is also a feasible solution of $S$, then it is the optimal solution for IP $(S)$

### 3.3.5 LP Relaxation

To perform the LP relaxation, we expand the feasible region

$$x \in \{0, 1\} \rightarrow x \in [0, 1] \tag{3.75}$$
$$y \in Z^+ \rightarrow y \geq 0 \tag{3.76}$$

If we have $Z_LP(s) = conv(s)$ then

$$LP(s) : x \in R_+^n : Ax \leq b \tag{3.77}$$

The closer $LP(s)$ is to $conv(s)$ the better. Interestingly, we only need to know the convex in the direction of $c$.
There are several formulation problem have the property of $Z_{LP}(s) = conv(s)$, such as:
- Assignment Problem
- Spawning Tree Problem
- Max Flow Problem
- Matching Problem

# Chapter 4

# Decomposition Principle

# Chapter 5

# Ellipsoid Algorithm

# Chapter 6

# Projective Algorithm

# Chapter 7

# Interior-Point Algorithm

# Part II

# Graph and Network Theory

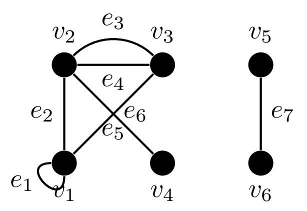
# Chapter 8

# Graphs and Subgraphs

## 8.1 Graph

**Definition 8.1.1** (Graph)**.** A **graph** G consists of a finite set $V(G)$ on vertices, a finite set $E(G)$ on edges and an **incident relation** than associates with any edge $e \in E(G)$ an unordered pair of vertices not necessarily distinct called **ends**.

**Example.** The following graph



can be represented as

$$
\begin{aligned}
V = V(G) &= \{v_1, v_2, v_3, v_4, v_5, v_6\} && (8.1)\\
E = E(G) &= \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\} && (8.2)\\
e_1 = v_1v_2, \quad e_2 = v_2v_4,& \quad \dots && (8.3)
\end{aligned}
$$

**Definition 8.1.2** (Loop, Parallel, Simple Graph)**.** An edge with identical ends is called a **loop**, Two edges having the same ends are said to be **parallel**, A graph without loops or parallel edges is called **simple graph**

**Definition 8.1.3** (Adjacent)**.** Two edges of a graph are **adjacent** if they have a common end, two vertices are **adjacent** if they are jointed by an edge.

## 8.2 Graph Isomorphism

## 8.3 The Adjacency and Incidence Matrices

## 8.4 Subgraph

**Definition 8.4.1** (Subgraph)**.** Given two graphs $G$ and $H$, $H$ is a **subgraph** of $G$ if $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ and an edge has the same ends in $H$ as it does in $G$. Furthermore, if $E(H) \neq E(G)$ then $H$ is a proper subgraph.

**Definition 8.4.2** (Spanning)**.** A subgraph $H$ on $G$ is **spanning** if $V(H) = V(G)$

**Definition 8.4.3** (Vertex-induced, Edge-induced)**.** For a subset $V^{'} \subseteq V(G)$ we define an **vertex-induced** subgraph $G[V^{'}]$ to be the subgraph with vertices $V^{'}$ and those edges of $G$ having both ends in $V^{'}$. The **edge-induced** subgraph $G[E^{'}]$ has edges $E^{'}$ and those vertices of $G$ that are ends to edges in $E^{'}$.

**Notice:** If we combine node-induced or edge-induced subgraphs $G(V^{'})$ and $G(V-V^{'})$, we cannot always get the entire graph.

## 8.5 Degree

**Definition 8.5.1** (Degree)**.** Let $v \in V(G)$, then the **degree** of $v \in V(G)$ denote by $d_G(v)$ is defines to be the number of edges incident of $v$. Loops counted twice.

**Theorem 8.1.** *For any graph $G = (V, E)$*

$$\sum_{v\in V} d(v) = 2|E| \tag{8.4}$$

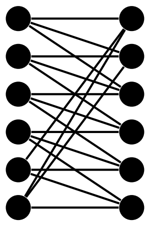
*Proof.* $\forall$ edge $e = uv$ with $u \neq v$, $e$ is counted once for $u$ and once for $v$, a total of two altogether. If $e = uu$, a loop, then it is counted twice for $u$. □

**Problem 8.1.** Explain clearly, what is the largest possible number of vertices in a graph with 19 edges and all vertices of degree at least 3. Explain why this is the maximum value.

**Solution.** The maximum number is 12.

*Proof.* First we prove 12 vertices is possible, then we prove 13 vertices is not possible

- The following graph contains 12 vertices and 18 edges, each vertex has a degree of 3.



- For 13 vertices and each vertex has a degree of at least 3 will require at least

$$2|E| = \sum_{v\in V} d(v) \geq 3 \times |N| = 3 \times 13 \Rightarrow |E| \geq 19.5 > 19 \tag{8.5}$$

  edges, i.e., 13 vertices is not possible.

□

**Corollary 8.1.1.** *Every graph has an even number of odd degree vertices.*

*Proof.*

$$V = V_E \cup V_O \Rightarrow \sum_{v\in V} d(v) = \sum_{v\in V_E} d(v) + \sum_{v\in V_O} d(v) = 2|E| \tag{8.6}$$

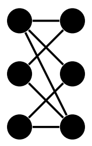
□

## 8.6 Special Graphs

**Definition 8.6.1** (Complete Graph)**.** A **complete** graph $K_n(n \geq 1)$ is a simple graph with $n$ vertices and with exactly one edge between each pair of distinct vertices.

**Definition 8.6.2** (Cycle)**.** A **cycle** graph $C_n(n \geq 3)$ consists of $n$ vertices $v_1, ...v_n$ and $n$ edges $\{v_1, v_2\}, \{v_2, v_3\}, ...\{v_{n-1}, v_n\}$

**Definition 8.6.3** (Wheel)**.** A **wheel** graph $W_n(n \geq 3)$ is a simple graph obtains by adding one vertex to the cycle graph $C_n$, and connecting this new vertex to all vertices of $C_n$

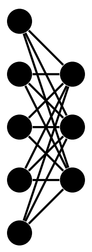


**Definition 8.6.4** (Bipartite Graph)**.** A simple graph is said to be **bipartite** if the vertex set can be expressed as the union of two disjoint non-empty subsets $V_1$ and $V_2$ such that every edges has one end in $V_1$ and another end in $V_2$

**Example.** Here is an example for bipartite graph

**Definition 8.6.5** (Complete Bipartite Graph)**.** The **complete bipartite graph** $K_{mn}$ is the bipartite graph $V_1$ containing $m$ vertices and $V_2$ containing $n$ vertices such that each vertiex in $V_1$ is adjacent to every vertex in $V_2$

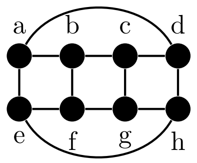
**Example.** Here is an example for $K_{53}$



**Theorem 8.2.** *A graph G is bipartite iff every cycle is even.*

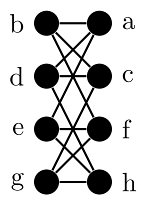
*Proof.* (⇒) If the graph $G$ is bipartite, by definition, the vertices of graph can be partition into two groups, that within the group there is no connection between vertices. Therefore, for each cycle, the odd index of vertices and even index of vertices has to be choose alternatively from each groups. Therefore the cycle has to be even.
(⇐) Prove by contradiction. A graph can be connected or not connected.
If $G$ is connected and has at least two vertices, for an arbitrary vertex $v \in V(G)$, we can calculate the minimum number of edges between the other vertices $v'$ and $v$ (i.e., length, denoted by $l(v', v)$), for all the vertices that has odd length to $v$, assign them to set $V_1$, for the rest of vertices (and $v$), assign to set $V_2$. Assume that $G$ is not bipartite, which means there are at least one edge between distinct vertices in set $V_1$ or set $V_2$, without lost of generality, assume that edge is $uw$, $u, w \in V_1$. For all vertices in $V_1$ there is an odd length of path between the vertex and $v$, therefore, there exists an odd $l(u, v)$, and an odd $l(w - v)$. The length of cycle $l(u, w, v) = 1 + l(u, v) + l(w, v)$, which is an odd number, it contradict with the prerequisite that all cycles are even, which means the assumption that $G$ is not bipartite is incorrect, $G$ should be bipartite.
If $G$ is not connected. Then $G$ can be partition into a set of disjointed subgraphs which are connected with at least two vertices or contains only one vertex. For the component that has more that one vertices, we already proved that it has to be bipartite. For the subgraph $G_i \subset G, i = 1, 2, ..., n$, the vertices can be partition into $V_{i1} \in V(G_i)$ and $V_{i2} \in V(G_i)$, where $V_{i1} \cap V_{i2} = \emptyset$, the union of those subgraphs are bipartite too because $V_1 = \cup_{i=1}^{n} V_{i1} \in V(G)$ and $V_2 = \cup_{i=1}^{n} V_{i2} \in V(G)$ satisfied the condition of bipartite. For the subgraph that has one one vertices, those vertices can be assigned into either $V_1$ or $V_2$. □

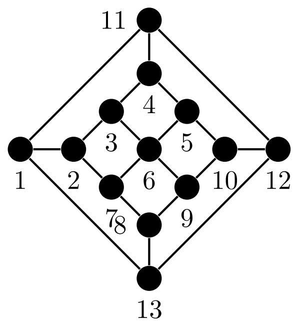
**Example.** The following graph is bipartite, it only contains even cycles.
We can rearrange the graph to be more clear as following



The vertices of graph $G$ can be partition into two sets, $\{a, c, f, h\}$ and $\{b, d, e, g\}$

**Example.** The following graph is not bipartite
The cycle $c = v_1 v_1 1 v_4 v_3 v_2$ have odd number of vertices.





## 8.7 Directed Graph

**Definition 8.7.1.** A graph $G = (V, E)$ is called directed if for each edge $e \in E$, there is a **head** $h(e) \in V$ and a **tail** $t(e) \in V$ and the edges of $e$ are precisely $h(e)$ and $t(e)$, denoted $e = (t(e), h(e))$

**Definition 8.7.2.** We call directed graphs **digraphs**, we call edges in a digraph are called **arcs**, and vertices in a digraph **nodes**

**Definition 8.7.3.** Similar as in the undirected case we have walks, traces, paths and cycles in digraphs.

**Definition 8.7.4.** A vertex $v \in V$ is **reachable** from a vertex $u \in V$ if there is a $(u, v)$-dipath. If at the same time $u$ is reachable from $v$, they are **strongly connected**

**Definition 8.7.5.** A digraph is strongly connected if every pair of vertices are strongly connected.

**Definition 8.7.6.** A digraph is **strict** if it has no loops and whenever $e$ and $f$ are parallel, $h(e) = t(f)$

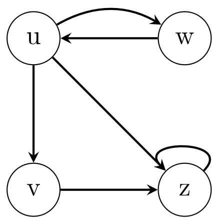
**Definition 8.7.7.** For a vertex $v$ in a digraph $D$, the **indegree** of $v$ in $D$, denoted by $d^+(v)$ is the number of arcs of $D$ having head $V$. The **outdegree** of $v$ is denoted by $d^-(v)$ is the number of arcs of $D$ having tail $v$.

Let $D = (V, A)$ be a digraph with no loops a vertex-arc **incident matrix** for $D$ is a $(0, 1, -1)$ matrix $N$ with rows indexed by $V = \{v_1, ..., v_n\}$ and column indexed by $A = \{e_1, ..., e_m\}$ and where entry $(i, j)$ in the matrix $n_{ij}$ is

$$n_{ij} = \begin{cases} 1, & \text{if } v_i = h(e_j) \\ -1, & \text{if } v_i = t(e_j) \\ 0, & \text{otherwise} \end{cases} \tag{8.7}$$

$$\begin{bmatrix} -1 & 0 & -1 & -1 & 1 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{8.8}$$

## 8.8 Sperner's Lemma



# Chapter 9

# Paths, Trees, and Cycles

## 9.1 Walk

**Definition 9.1.1** (walk)**.** A **walk** in a graph $G$ is a finite sequence $w = v_0e_1v_1e_2...e_kv_k$, where for each $e_i = v_{i-1}v_i$ the edge and its ends exists in $G$. We say that walk $v_0$ to $v_k$ on $(v_0, v_k)$-walk.

**Example.**

$$w = v_2e_4v_3e_4v_2e_5v_3 \tag{9.1}$$

is a walk, or $(v_2, v_3)$-walk

**Definition 9.1.2** (origin, terminal, internal, length)**.** For $(v_0, v_k)$-walk, The vertices $v_0$ and $v_k$ are called the **origin** and the **terminal** of the walk w, $v_1..v_{k-1}$ are called **internal** vertices. The integer $k$ is the **length** of the walk. Length of $w$ equals to the number of edges.

We can create a reverse walk $w^{-1}$ by reversing $w$.

$$w^{-1} = v_ke_kv_{k-1}e_{k-1}...e_2v_1 \tag{9.2}$$

(The reverse walk is guaranteed to exist because it is an undirected graph)
Given two walks $w$ and $w'$ we can create a third walk denoted by $ww'$ by concating $w$ and $w'$. The new walk's origin is the same as terminal.

## 9.2 Path and Cycle

**Definition 9.2.1** (trail)**.** A **trail** is a walk with no repeating edges. e.g., $v_3e_4v_2e_5v_3$

**Definition 9.2.2** (path)**.** A **path** is a trail with no repeating vertices. e.g., $v_3e_4v_2$

**Notice:** Paths $\subseteq$ Trails $\subseteq$ Walks

**Definition 9.2.3** (closed, cycle)**.** A path is **closed** if it has positive length and its origin and terminal are the same. e.g., $v_1e_2v_2e_4v_3e_3v_1$. A closed trail where origin and internal vertices are distinct is called a **cycle** (The only time a vertex is repeated is the origin and terminal)

**Definition 9.2.4** (even/odd cycle)**.** A cycle is **even** if it has a even number of edges otherwise it is **odd**.

**Problem 9.1.** Prove that if $C_1$ and $C_2$ are cycles of a graph, then there exists cycles $K_1, K_2, ..., K_m$ such that $E(C_1)\Delta E(C_2) = E(K_1) \cup E(K_2) \cup ... \cup E(K_m)$ and $E(K_i) \cap E(K_j) = \emptyset, \forall i \neq j$. (For set $X$ and $Y$, $X\Delta Y = (X - Y) \cup (Y - X)$, and is called the symmetric difference of $X$ and $Y$)

*Proof.* Proof by constructing $K_1, K_2, ...K_m$. Denote

$$C_1 = v_{11}e_{11}v_{12}e_{12}v_{13}e_{13}...v_{1n}e_{1n}v_{11} \tag{9.3}$$

$$C_2 = v_{21}e_{21}v_{22}e_{22}v_{23}e_{23}...v_{2k}e_{2k}v_{21} \tag{9.4}$$

Assume both cycle start at the same vertice, $v_{11} = v_{12}$. (If there is no intersected vertex for $C_1$ and $C_2$, just simply set $K_1 = C_1$ and $K_2 = C_2$)
The following algorithm can give us all $K_j, j = 1, 2, ..., m$ by constructing $E(C_1)\Delta E(C_2)$. Also, the complexity is $O(mn)$, which makes the proof doable.
Now we prove that $K_i \cap K_j = \emptyset, \forall i \neq j$. For each $K_j$, it is defined by two $(v_o, v_t)$-paths in the algorithm. From

**Algorithm 2** Find $K_1, K_2, ...K_m$ by constructing $E(C_1)\Delta E(C_2)$

```
Require: Graph $G$, cycle $C_1$ and $C_2$
Ensure: $K_1, K_2, ...K_m$
 1: Initial, $K \leftarrow \emptyset$, $j = 1$
 2: Set temporary storage units, $v_o \leftarrow v_{11}$, $v_t \leftarrow \emptyset$
 3: for $i = 1, 2, ..., n$ do
 4:   if $e_{1i} \in C_2$ then
 5:     if $v_o \neq v_{1i}$ then
 6:       $v_t \leftarrow v_{1i}$
 7:       concate $(v_o, v_t)$-path $\subset C_1$ and $(v_o, v_t)$-path $\subset C_2$ to create a new $K_j$
 8:       Append $K$ with $K_j$, $K \leftarrow K \cup K_j$
 9:       Reset temporary storage unit. $v_o \leftarrow v_{1(i+1)}$ (or $v_{11}$ if $i = n$), $v_t \leftarrow \emptyset$
10:     else
11:       $v_o \leftarrow v_{1(i+1)}$ (or $v_{11}$ if $i = n$)
12:     end if
13:   end if
14: end for
```

the algorithm we know that all the edges in $(v_o, v_t)$-path in $C_1$ are not intersecting with $C_2$, because if the edge in $C_1$ is intersected with $C_2$, either we closed the cycle $K_j$ before the edge, or we updated $v_o$ after the edge (start a new $K_j$ after that edge). By definition of cycle, all the $(v_o, v_t)$-path that are subset of $C_1$ are not intersecting with each other, as well as all the $(v_o, v_t)$-path that are subset of $C_2$. Therefore, $K_i \cap K_j = \emptyset, \forall i \neq j$. □

**Definition 9.2.5** (connected vertices)**.** Two vertices $u$ and $v$ in a graph are said to be **connected** if there is a path between $u$ and $v$.

**Definition 9.2.6** (component)**.** Connectivity between vertices is an equivalence relation on $V(G)$, if $V_1, ...V_k$ are the corresponding equivalent classes then $G[V_1]...G[V_k]$ are **components** of G. If graph has only one component, then we say the graph is connected. A graph is connected iff every pair of vertices in G are connected, i.e., there exists a path between every pair of vertices.

**Problem 9.2.** If $G$ is a simple graph with at least two vertices, prove that $G$ has two vertices with the same degree.

*Proof.* A simple graph can only be connected or not connected.

- If $G$ is connected, i.e., for all vertices, the degree is greater than 0. Also the graph is simple, for a graph with $|N|$ vertices, the degree of each vertex is less or equal to $|N| - 1$ (cannot have loop or parallel edge). For $|N|$ vertices, to make sure there is no two vertices that has same degree, it will need $|N|$ options for degrees, however, we only have $|N| - 1$ option. According to pigeon in holes principle, there has to be at least two vertices with the same degree.

- If $G$ is not connected, i.e., the graph has more than one component. One of the following situation will happen:
  - For all components, each component contains only one vertex. Since we have at least two vertices, which means there are at least two component that has only one vertex. For those vertices, at least two vertices has the same degree as 0.
  - At least one component has more than one vertices. In this situation, we can find a component that has more than one vertices as a subgraph $G'$ of the graph $G$. That $G'$ is a connected simple graph by definition. We have already proved that a connected simple graph has two vertices with the same degree, which means $G$ has two vertices with the same degree.

□

## 9.3 Tree and forest

**Definition 9.3.1** (acyclic graph)**.** A graph is called **acyclic** if it has no cycles

**Definition 9.3.2** (forest, tree)**.** A acyclic graph is called a **forest**. A connected forest is called a **tree**.

**Theorem 9.1.** *Prove that $T$ is a tree, if $T$ has exactly one more vertex than it has edges.*

*Proof.*

1. First we prove for any tree $T$ that has at least two vertices, there has to be at least one leaf, i.e., now we prove that we can find $u$ with degree of 1. Proof by constructing algorithm. (In fact we can prove that there are at least two leaves.)
   The above algorithm is guaranteed to have an end because a tree is acyclic by definition

**Algorithm 3** Find one leaf in a tree

```
Require: $d(u) = 1$
Ensure: A tree $T$ has at least one vertex
 1: Let $u$ and $v$ be any distinct vertex in a tree $T$
 2: Let $p$ be the path between $u$ and $v$
 3: while $d(u) \neq 1$ do
 4:   if $d(u) > 1$ then
 5:     Let $n(u)$ be the set of neighboring vertices of $u$
 6:     In $n(u)$, find a $u'$ that the edge between $u$ and $u'$, denoted by $e$, $e \notin p$
 7:     $u \leftarrow u'$
 8:     $p \leftarrow p \cup e$
 9:   end if
10: end while
```

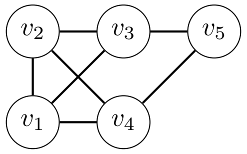
2. Then, if we remove one leaf in the tree, i.e., we remove an edge and a vertex, where that vertex only connects to the edge we removed. One of the following situations will happen:
   (a) Situation 1: The remaining of $T$ is one vertex. In this case, $T$ has two vertices an one edge. (Exactly one more vertex than it has edges)
   (b) Situation 2: The remaining of $T$ is another tree $T^{'}$ (removal of edges will not change acyclic and connectivity), where $|V(T)| = |V(T^{'})| + 1$ and $|E(T)| = |E(V^{'}| + 1$. (one edge and one vertex has been removed)
3. Do the leaf removal process recursively to $T^{'}$ if Situation 2 happens until Situation 1 happens.

□

## 9.4 Spanning tree

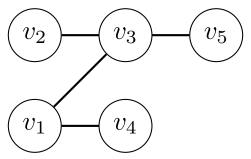
**Definition 9.4.1** (spanning tree)**.** A subgraph T of G is a **spanning tree** if it is spanning ($V(T) = V(G)$) and it is a tree.

**Example.** In the following graph
This is a spanning tree



**Problem 9.3.** Prove that if $T_1$ and $T_2$ are spanning trees of $G$ and $e \in E(T_1)$, then there exists a $f \in E(T_2)$, such that $T_1 - e + f$ and $T_2 + e - f$ are both spanning trees of $G$.

*Proof.* One of the following situation has to happen:



1. If for given $e \in E(T_1)$, $\exists f = e \in E(T_2)$, then $T_1 - e + f = T_1$, $T_2 + e - f = T_2$ are both spanning trees of $G$
2. If for given $e \in E(T_1)$, $e \notin E(T_2)$, the following will find an edge $f$ that $T_1 - e + f$ and $T_2 + e - f$ are both spanning trees of $G$.
   (a) $T_1$ is a spanning tree, removal of $e \in E(T_1)$ will disconnect the spanning tree into two components (by definition of spanning tree), denoted by $G_1 \subset G$ and $G_2 \subset G$, by definition, $V(G_1)$ and $V(G_2)$ is a partition of $V(G)$.
   (b) Add $e$ into $T_2$. We can proof that by adding an edge into a tree will create exactly one cycle, denoted by $C$, $e \in E(C)$.
   (c) For $C$, since it is a cycle and one end of $e$ is in $V(G_1)$, the other end of $e$ is in $V(G_2)$, there has to be at least two edges (can be more) that has one end in $V(G_1)$ and the other end in $V(G_2)$, denote the set of those edges as $E \subset E(C)$, one of those edges is $e \in E$
   (d) Choose any $f \in E$ and $f \neq e$, for that $f$, $T_1 - e + f$ and $T_2 + e - f$ are both spanning trees of $G$.
   (e) Prove that $T_1 - e + f$ is a spanning tree
      i. $T_1 - e + f$ have the same set of vertices as $T_1$, therefore it is spanning.
      ii. It is connected both within $G_1$ and $G_2$, for $f$, one end is in $V(G_1)$, the other end is in $V(G_2)$ therefore $T_1 - e + f$ is connected.
      iii. $T_1 - e + f$ have the same number of edges as $T_1$, which is $|T_1| - 1$, therefore $T_1 - e + f$ is a tree. (We have proven the connectivity in the previous step.)
      iv. $T_1 - e + f$ is spanning, connected, a tree, therefore it is a spanning tree.
   (f) Prove that $T_2 + e - f$ is a spanning tree
      i. $T_2 + e - f$ have the same set of vertices as $T_2$, therefore it is spanning.
      ii. $T_2$ is connected, adding an edge will not break connectivity, therefore $T_2 + e$ is connected, removing an edge in a cycle will not break connectivity, therefore $T_2 + e - f$ is connected.
      iii. $T_2 - e + f$ have the same number of edges as $T_2$, which is $|T_2| - 1$, therefore $T_2 + e - f$ is a tree. (We have proven the connectivity in the previous step.)
      iv. $T_2 - e + f$ is spanning, connected, a tree, therefore it is a spanning tree.

□

**Theorem 9.2.** *Every connected graph has a spanning tree.*

*Proof.* Prove by constructing algorithm: □

**Algorithm 4** Find a spanning tree for connected graph (Prim's Algorithm in unweighted graph)

```
Require: a connected graph G and an enumeration $e_1, ...e_m$ of the edges of G
Ensure: a spanning tree T of G
1: Let T be the spanning subgraph of $G$ with $V(T) = V(G)$ and $E(T) = \emptyset$
2: $i \leftarrow 1$
3: while $i \leq |E|$ do
4:    if $T + e_i$ is acyclic then
5:       $T \leftarrow T + e_i$
6:       $i \leftarrow i + 1$
7:    end if
8: end while
```

**Notice:** This algorithm can be improved, one idea is to make summation of edges in spanning subgraph less or equation to $|V|-1$

For the complexity of spanning tree algorithm:

1. Space complexity, $2|E|$, which is $O(|E|)$
2. Time complexity
   (a) How to check for acyclic?
      i. At every stage $T$ has certain components $V_1, ...V_t$, (every time we add an edge, the number of components minus 1)
      ii. So at the beginning $t = |V|$ with $|V_i| = 1 \forall i$ and at the end, $t = 1$.
   (b) Count the amount of work for the algorithm.
      i. Need to check for acyclic for each edge, which costs $O(|E|)$
      ii. Need to flip the pointer for each vertex, for each vertex, at most will be flipped $\log|V|$ times, altogether $|V|\log|V|$ times.
      iii. The time complexity is $O(|E| + |V|\log|V|)$
3. First we need to input the data, create an array such that the first and the second entries are the ends of $e_1$, third and fourth are the ends of $e_2$, and so on.
4. The amount of storage needs in $2|E|$, which is $O(|E|)$
5. The main work involved in the algorithm is for each edges $e_i$ and the current $T$, to determine if $T + e_i$ creates a cycle.
6. suppose we keep each component $V_i$ by keeping for each vertex a pointer from the vertex to the name of the component containing it. Thus if $\mu \in V_3$, there will be a pointer from $\mu$ to integer 3.
7. Then when edge $e_i = \mu v$ is encountered in Step 2, we see that $T + e_i$ contains a cycle if and only if $\mu$ and $v$ point to same integer which means they are in the same component
8. If they are not in the same component, we want to add the edge which means then I have to update the pointers.

To prove algorithm we need to show the output is a spanning tree, which means three properties must hold:

- spanning (Step I)
- acyclic (We never add an edge that create a cycle)
- connected (Proof by contradiction)

So it is sufficient to show that the output will be connected.

*Proof.* (Proof by Contradiction) Suppose the output graph $T$ of the algorithm is NOT connected. Let $T_1$ be a component of $T$, let $x \in T_1$ and $y \notin T_1$. But $G$ is a connected graph (given from the beginning), so there must be a path in $G$ that connects $x$ and $y$. Let such a path in $G$ be $p = xe_1v_1e_2, ..v_{k-1}e_ky$. Clearly, $p \notin T_1$. So there must be a first vertex in $P$ that not in $T_1$. So $e_i \notin E(T)$, the only way this can happen when applying the algorithm is if $T + e_i$ creates a cycle $C$, i.e., $e_i \in C$, so $C - e_i$ is a path connecting $v_{i-1}$ and $v_i$. So $c - e_i \in T$, so $v_{i-1}$ is connected to $v_i \in T$. Contradiction. □

## 9.5 Cayley's Formula

## 9.6 Connectivity

## 9.7 Blocks

# Chapter 10

# Euler Tours and Hamilton Cycles

## 10.1 Euler Tours

## 10.2 Hamilton Cycles

# Chapter 11

# Matriod, Planarity

## 11.1 Plane and Planar Graphs

## 11.2 Dual Graphs

## 11.3 Matroids

**Definition 11.3.1** (Matroids)**.** Let $S$ be a finite set of **elements** and let $d$ be a collection of subsets of $S$ satisfying the property

$$\text{If } x \leq y, y \in d, \Rightarrow x \in d \tag{11.1}$$

The pair $(S, d)$ is called an **independent system** and the members of $d$ are called **independent sets**.

**Example.** Let $G$ be a graph and let $S \in E(G)$ define $M \subseteq S$ to be independent if $M$ is a matching

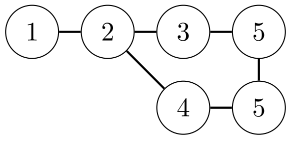
$$S = \{(1,2),(2,3),(2,4),(3,5),(4,6),(5,6)\} \tag{11.2}$$
$$d = \{\emptyset, \{(1,2)\}, \{(2,3)\}, ..., \{\text{other matching...}\}\} \tag{11.3}$$

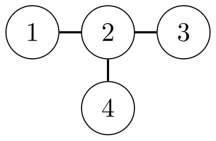
**Example.** Let $G$ be a graph and let $S = V(G)$ define $X \subseteq S$ to be independent if no two member of $x$ are adjacent.

$$S = \{1,2,3,4\} \tag{11.4}$$
$$d = \{\emptyset, 1, 2, 3, 4, (1,3), (1,4), (3,4), (1,3,4)\} \tag{11.5}$$

**Example.** Let $G$ be a connected graph and let $S = E(G)$, define $X \subseteq S$ to be independent if $G[X]$ contains cycles.

Given any independent system, there is a natural combinatorial optimization problem of finding the maximum cardinality independent set.
Let's try the following: **Greedy algorithm**
Step 1: Set $I = \emptyset$
Step 2: If there exists $e \in S \setminus I$ such that $I + e$ is independent, set $I \leftarrow I + e$ and go to Step 1, otherwise stop.
Those Independent systems for which the greedy algorithm guarantee to find a maximum cardinality independent set are very special called **matroids**





## 11.4 Independent Sets

## 11.5 Ramsey's Theorem

## 11.6 Turán's Theorem

## 11.7 Schur's Theorem

## 11.8 Euler's Formula

## 11.9 Bridges

## 11.10 Kuratowski's Theorem

## 11.11 Four-Color Theorem

## 11.12 Graphs on other surfaces

# Chapter 12

# Minimum Spanning Tree Problem

## 12.1 Basic Concepts

**Example.** A company wants to build a communication network for their offices. For a link between office $v$ and office $w$, there is a cost $c_{vw}$. If an office is connected to another office, then they are connected to with all its neighbors. Company wants to minimize the cost of communication networks.

**Definition 12.1.1** (Cut vertex)**.** A vertex $v$ of a connected graph $G$ is a **cut vertex** if $G \setminus v$ is not connected.

**Definition 12.1.2** (Connection problem)**.** Given a connected graph $G$ and a positive cost $C_e$ for each $e \in E$, find a minimum-cost spanning connnected subgraph of $G$. (Cycles all allowed)

**Lemma 12.1.** *An edge $e = uv \in G$ is an edge of a cycle of $G$ iff there is a path $G \setminus e$ from $u$ to $v$.*

**Definition 12.1.3** (Minumum spanning tree problem)**.** Given a connected graph graph $G$, and a cost $C_e, \forall e \in E$, find a minimum cost spanning tree of $G$

The only way a connection problem will be different than MSP is if we relax the restriction on $C_e > 0$ in the connection problem.

## 12.2 Kroskal's Algorithm

**Algorithm 5** Kroskal's Algorithm, $O(m \log m)$

```
Require: A connected graph
Ensure: A MST
  Keep a spanning forest $H = (V, F)$ of $G$, with $F = \emptyset$
  while $|F| < |V| - 1$ do
    add to $F$ a least-cost edge $e \notin F$ such that $H$ remains a forest.
  end while
```

## 12.3 Prim's Algorithm

**Algorithm 6** Prim's Algorithm, $O(nm)$

```
Require: A connected graph
Ensure: A MST
  Keep $H = (V(H), T)$ with $V(H) = \{v\}$, where $r \in V(G)$ and $T = \emptyset$
  while $|V(T)| < |V|$ do
    Add to $T$ a least-cost edge $e \notin T$ such that $H$ remains a tree.
  end while
```

## 12.4 Comparison between Kroskal's and Prim's Algorithm

- Kroskal start with a forest that contains all vertices, Prim start with a tree that only contain one vertex.
- Kroskal cannot gurantee every step it is a tree but can gurantee it is spanning, Prim can gurantee every step it is a tree but cannot gurantee spanning.

## 12.5 Extensible MST

**Definition 12.5.1** (cut)**.** For a graph $G = (V, E)$ and $A \subseteq V$ we denote $\delta(A) = \{e \in E : e \text{ has an end in } A \text{ and an end in } V \setminus A\}$. A set of the form $\delta(A)$ for some $A$ is called a **cut** of $G$.

**Definition 12.5.2.** We also define $\gamma(A) = \{e \in E : \text{both ends of } e \text{ are in } A\}$

**Theorem 12.2.** *A graph $G = (V, E)$ is connected iff there is no $A \subseteq V$ such that $\emptyset \neq A \neq V$ with $\delta(A) = \emptyset$*

**Definition 12.5.3.** Let us call a subset $A \in E$ **extensible** to a minimum spanning tree problem if $A$ is contained in the edge set of some MST of $G$

**Theorem 12.3.** *Suppose $B \subseteq E$, that $B$ is extensible to an MST and that $e$ is a minimum cost edge of some cut $D$ satisfying $D \cap B = \emptyset$, then $B \cup \{e\}$ is extensible to an MST.*

## 12.6 Solve MST in LP

Given a connected graph $G = (V, E)$ and a cost on the edges $C_e$ for all $e \in E$, Then we can formulate the following LP

$$X_e = \begin{cases} 1, \text{if edge } e \text{ is in the optimal solution} \\ 0, \text{otherwise} \end{cases} \tag{12.1}$$

The formulation is as following

$$\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e && (12.2) \\
\text{s.t.} \quad & \sum_{e \in E} x_e = |V| - 1 && (12.3) \\
& x_e \geq 0 && (12.4) \\
& e \in E && (12.5) \\
& \sum_{e \in E(S)} x_e = |S| - 1, \forall S \subseteq V, S \neq \emptyset && (12.6) \\
& && (12.7)
\end{aligned}$$

# Chapter 13

# Shortest-Path Problem

## 13.1 Basic Concepts

All Shortest-Path methods are based on the same concept, suppose we know there exists a dipath from $r$ to $v$ of a cost $y_v$. For each vertex $v \in V$ and we find an arc $(v, w) \in E$ satisfying $y_v + v_{vw} < y_w$. Since appending $(v, w)$ to the dipath to $v$ takes a cheaper dipath to $w$ then we can update $y_w$ to a lower cost dipath.

**Definition 13.1.1** (feasible potential)**.** We call $y = (y_v : v \in V)$ a **feasible potential** if it satisfies

$$y_v + c_{vw} \geq y_w \quad \forall (v, w) \in E \tag{13.1}$$

and $y_r = 0$

**Proposition 1.** *Feasible potential provides lower bound for the shortest path cost.*

*Proof.* Suppose that you have a dipath $P = v_0 e_1 v_1, ..., e_k v_k$ where $v_0 = r$ and $v_k = v$, then

$$C(P) = \sum_{i=1}^{k} C_{e_i} \geq \sum_{i=1}^{k} (y_{v_i} - y_{v_{i-1}}) = y_{v_k} - y_{v_0} \tag{13.2}$$

□

## 13.2 Breadth-First Search Algorithm

## 13.3 Ford's Method

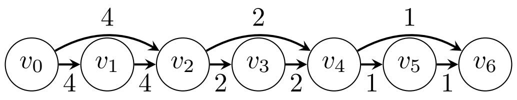
Define a predecessor function $P(w), \forall w \in V$ and set $P(w)$ to $v$ whenever $y_w$ is set to $y_v + c_{vw}$

**Algorithm 7** Ford's Method

```
Ensure: Shortest Paths from $r$ to all other nodes in $V$
Require: A digraph with arc costs,starting node $r$
  Initialize, $y_r = 0$ and $y_v = \infty, v \in V \setminus r$
  Initialize, $P(r) = 0, P(v) = -1, \forall v \in V \setminus r$
  while y is not a feasible potential do
    Let $e = (v, w) \in E$ (this could be problematic)
    if $y_v + c_{vw} < y_w$ (incorrect) then
      $y_w \leftarrow y_v + c_{vw}$ (correct it)
      $P(w) = v$ (set $v$ as predecessor)
    end if
  end while
```

**Notice:** Technically speaking, this is not an algorithm, for the following reasons: 1) We did not specify how to pick $e$, 2) This procedure might not stop given some situations, e.g., if there is a cycle with minus total weight

**Notice:** This method can be really bad. Here is another example that could take $O(2^n)$ to solve.



## 13.4 Ford-Bellman Algorithm

**Algorithm 8** Ford-Bellman Algorithm

```
Ensure: Shortest Paths from $r$ to all other nodes in $V$
Require: A digraph with arc costs,starting node $r$
  Initialize $y$ and $p$
  for $i = 0; i < N; i + +$ do
    for $\forall e = (v, w) \in E$ do
      if $y_v + c_{vw} < y_w$ (incorrect) then
        $y_w \leftarrow y_v + c_{vw}$ (correct it)
        $P(w) = v$ (set $v$ as predecessor)
      end if
    end for
  end for
  for $\forall e = (v, w) \in E$ do
    if $y_v + c_{vw} < y_w$ (incorrect) then
      Return error, negative cycle
    end if
  end for
```

**Notice:** Only correct the node that comes from a node that has been corrected.

A usual representation of a digraph is to store all the arcs having tail $v$ in a list $L_v$ to **scan** $v$ means the following:

- For $(v, w) \in L_v$, if $(v, w)$ is incorrect, then correct $(v, w)$

For Bellman, will either terminate with shortest path from $r$ to all $v \in V \setminus r$ or it will terminate stating that there is a negative cycle. In $O(mn)$
In the algorithm if $i = n$ and there exists a feasible potential, the problem has a negative cycle.
Suppose that the nodes of $G$ can be ordered from left to right so that all arcs go from left to right. That is suppose there is an ordering $v_1, v_2, ..., v_n \in V$ so that $(v_i, v_j) \in V$ implies $i < j$. We call such an ordering **topological** sort.
If we order $E$ in the sequence that $v_iv_j$ precedes $v_kv_i$ if $i < k$ based on topological order then ford algorithm will terminate in one pass.

## 13.5 SPFA Algorithm

## 13.6 Dijkstra Algorithm

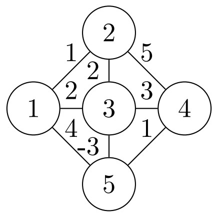
## 13.7 A* Algorithm

## 13.8 Floyd-Warshall Algorithm

If all weights/distances in the graph are nonnegative then we could use Dijkstra within starting nodes being any one of the vertices of the graph. This method will take $O(n^3)$

**Algorithm 9** Dijkstra Algorithm

```
Ensure: Shortest Paths from $r$ to all other nodes in $V$
Require: A digraph with arc costs,starting node $r$
  Initialize $y$ and $p$
  $S \leftarrow V$
  while $S \neq \emptyset$ do
    Choose $v \in S$ with minimum $y_v$
    $S \leftarrow S \setminus v$
    for $\forall w, (v, w) \in E$ do
      if $y_v + c_{vw} < y_w$ (incorrect) then
        $y_w \leftarrow y_v + c_{vw}$ (correct it)
        $P(w) = v$ (set $v$ as predecessor)
      end if
    end for
  end while
```



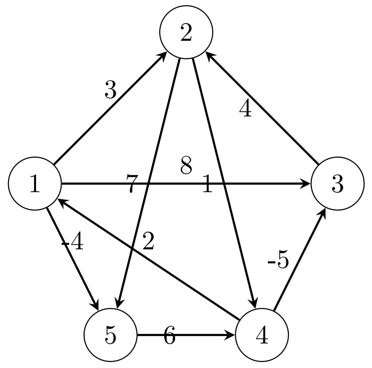
If weight/distances are arbitrary and we would like to find shortest path between all pairs of vertices or detect a negative cycle we could use Bellman-Ford Algorithm with $O(n^4)$
We would like an algorithm to find shortest path between any two pairs in a graph for arbitrary weights (determined, negative, cycles) in $O(n^3)$
Let $d_{ij}^k$ denote the length of the shortest path from $i$ to $j$ such that all intermediate vertices are contained in the set $\{1, ..., k\}$
In this case $d_{14}^5 = 5$
If the vertex $k$ is not an intermediate vertex on $p$, then $d_{ij}^k = d_{ij}^{k-1}$, notice that $d_{15}^4 = -1$, node 4 is not intermediate, so $d_{15}^3 = -1$
If the vertex $k$ is an intermediate on $p$, then $d_{ij}^k = d_{ik}^{k-1} + d_{kj}^{k-1}$, $d_{14}^5 = 0$ ($p = 1 \to 3 \to 5 \to 4$), i.e., $d_{14}^5 = d_{15}^4 + d_{54}^4 = 0$
Therefore $d_{ij}^k = \min\{d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}\}$
Input: graph $G = (V, E)$ with weight on edges Output: Shortest path between all pairs of vertices on existence of a negative cycle Step 1: Initialize

$$d_{ij}^0 = \begin{cases} c_{ij} & \text{distance from } i \text{ to } j \text{ if } (i,j) \in E \\ 0 & \text{if } i = j \\ \infty & \text{if } (i,j) \notin E \end{cases} \tag{13.3}$$

Step: For k = 1 to n For i = 1 to n For j = 1 to n $d_{ij}^k = \min\{d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}\}$ Next j Next i Next k Between optimal matrix $D^n$

$$D^0 = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \tag{13.4}$$

$$\Pi^0 = \begin{bmatrix} & 1 & 1 & & 1 \\ & & & 2 & 2 \\ & 3 & & & \\ 4 & & 4 & & \\ & & & 5 & \end{bmatrix} \tag{13.5}$$



$$D^1 = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \mathbf{5} & -5 & 0 & -\mathbf{2} \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \tag{13.6}$$

$$\Pi^1 = \begin{bmatrix} & 1 & 1 & & 1 \\ & & & 2 & 2 \\ & 3 & & & \\ 4 & \mathbf{1} & 4 & & \mathbf{1} \\ & & & 5 & \end{bmatrix} \tag{13.7}$$

$$D^2 = \begin{bmatrix} 0 & 3 & 8 & \mathbf{4} & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \mathbf{5} & \mathbf{11} \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \tag{13.8}$$

$$\Pi^2 = \begin{bmatrix} & 1 & 1 & \mathbf{2} & 1 \\ & & & 2 & 2 \\ & 3 & & \mathbf{2} & \mathbf{2} \\ 4 & 1 & 4 & & 1 \\ & & & 5 & \end{bmatrix} \tag{13.9}$$

$$D^3 = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -\mathbf{1} & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \tag{13.10}$$

$$\Pi^3 = \begin{bmatrix} & 1 & 1 & 2 & 1 \\ & & & 2 & 2 \\ & 3 & & 2 & 2 \\ 4 & \mathbf{3} & 4 & & 1 \\ & & & 5 & \end{bmatrix} \tag{13.11}$$

$$D^4 = \begin{bmatrix} 0 & 3 & -\mathbf{1} & 4 & -4 \\ \mathbf{3} & 0 & -\mathbf{4} & 1 & -\mathbf{1} \\ \mathbf{7} & 4 & 0 & 5 & \mathbf{3} \\ 2 & -1 & -5 & 0 & -2 \\ \mathbf{8} & \mathbf{5} & \mathbf{1} & 6 & 0 \end{bmatrix} \tag{13.12}$$

$$\Pi^4 = \begin{bmatrix} & 1 & \mathbf{4} & 2 & 1 \\ \mathbf{4} & & \mathbf{4} & 2 & \mathbf{1} \\ \mathbf{4} & 3 & & 2 & \mathbf{1} \\ 4 & 3 & 4 & & 1 \\ \mathbf{4} & \mathbf{3} & \mathbf{4} & 5 & \end{bmatrix} \tag{13.13}$$

$$D^5 = \begin{bmatrix} 0 & \mathbf{1} & -\mathbf{3} & \mathbf{2} & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix} \tag{13.14}$$

$$\Pi^5 = \begin{bmatrix} & \mathbf{3} & 4 & \mathbf{5} & 1 \\ 4 & & 4 & 2 & 1 \\ 4 & 3 & & 2 & 1 \\ 4 & 3 & 4 & & 1 \\ 4 & 3 & 4 & 5 & \end{bmatrix} \tag{13.15}$$

Time complexity $O(n^3)$
If during the previous processes, there exist an element of negative value in the diagonal, it means there exists negative cycle.

## 13.9 Johnson's Algorithm

# Chapter 14

# Maximum Flow Problem

## 14.1 Basic Concept

Let $D = (V, A)$ be a strict diagraph with distinguished vertices $s$ and $t$. We call $s$ the source and $t$ the sink, let $u = \{u_e : e \in A\}$ be a nonnegative integer-valued capacity function defined on the arcs of $D$. The maximum flow problem on $(D, s, t, u)$ is the following Linear program.

$$\max \quad v \tag{14.1}$$

$$\text{s.t.} \quad \sum_{h(e)=i} x_e - \sum_{t(e)=i} x_e = \begin{cases} -v, & \text{if } i = s \\ v, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases} \tag{14.2}$$

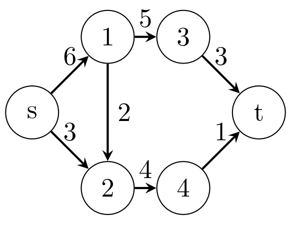
$$0 \le x_e \le u_e, \quad \forall e \in A \tag{14.3}$$

We think of $x_e$ as being the flow on arc $e$. Constraint says that for $i \neq s, t$ the flow into a vertex has to be equal to the flow out of vertex. That is, flow is conceded at vertex $i$ for $i = s$ and for $i = t$ the net flow in the entire digraph must be equal to $v$. A $\mathbf{x_e}$ that satisfied the above constraints is an $(s,t)$-flow of value $v$. If in addition it satisfies the bounding constraints, then it is a feasible $(s,t)$-flow. A feasible $(s,t)$-flow that has maximum $v$ is optimal on maximum.

## 14.2 Solving Maximum Flow Problem in LP

**Theorem 14.1.** *For $S \subseteq V$ we define $(S, \bar{S})$ to be a $(s,t)$-cut if $s \in S$ and $t \in \bar{S} = V - S$, the capacity of the cut, denoted $u(S, \bar{S})$ as $\sum\{u_e : e \in \delta^-(S)\}$ where $\delta^-(S) = \{e \in A : t(e) \in S$ and $h(e) \in \bar{S}\}$*

**Example.** For the following graph:
Let $S = \{1, 2, 3, s\}$, $\bar{S} = \{4, t\}$

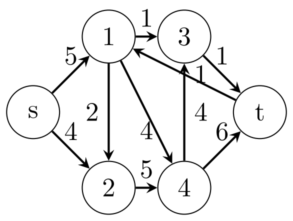


then $\delta^-(S) = \{(2,4), (3,t)\} \Rightarrow u(S, \bar{S}) = 7$

**Definition 14.2.1.** If $(S, \bar{S})$ has minimum capacity of all $(s,t)$-cuts, then it is called **minimum cut**.

**Definition 14.2.2.** Let $\delta^+(S) = \delta^-(V - S)$

**Example.** Let $S = \{s, 1, 2, 3\}$, $\bar{S} = \{4, t\}$, $u(S, \bar{S}) = u_{14} + u_{24} + u_{3t} = 10$, $\delta^-(S) = \{(1,4), (2,4), (3,t)\}$, $\delta^+ S = \{(4,3), (t,1)\}$



**Lemma 14.2.** *If $x$ is a $(s,t)$ flow of value $v$ and $(S,\bar{S})$ is a $(s,t)$-cut, then*

$$v = \sum_{e\in\delta^-(S)} x_e - \sum_{e\in\delta^+(S)} x_e \tag{14.4}$$

*Proof.* Summing the first set of constraints over the vertices of $S$,

$$\sum_{i\in S}(\sum_{h(e)=i} x_e - \sum_{t(e)=i} x_e) = -v \tag{14.5}$$

Now for an arc $e$ with both ends in $S$, $x_e$ will occur twice once with a positive and once with negative so they cancel and the above sum is reduced to

$$\sum_{e\in\delta^+(S)} x_e - \sum_{e\in\delta^-(S)} x_e = -v \tag{14.6}$$

□

**Notice:** Flow is the prime variable, capacity is the dual variable.

**Corollary 14.2.1.** *If $x$ is a feasible flow of value $v$, and $(S,\bar{S})$ is an $(s,t)$-cut, then*

$$v \le u(S,\bar{S}) \quad \textit{(Weak duality)} \tag{14.7}$$

**Definition 14.2.3.** Define an arc $e$ to be **saturated** if $x_e = u_e$, and to be **flowless** if $x_e = 0$

**Corollary 14.2.2.** *Let $x$ be a feasible flow and $(S,\bar{S})$ be a $(s,t)$-cut, if $\forall e \in \delta^-(S)$ is saturated, and $\forall e \in \delta^+(S)$ is flowless, then $x$ is a maximum flow and $(S,\bar{S})$ is a minimum cut. (Strong duality)*

*Proof.* If every arc of $\delta^-(S)$ is saturated then

$$\sum_{e\in\delta^-(S)} x_e = \sum_{e\in\delta^-(S)} u_e \tag{14.8}$$

If every arc of $\delta^+(S)$ is flowless then

$$\sum_{e\in\delta^+(S)} x_e = 0 \tag{14.9}$$

$\Rightarrow$ $x$ is as large as it can get when as $u(S,\bar{S})$ is as small as it can get. □

## 14.3 Prime and Dual of Maximum Network Flow Problem

The LP of maximum flow can be modeled as following, WLOG, we let $s = v_1 \in V, t = v_{|V|} \in V$.

$$\max \quad f = \begin{bmatrix}0 & 0 & \cdots & 0 & 1\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ f\end{bmatrix} \tag{14.10}$$

$$\text{s.t.} \quad \begin{bmatrix}\mathbf{A} & \mathbf{F}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ f\end{bmatrix} = \mathbf{0} \tag{14.11}$$

$$\mathbf{I}\mathbf{x} \le \mathbf{u} \tag{14.12}$$

$$\begin{bmatrix}\mathbf{x}\\ f\end{bmatrix} \ge 0 \tag{14.13}$$

In which $\mathbf{A}$ is the vertex-arc incident matrix and $\mathbf{F}$ is a column vector where the first row is -1, last row is 1 and all other rows are 0s, which is because we denote the first vertex as source $s$ and the last vertex as the sink $t$. $\mathbf{u}$ is the column vector of upper bound of each arcs.

$$\mathbf{A} = \mathbf{A}_{|E|\times|V|} = [a_{ij}], \text{ where } a_{ij} = \begin{cases} 1, & \text{if } v_i = h(e_j) \\ -1, & \text{if } v_i = t(e_j) \\ 0, & \text{otherwise} \end{cases} \tag{14.14}$$

$$\mathbf{F} = \begin{bmatrix} -1 & \cdots & 0 & \cdots & 1 \end{bmatrix}^\top \tag{14.15}$$

$$\mathbf{u} = \begin{bmatrix} u_1 & u_2 & \cdots & u_{|E|} \end{bmatrix}^\top \tag{14.16}$$

Then, we take the dual of LP

$$\min \quad \mathbf{u}\mathbf{w_E} \tag{14.17}$$

$$\text{s.t.} \quad \begin{bmatrix} \mathbf{w_V} & \mathbf{w_E} \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ \mathbf{I} \end{bmatrix} \geq 0 \tag{14.18}$$

$$\begin{bmatrix} \mathbf{w_V} & \mathbf{w_E} \end{bmatrix} \begin{bmatrix} \mathbf{F} \\ \mathbf{0} \end{bmatrix} = 1 \tag{14.19}$$

$$\mathbf{w_V} \quad \text{unrestricted} \tag{14.20}$$

$$\mathbf{w_E} \geq \mathbf{0} \tag{14.21}$$

In which $\mathbf{w_V}$ is "whether or not" vertex $v$ is in $S$ where $(S, \bar{S})$ represents a cut, $\mathbf{w_E}$ is "whether or not" an arc in in $\delta^+(S)$. $\mathbf{u}, \mathbf{E}, \mathbf{F}$ have the same meaning as in prime.

$$\mathbf{w_V} = \begin{bmatrix} w_1 & w_2 & \cdots & w_{|V|} \end{bmatrix}^\top \tag{14.22}$$

$$\mathbf{w_E} = \begin{bmatrix} w_{|V|+1} & w_{|V|+2} & \cdots & w_{|V|+|E|} \end{bmatrix}^\top \tag{14.23}$$

To make it more clear, it can be rewritten as following

$$\min \quad \sum_{e \in E} u_e w_e \tag{14.24}$$

$$\text{s.t.} \quad w_i - w_j + w_{|V|+e} \geq 0, \forall e = (i,j) \in E \tag{14.25}$$

$$-w_1 + w_{|V|} = 1 \tag{14.26}$$

$$\mathbf{w_V} \quad \text{unrestricted} \tag{14.27}$$

$$\mathbf{w_E} \geq \mathbf{0} \tag{14.28}$$

The meaning for the first set of constraint is to decide whether or not an arc is in $\delta^+(S)$ of a $(S, \bar{S})$, which is decided by $w_V$. The $w_1 - w_{|V|} = 1$, which is the second set of constraint means the source $s = v_1$ and the sink $t = v_{|V|}$ has to be in $S$ and $\bar{S}$ respectively.

## 14.4 Maximum Flow Minimum Cut Theorem

**Definition 14.4.1.** Let $P$ be a path, (not necessarily a dipath), $P$ is called **unsaturated** if every **forward** arc is unsaturated ($x_e < u_e$) and ever **reverse** arc has positive flow ($x_e > 0$). If in addition $P$ is an $(s,t)$-path, then $P$ is called an **x-augmenting path**

**Theorem 14.3.** *A feasible flow $x$ in a digraph $D$ is maximum iff $D$ has no augmenting paths.*

*Proof.* (Prove by contradiction)
($\Rightarrow$) Let $x$ be a maximum flow of value $v$ and suppose $D$ has an augmenting path. Define in $P$ (augmenting path):

$$D_1 = \min\{u_e - x_e : e \text{ forward in } P\} \tag{14.29}$$

$$D_2 = \min\{x_e : e \text{ backward in } P\} \tag{14.30}$$

$$D = \min\{D_1, D_2\} \tag{14.31}$$

Since $P$ is augmenting, then $D > 0$, let

$$\hat{x_e} = \begin{cases} x_e + D & \text{If } e \text{ is forward in } P \\ x_e - D & \text{If } e \text{ is backward in } P \\ x_e & otherwise \end{cases} \tag{14.32}$$

It is easy to see that $\hat{x}$ is feasible flow and that the value is $V + D$, a contradiction.
($\Leftarrow$) Suppose $D$ admits no x-augmenting path, Let $S$ be the set of vertices reachable from $s$ by x-unsaturated path clearly $s \in S$ and $t \notin S$ (because otherwise there would be an augmenting path). Thus, $(S, \bar{S})$ is a $(s,t)$-cut.
Let $e \in \delta^-(S)$ then $e$ must be saturated. For otherwise we could add the $h(e)$ to $S$
Let $e \in \delta^+(S)$ then $e$ must be flow less. For otherwise we could add the $t(e)$ to $S$.
According to previous corollary, that $x$ is maximum. □

**Theorem 14.4.** *(Max-flow = Minimum-cut) For any digraph, the value of a maximum $(s,t)$-flow is equal to the capacity of a minimum $(s,t)$-cut*

## 14.5 Ford-Fulkerson Method

Finding augmenting paths is the key of max-flow algorithm, we need to describe two functions, labeling and scanning a vertex.
A vertex is first labeled if we can find x-unsaturated path from $s$, i.e., $(s,v)$-unsaturated path.
The vertex $v$ is scanned after we attempted to extend the x-unsaturated path.
This algorithm is incomplete/incorrect, needs to be fixed

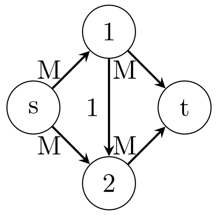
**Algorithm 10** Labeling algorithm

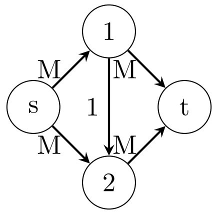
```
Ensure: Max-flow $x$ with value $v$
Require: Digraph with source $s$ and sink $t$, a capacity function $u$ and a feasible flow (could be $x_e = 0$)
  Initialize, $v \leftarrow x$
  Designate all vertices as unlabeled and unscanned
  Label $s$
  while There exists vertex unlabeled or unscanned do
    Let $i$ be such a vertex, for each arc $e$ with $t(e) = i, x_e < u_e$ and $h(e)$ unlabeled, label $h(e)$
    For each arc $e$ with $h(e) = i, x_e > 0$ and $t(e)$ unlabeled, label $t(e)$, designate $i$ as scanned.
    If $t$ is not label
  end while
  $x$ is the maximum.
```

**Algorithm 11** Ford-Fulkerson algorithm

```
Ensure: Max-flow $x$ with value $v$
Require: Digraph with source $s$ and sink $t$, a capacity function $u$ and a feasible flow (could be $x_e = 0$)
  Initialize, $v \leftarrow x$
  Designate all vertices as unlabeled and unscanned
  Label $s$
  while There exists vertex unlabeled or unscanned do
    Let $i$ be such a vertex, for each arc $e$ with $t(e) = i, x_e < u_e$ and $h(e)$ unlabeled, label $h(e)$
    For each arc $e$ with $h(e) = i, x_e > 0$ and $t(e)$ unlabeled, label $t(e)$, designate $i$ as scanned.
    If $t$ is not label
  end while
  $x$ is the maximum.
```

Labeling algorithm can be exponential, the following is an example





## 14.6 Polynomial Algorithm for max flow

Let $(D, s, t, u)$ be a max flow problem and let $x$ be a feasible flow for $D$, the **x-layers** of $D$ are define be the following algorithm
Layer algorithm (Dinic 1977) Input: A network $(D, s, t, u)$ and a feasible flow $x$ Output: The **x-layers** $V_0, V_1, ..., V_l$ where $V_i \cap V_j = \emptyset \forall i \neq j$
Step 1: Set $V_0 = \{s\}, i \leftarrow 0 and l(x) = 0$ Step 2: Let $R$ be the set of vertices $w$ such that there is an arc $e$ with either:

- $t(e) \in V_i, h(e) = w, x_e < u_e$ or
- $h(e) \in V_j, t(e) = w, x_e > 0$

Step 3: If $t \in R$, set $V_{i+1} = \{t\}$, $l(t) = i + 1$ and stop. Set $V_{i+1} \leftarrow R \setminus \cup_{0 \leq j \leq i} V_j$, $l \leftarrow i + 1, l(x) = i$, goto Step 2. If $V_{i+1} = \emptyset$, set $l(x) = i$ and Stop.

**Example.** For the following graph

$$\text{Second iteration} \quad (14.33)$$
$$V_0 = \{s\}, i = 0, l(x) = 0 \quad (14.34)$$
$$R = \{1, 2\} \quad (14.35)$$
$$V_1 \leftarrow \{1, 2\}, i = 1, l(x) = 1 \quad (14.36)$$
$$R = \{3, 4, 5\} \quad (14.37)$$
$$V_2 \leftarrow \{3, 4\}, i = 2, l(x) = 2 \quad (14.38)$$
$$R = \{1, 5, 6, 3\} \quad (14.39)$$
$$V_3 \leftarrow \{5, 6\}, i = 3, l(x) = 3 \quad (14.40)$$
$$R = \{4, t\} \quad (14.41)$$
$$V_4 = \{t\} \quad (14.42)$$
$$A_1 = \{(s, 1), (s, 2)\} \quad (14.43)$$
$$A_2 = \{(1, 3), (2, 4)\} \quad (14.44)$$
$$A_3 = \{(3, 5), (4, 6)\} \quad (14.45)$$
$$A_4 = \{(5, t), (6, t)\} \quad (14.46)$$

The layer network $D_x$ is defined by $V(D_x) = V_0 \cup V_1 \cup V_2 \cdots \cup V_{l(x)}$
Suppose we have computed the layers of $D$ and $t \in V_l(x)$, the last layer (last layer I am goin to $V_e$)
For each $i, 1 \leq i \leq l$, define a set of arcs $A_i$ and a function $\hat{u}$ on $A_i$ as following. For each $e \in A(D)$

- If $t(e) \in V_{i-1}, h(e) \in V_i$ and $x_e < u_e$ then add arc $e$ to $A_i$ and define $\hat{u}_e = u_e - x_e$
- If $h(e) \leftarrow V_{i-1}, t(e) \in V_i$ and $x_e > 0$ then add arc $e' = (h(e), t(e))$ to $A_i$ with $\hat{u}_e - x_e$

Let $\hat{u}$ be the capacity function on $D_x$ and let the source and sink of $D_x$ be $s$ and $t$
We can think of $D_x$ as being make of arc shortest (in terms of numbers of arcs) x-augmenting paths.
A feasible flow in a network is said to be maximal (does not means maximum) if every $(s,t)$-directed path contains at least one saturated arc.
For layered algorithm $V_0, V_1, ..., V_L$
Arcs:

- If $t(e) \in V_{i-1}$, $h(e) \in V_i$ and $x_e < u_e$, then add $e$ to $A_i$ with $\hat{u_e} = u_e - x_e$
- If $h(e) \in V_{i-1}$, $t(e) \in V_i$ and $x_e > 0$, then add arc $e' = (h(e), t(e))$ to $A_i$ and define $\hat{u_e} = x_e$

Maximal Flow: If every directed $(s,t)$-path has at least one saturated arc.
Computing maximal flow is easier than computing maximum flow, since we never need to consider canceling flows on reverse arcs,
Let $\hat{x}$ be a maximal flow on the layered network $D_x$, we can define new flows in $D(x')$ by

$$x'_e = x_e + \hat{x_e}, \quad \text{If } t(e) \in V_{i-1}, h(e) \in V_i \tag{14.47}$$
$$x'_e = x_e - \hat{x_e}, \quad \text{If } h(e) \in V_{i-1}, t(e) \in V_i \tag{14.48}$$

## 14.7 Dinic Algorithm

Input: A layered network $(D_x, s, t, \hat{u})$ and a feasible flow x Output: A maximal flow $\hat{x}$ from $D_x$
Step 1: Set $H \leftarrow D_x$ and $i \leftarrow S$ Step 2: If there is no arc $e$ with $t(e) = i$, goto Step 4, otherwise let $e$ be such an arc Step 3: Set $T(h(e)) \leftarrow i$ and $i \leftarrow h(e)$, if $i = t$ goto Step 5, otherwise goto Step 2. Step 4: If $i = s$, Stop, Otherwise delete $i$ and all incident arcs with $H$, set $i \leftarrow T(i)$ and goto Step 2 Step 5: Construct the directed path, $s = i_0 e_1 i_1 e_2 ... e_k i_k = t$ where $i_{j-1} = T(i_j), 1 \leq j \leq k$. Set $D = \min\{\hat{u_{e_j}} - \hat{x_{e_j}} : i \leq j \leq k\}$, set $\hat{x_{e_j}} \leftarrow \hat{x_{e_j}} + D, i \leq j \leq k$. Delete from $H$ all saturated arcs on this path, set $i \leftarrow 1$ and goto Step 2.

**Algorithm 12** Dinic Algorithm

```
Ensure: A maximal flow $\hat{x}$ from $D_x$
Require: A layered network $(D_x, s, t, \hat{u})$ and a feasible flow x
  Initialize $H \leftarrow D_x$ and $i \leftarrow S$
```

**Theorem 14.5.** *Dinic algorithm runs in $O(|E||V|^2)$*

*Proof.* Step 1 is $O(|E||V|)$ Step 2 runs Step 1 for $O(|V|)$ times □

# Chapter 15

# Minimum Weight Flow Problem

## 15.1 Transshipment Problem

Transshipment Problem $(D, b, w)$ is a linear program of the form

$$\begin{aligned} \min \quad & wx & (15.1) \\ \text{s.t.} \quad & Nx = b & (15.2) \\ & x \geq 0 & (15.3) \end{aligned}$$

Where $N$ is a vertex-arc incident matrix. For a feasible solution to LP to exist, the sum of all $b$s must be zero. Since the summation of rows of $N$ is zero. The interpretation of the LP is as follows.
The variables are defined on the edges of the digraph and that $x_e$ denote the amount of flow of some commodity from the tail of $e$ to the head of $e$
Each constraints

$$\sum_{h(e)=i} x_e - \sum_{t(e)=i} x_e = b_i \tag{15.4}$$

represents consequential of flow of all edges into $k$ vertex that have a demand of $b_i > 0$, or a supply of $b_i < 0$. If $b_i = 0$ we call that vertex a transshipment vertex.

## 15.2 Network Simplex Method

**Lemma 15.1.** *Let $C_1$ and $C_2$ be distinct cycles in a graph $G$ and let $e \in C_1 \cup C_2$. Then $(C_1 \cup C_2) \setminus e$ contains a cycle.*

*Proof.* Case 1: $C_1 \cap C_2 = \emptyset$. Trivia.
Case 2: $C_1 \cap C_2 \neq \emptyset$. Let $e \in C_2$ and $f = uv \in C_1 \setminus C_2$. Starting at $v$ traverse $C_1$ in the direction away from $u$ until the first vertex of $C_2$, say $x$. Denote the $(v, x)$-path as $P$. Starting at $u$ traverse $C_1$ in the direction away from $v$ until the first vertex of $C_2$, say $y$. Denote the $(u, y)$-path as $Q$. $C_2$ is a cycle, there are two $(x, y)$-path in $C_2$. Denote the $(x, y)$-path without $e$ as $R$. Then $vPxRyQ^{-1}uf$ is a cycle. □

**Theorem 15.2.** *Let $T$ be a spanning tree of $G$. And let $e \in E \setminus T$ then $T + e$ contains a unique cycle $C$ and for any edge $f \in C$, $T + e - f$ is a spanning tree of $G$*

Let $(D, b, w)$ be a transshipment problem. A feasible solutions $x$ is a **feasible tree solution** if there is a spanning tree $T$ such that $||x|| = \{e \in A, x_e \neq 0\} \subseteq T$.
The strategy of network simplex algorithm is to generate negative cycles, if negative cycle exists, it means the solution can be improved.
For any tree $T$ of $D$ and for $e \in A \setminus T$, it follows from above theorem that $T + e$ contains a unique cycle. Denote that cycle $C(T, e)$ and orient it in the direction of $e$, define

$$\begin{aligned} w(T, e) = & \sum \{w_e : e \text{ forward in } C(T, e)\} \\ & - \sum \{w_e : e \text{ reverse in } C(T, e)\} \end{aligned} \tag{15.5}$$

We think of $w(T, e)$ as the weight of $C(T, e)$.

### 15.2.1 Network Simplex Method

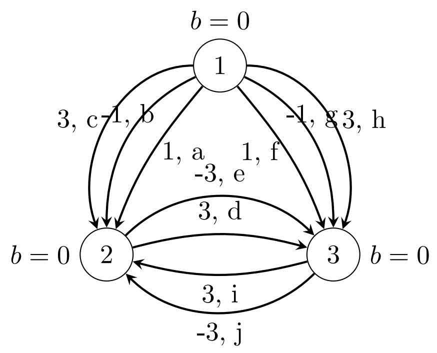
**Algorithm 13** Network Simplex Method Algorithm

```
Ensure: An optimal solution or the conclusion that $(D, b, w)$ is unbounded
Require: A transshipment problem $(D, b, w)$ and a feasible tree solution $x$ containing to a spanning tree $T$
  while $\exists e \in A \setminus T, w(T, e) < 0$ do
    let $e \in A \setminus T$ be such that $w(T, e) < 0$.
    if $C(T, e)$ has no reverse arcs then
      Return unboundedness
    else
      Set $\theta = \min\{x_f : f$ reverse in $C(T, e)\}$ and set $f = \{f \in C(T, e) : f$ reverse in $C(T, e), x_f = \theta\}$
      if $f$ forward in $C(T, e)$ then
        $x_f \leftarrow x_f + \theta$
      else
        $x_f \leftarrow x_f - \theta$
      end if
      Let $f \in F$ and $T \leftarrow T + e - f$
    end if
  end while
  Return $x$ as optimal
```

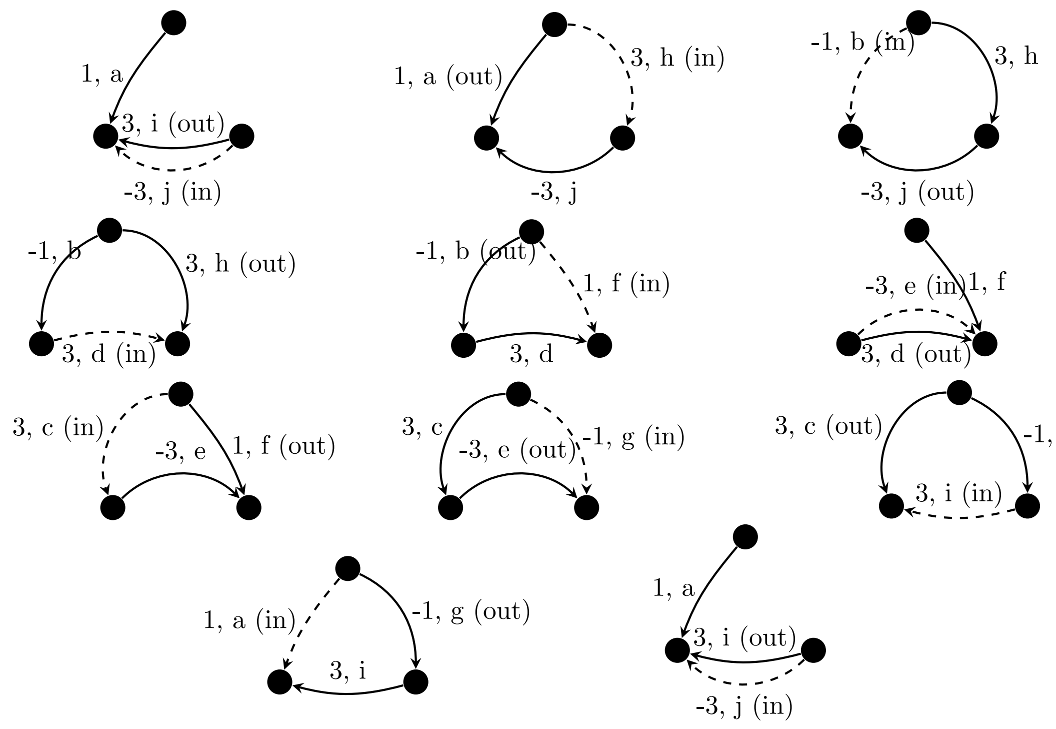
### 15.2.2 Example for cycling

**Notice:** Similar to Simplex Method in LP, even though in worst case may be inefficient. In most cases it is simple and empirically efficient. Also, similarily, there will be cycling problems.

The following is an example of cycling



Then for the following steps we can detect cycling:
$w(T, j) = w_j - w_i = -3 - 3 = -6$, therefore $j$ in entering basis, $i$ is leaving basis.
$w(T, h) = w_h + w_j - w_a = 3 - 3 - 1 = -1$, therefore $h$ is entering basis, $a$ is leaving basis.
$w(T, b) = w_b - w_j - w_h = -1 + 3 - 3 = -1$, therefore $b$ is entering basis, $j$ is leaving basis.
$w(T, d) = w_d - w_h + w_b = 3 - 3 - 1 = -1$, therefore $d$ is entering basis, $h$ is leaving basis.
$w(T, f) = w_f - w_d - w_b = 1 - 3 + 1 = -1$, therefore $f$ is entering basis, $b$ is leaving basis.
$w(T, e) = w_e - w_d = -3 - 3 = -6$, therefore $e$ is entering basis, $d$ is leaving basis.
$w(T, c) = w_c + w_e - w_f = 3 - 3 - 1 = -1$, therefore $c$ is entering basis, $f$ is leaving basis.
$w(T, g) = w_g - w_e - w_c = -1 + 3 - 3 = -1$, therefore $g$ is entering basis, $e$ is leaving basis.
$w(T, i) = w_i - w_c + w_g = 3 - 3 - 1 = -1$, therefore $i$ is entering basis, $c$ is leaving basis.



$w(T, a) = w_a - w_i - w_g = 1 - 3 + 1 = -1$, therefore $a$ is entering basis, $g$ is leaving basis.

The last graph is the same as the first graph, i.e., cycling detected.

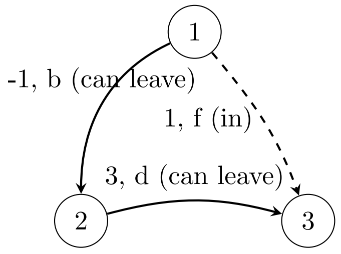
### 15.2.3 Cycling prevention

To Avoid cycling we will introduce the Modified Network Simplex Method. Let $T$ be a **rooted** spanning tree. Let $f$ be an arc in $T$, we say $f$ is **away** from the root $r$ if $t(f)$ is the component of $T - f$. Otherwise we say $f$ is **towards** $r$.
Let $x$ be a feasible tree solution associated with $T$, then we say $T$ is a **strong feasible tree** if for every arc $f \in T$ with $x_f = 0$ then $f$ is away from $r \in T$.
Modification to NSM:

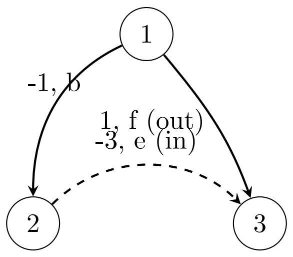
- The algorithm is initialed with a strong feasible tree.
- $f$ in pivot phase is chosen to be the first reverse arc of $C(T, e)$ having $x_f = \theta$. By "first", we mean the first arc encountered in traversing $C(T, e)$ in the direction of $e$, starting at the vertex $i$ of $C(T, e)$ that minimizes the number of arcs in the unique $(r, i)$-path in $T$.

**Notice:** In the second rule above, $r$ could also be in the cycle, in that case, $i$ is $r$.

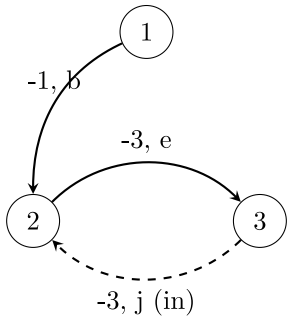
Continue the previous example. Now should how we can avoid cycling:
The first few (four) steps are the same as previous example, starting from
$w(T, f) = w_f - w_d - w_b = 1 - 3 + 1 = -1$. $f$ is entering basis, both $b$ and $d$ can leave the basis, according to



the modified pivot rule, we choose the "first" arc encountered in traversing $C(T, e)$, which is $d$ to leave the basis, instead of $b$.
$w(T, e) = w_e - w_f + w_b = -5$, $e$ is entering basis, $f$ is leaving basis. Now the only arc to enter basis and maintain

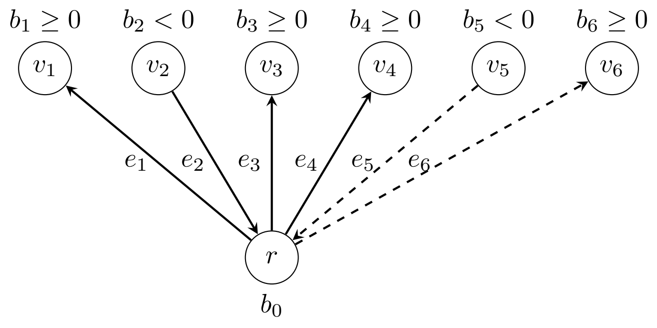


negative $w$ is $j$.
$w(T, j) = w_j + w_e = -6$, but in $C(T, j)$ there is no reversing arc, therefore we detect unboundedness.



## 15.2.4 Finding Initial Strong Feasible Tree

Pick a vertex in $D$ to be root $r$. The tree $T$ has an arc $e$ with the $t(e) = r$ and $h(e) = v$. For each $v \in V \setminus r$ with $b_v \geq 0$ and has an arc $e$ with $h(e) = r$ and $t(e) = v$ for each $v \in V \setminus r$ for which $b_v < 0$. Wherever possible the arcs of $T$ are chosen from $A$, where an appropriate arc doesn't exist. We create an **artificial arc** and give its weight $|V|(\max\{w_e : e \in A\}) + 1$. This is similar to Big-M method and if optimal solution contains artificial arcs ongoing arc problem is infeasible.
Here is an example after adding artificial arcs:
Where $e_5$ and $e_6$ are artificial arcs, the weight of those arcs are $|V|(\max\{w_e : e \in \mathcal{A}\}) + 1$. And the above tree is



a basic feasible solution.
We need to prove that such artificial arc has sufficiently large weight to guarantee

- It will leave the basis, and
- It will not enter the basis again (for this, just delete the artificial arc after it leaves the basis, then it will never enter the basis again)

*Proof.* Now prove that such arcs will always leave the basis. Before the prove we give some notation.

- Define set $E$ as the set of arcs which is not artificial arc, in the above example, $E = \{e_1, e_2, e_3, e_4\}$.
- Define set $A$ as the set of arcs which are artificial arcs, in the above example, $A = \{e_5, e_6\}$. Noticed that $E \cap A = \emptyset$.
- Define set $M$ as the vertices in the spanning tree that is reachable from $r$ by $E$, in the above example, $M = \{v_1, v_2, v_3, v_4\}$.
- Define $M' = (V \setminus r) \setminus M$ in the tree that can only be reached from $r$ by $A$, i.e., artificial arcs, in the above example, $M' = \{v_5, v_6\}$.

Then the initial basic feasible solution is a graph

$$G_0 =< M \cup M' \cup \{r\}, E \cup A > \tag{15.6}$$

Denote the origin graph

$$G =< V, \mathcal{A} > \tag{15.7}$$

Notice that with the artificial arcs, $G_0$ is not a subgraph of $G$.
Let $(M \cup \{r\}, M')$ be a cut in the origin graph $G$. For the vertices in $M'$, one of the following cases will happen:

- case 1: $\sum_{v \in M'} b_v \geq 0$
- case 2: $\sum_{v \in M'} b_v < 0$

For case 1, we claim that at least one of the vertices $v_{M'} \in M'$ with $b_{v_{M'}} \geq 0$ linked by an arc, say $f$, such that $h(f) = v_{M'}$ and $t(f) = v_M \in M$. Otherwise the balance of flow cannot hold in the origin graph $G$. Furthermore, denote the artificial arc from $r$ to $v_{M'}$ by $e_{rv_{M'}}$.
Notice that for $v_M$ there is not necessarily be an arc between $r$ and $v_M$, but there must exists an $(r, v_M)$-path denoted by $P$, for $M$ is the set of vertices that reachable from $r$ by arcs in $E$.
Take that arc $f$ as entering arc to the basis. Then

$$C(T, f) = re_{rv_{M'}} v_{M'} f v_M P r \tag{15.8}$$

For

$$w(T, f) = w_f - w_{e_{rv_{M'}}} + \sum_{e \in P} d_e w_e \tag{15.9}$$

where $d_e = 1$ if $w_e$ is forward in $P$ and $d_e = -1$ otherwise.
Now that $w_{e_{rv_{M'}}} = |V|(\max\{w_e : e \in \mathcal{A}\}) + 1$, it guarantees that

$$\begin{align}
w(T, f) &= w_f + \sum_{e \in P} d_e w_e - w_{e_{rv_{M'}}} \tag{15.10}\\
&\leq w_f + \sum_{e \in P} w_e - w_{e_{rv_{M'}}} \tag{15.11}\\
&\leq \sum_{e \in \mathcal{A}} w_e - w_{e_{rv_{M'}}} \tag{15.12}\\
&\leq |V|(\max\{w_e : e \in \mathcal{A}\}) - w_{e_{rv_{M'}}} \tag{15.13}\\
&\leq -1 < 0 \tag{15.14}
\end{align}$$

So $f$ can enter the basis, and the artificial variable $e_{rv_{M'}}$ will leave the basis, for it is the most violated reverse arc in the $C(T, f)$. When we put $f$ into the basis, update $G_0$, such that $M \leftarrow M \cup \{v_{M'}\}$ and $M' \leftarrow M' \setminus \{v_{M'}\}$.
For case 2, it is similar. At least one of the vertices $v_{M'} \in M'$ with $b_{v_{M'}} < 0$ linked by an arc, say $f\prime$, such that $t(f') = v_{M'}$ and $h(f') = v_M \in M$. Otherwise the balance of flow cannot hold in the origin graph $G$. Furthermore, denote the artificial arc from $v_{M'}$ to $r$ by $e_{v_{M'}r}$.
Similarly we can find a cycle $C(T, f') = rP'v_M f' v_{M'} e_{v_{M'}r} r$. $w(T, f') = w_{f'} - w_{e_{rv_{M'}}} + \sum_{e \in P'} d_e w_e$, where $d_e = 1$ if $w_e$ is forward in $P\prime$ and $d_e = -1$. We can prove $w(T, f') \leq -1 < 0$. That that $f'$ as entering arc to the basis, similarly move $v_{M'}$ form set $M'$ to $M$.
The above case can be dealt with iteratively until set $M'$ become $\emptyset$, at which stage there is no artificial arc in the basic feasible solution. Which means all the artificial variable can leave the basis. □

**Notice:** This algorithm can be really bad, its mimic of Simplex Method of LP, which means we can run into exponential operations

## 15.3 Transshipment Problem and Circulation Problem

**Definition 15.3.1.** The minimum weight circulation problem is defined as follows:

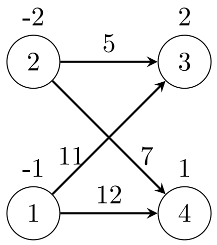
$$\begin{aligned} \min \quad & wx && (15.15)\\ \text{s.t.} \quad & Nx = 0 && (15.16)\\ & l \le x \le u && (15.17) \end{aligned}$$

It turns out that the circulation problem is equivalent with transshipment problem.

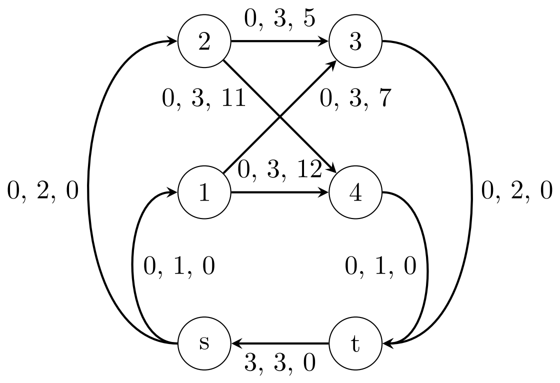
We will show how to transform any transshipment into circulation.
Let $(D, b, w)$ be a transshipment problem and define two new vertices $s$ and $t$.

- For each supply vertex $x$ add the arc $(s, x)$ to $D$ with $l_x = 0, u_x = -b_x, w_x = 0$.
- Similarly, for each demand vertex $x$, add the arc $(x, t)$ to $D$ with $l_x = 0, u_x = b_x, w_x = 0$.
- Finally, add an arc $(t, s)$ having $w_{ts} = 0, l_{ts} = u_{ts} = \sum\{b_x : \forall x, x \text{ is demand vertex}\}$.
- Each original arc is given a $l_x = 0, u_x = \sum\{b_x : \forall x, x \text{ is a demand vertex}\}$, $w_x$ remains unchanged.

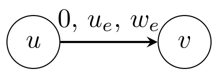
The following is a graph for transshipment problem.



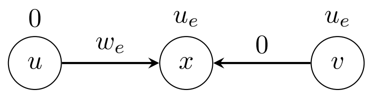
After the above procedures, it is now transformed into a circulation problem.

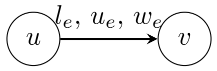


Then, we will show how to transform any circulation problem into transshipment problem.
If the lower bound of the arc is zero, i.e., $l_e = 0$, then for each such arc $(u, v)$, introduce a vertex in between $u$ and $v$, replace the arc $e = (u, v)$ by $e_1 = (u, x)$ and $e_2 = (x, v)$. Both arcs are uncapacitated. Let $w_1(u, x) = w(u, v)$ and $w_2(x, v) = 0$ be the new weights for the arcs. Let $u_e$ be the demands of newly added vertex $x$ and add $u_e$ to the supplies of vertex $v$ (in $v$ the supplies is the summation from all arcs that go to $v$).



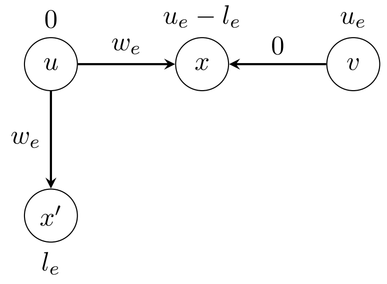
Will be transform into
If the lower bound of the arc is not zero, i.e., $l_e \neq 0$, then for each such arc $(u, v)$, introduce two new vertices, one vertex is in between $u$ and $v$, similar to the previous case, the difference is the demands of this new vertex is





$u_e - l_e$, the others stays the same. Then add the other vertex, this vertex, denoted as $x'$ is added along with an arc between $u$ and $x'$, the weight of this arc is $w(u,v)$, the demands on this new vertex is $l_e$.

Will be transform into



Perform the above procedures to all the arcs in a circulation problem. In $O(E)$ (polynomially times) transformation, such problem can be transformed into transshipment problem.

## 15.4 Out-of-Kilter algorithm

This algorithm is a Primal-dual method and is applied to the minimum weight circulation problem.

For LP optimality conditions we need primal feasibility, dual feasibility and complementary slackness, i.e., KKT conditions. Primal and dual feasibility are obvious so we need to show complementary slackness through following theorem.

**Theorem 15.3.** *Let $x$ be a feasible circulation flow for $(D, l, u, w)$. And suppose there exists a real value vector $\{y_i : i \in V\}$ which we called **vertex-numbers**. For all edges $e \in A$*

$$y_{h(e)} - y_{t(e)} > w_e \text{ implies } x_e = u_e \tag{15.18}$$
$$y_{h(e)} - y_{t(e)} < w_e \text{ implies } x_e = l_e \tag{15.19}$$

*Then $x$ is optimal to the circulation problem.*

*Proof.* For each $e \in A$ define

$$\gamma_e = \max\{y_{h(e)} - y_{t(e)} - w_e, 0\} \tag{15.20}$$
$$\mu_e = \max\{w_e - y_{h(e)} + y_{t(e)}, 0\} \tag{15.21}$$

Then

$$\gamma_e - \mu_e = y_{h(e)} - y_{t(e)} - w_e \tag{15.22}$$

Furthermore

$$\sum_{e\in A}(\mu_e l_e - \gamma_e u_e) \tag{15.23}$$

$$= \sum_{e\in A}(\mu_e l_e - \gamma_e u_e) + \sum_{i\in V} y_i(\sum_{h(e)=i} x_e - \sum_{t(e)=i} x_e) \tag{15.24}$$

$$= \sum_{e\in A}(\mu_e l_e - \gamma_e u_e + x_e(y_{h(e)} - y_{t(e)})) \tag{15.25}$$

$$= \sum_{e\in A}(\mu_e l_e - \gamma_e u_e + x_e(\gamma_e - \mu_e + w_e)) \tag{15.26}$$

$$= \sum_{e\in A}(\gamma_e(x_e - u_e) + \mu_e(l_e - x_e) + x_e w_e) \tag{15.27}$$

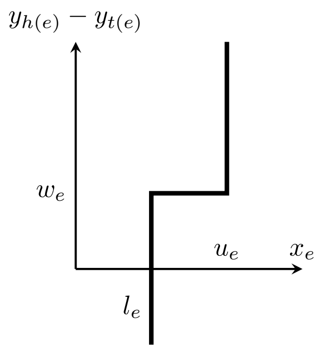
$$\leq \sum_{e\in A} x_e w_e \tag{15.28}$$

The last inequality will be satisfied as equality iff the first two hold. □

The following is the formulation of circulation problem

$$\begin{aligned}
(\text{P}) \quad \min \quad & wx && (15.29)\\
\text{s.t.} \quad & Nx = 0 \quad y && (15.30)\\
& x \geq l \quad z^l && (15.31)\\
& -x \leq -u \quad z^u && (15.32)\\
(\text{D}) \quad \max \quad & lz^l - uz^u && (15.33)\\
(\text{s.t.}) \quad & yN^{-1} + z^l - z^u \leq w && (15.34)\\
& y \quad free && (15.35)\\
& z^l, z^u \geq 0 && (15.36)\\
(\text{CS}) \quad & y_{h(e)} - y_{t(e)} > w_e \Rightarrow x_e = u_e && (15.37)\\
& y_{h(e)} - y_{t(e)} < w_e \Rightarrow x_e = l_e && (15.38)
\end{aligned}$$

There is an alternative way of circulation optimality for a circulation problem. We define a **kilter-diagram** as follows.
For every edge construct the following:

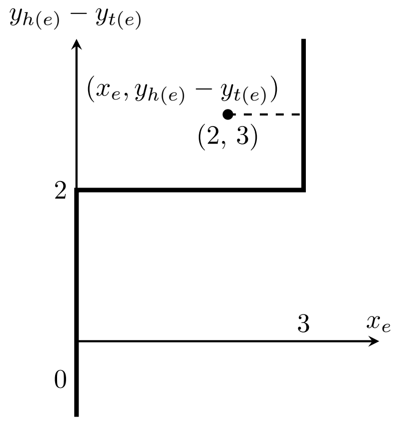


For each point $(x_e, y_{h(e)} - y_{t(e)})$ we define a **kilter-number** $k_e$, be the minimum positive distance change in $x_e$ required to put in on the kilter line.

**Example.** For edge $e : w_e = 2, l_e = 0, u_e = 3$, assume $x_e = 2, y_{h(e)} - y_{t(e)} = 3$, then $k_e = 1$

**Lemma 15.4.** *If for every circulation $x$ and vertex number $y$ we have $\sum_{e\in A} k_e = 0$, then $x$ is optimal.*

*Proof.* Since $k_e$ is a nonnegative number, then the only way that $\sum_{e\in A} k_e = 0$ is $k_e = 0, \forall e \in A$, which means $\forall e \in A, l_e \leq x_e \leq u_e$. Furthermore, the complementary slackness are satisfied. □



General idea of algorithm follows. Suppose we are given a circulation $x$ and vertex-numbers $y$ (we do not require feasibility). Usually we pick $x = 0, y = 0$. If every edge is in kilter-line then we are optimal.
Otherwise there is at least one edge $e^*$ that is out-of kilter. The algorithm consist of two phases, one called **flow-change** phase (horizontally), then other **number-change** phase (vertically).
In the flow-change phase, we want to find a new circulation for an out-of-kilter edge $e^*$ say $\hat{e}$ such that we reduce the kilter number $k_{e^*}$, without increasing any other kilter number for other edges.
To do this, denote the edges of $e^*$ to be $s$ and $t$, where such that $k_{e^*}$ will be decreased by increasing the flow from $s$ to $t$ on $e^*$.
If $e^* = (s, t)$ this will accomplished by increasing $x_{e^*}$ and if $e = (t, s)$ it is accomplished by decreasing $x_{e^*}$.
To do this we look for an $(s, t)$-path $p$ of the following edges.

- If $e$ is forward in $p$, then increasing $x_e$ does not increase $k_e$ and
- If $e$ is reversed in $p$, then decreasing $x_e$ dose not increase $k_e$

In terms of kilter diagram, an arc satisfies "forward" if it is forward and in left side of kilter line, and it satisfies "reversed" if it is reverse and in right side of kilter line.
Suppose we can not find such a path. From $s$ to $t$, let $x$ be the vertices that can decrease by an augmenting path. Then either we can change the vertex numbers $y$ so that $\sum_{e \in A} k_e$ does not increase but $x$ does, or we can show that problem is infeasible.
INPUT a minimum circulation problem $(D, l, u, w)$ a circulation $x$ and vertex-numbers $y$
OUTPUT conclusion that $(D, l, u, w)$ is infeasible or an minimum weighted flow.
Step 1: If every arc is in kilter ($k_e = 0, \forall e \in A$). Stop with $x$ is optimal. Otherwise let $e^*$ be an out-of-kilter arc. If increasing $x_{e^*}$ decreases $k_{e^*}$ set $s = h(e^*)$ and $t(e^*)$ otherwise set $s = t(e^*)$ and $t = h(e^*)$
Step 2: If there exists an $(s, t)$ augmenting path $p$ then goto Step 3, otherwise goto Step 4.
STEP 3: Set $y_e = y_{h(e)} - y_{t(e)}, e \in A$ Set $\Delta_1 = \min\{u_e - x_e : e \text{ is forward and } y_e \geq w_e\}$ Set $\Delta_2 = \min\{l_e - x_e : e \text{ is forward and } y_e < w_e\}$ Set $\Delta_3 = \min\{x_e - l_e : e \text{ is reverse and } y_e \leq w_e\}$ Set $\Delta_4 = \min\{x_e - u_e : e \text{ is reverse and } y_e > w_e\}$ $\Delta = \min\{\Delta_i, i = 1, 2, 3, 4\}$
Increase $x_e$ by $\Delta$ on each forward arc in $p$, decrease $x_e$ by $\Delta$ on each reverse arc in $p$.
If $e^* = (s, t)$ decrease $x_{e^*}$ by $\Delta$, otherwise increase $x_{e^*}$ by $\Delta$
If $k_{e^*} > 0$ goto Step 2. otherwise goto Step 1.
Step 4: Let $X$ be the set of vertices reachable from $s$ by augmenting paths, then $t \notin X$, if every arc $e$ with $h(e) \in X$ has $x_e \leq l_e$ and every arc $e$ with $t(e) \in X$ has $x_e \geq u_e$, and at least one of the above inequality is strict, then Stop with problem infeasible
Otherwise set $\delta_1 = \min\{w_e - y_e : t(e) \in X, y_e < w_e, x_e \leq u_e \neq l_e\}$ $\delta_2 = \min\{y_e - w_e : h(e) \in X, y_e > w_e, x_e \geq u_e \neq l_e\}$ $\delta = \min\{\delta_1, \delta_2\}$
Set $y_i = y_i + \delta$ for $i \notin X$
If $k_{e^*} > 0$, goto Step 2, otherwise goto Step 1.
Out-of-kilter takes $O(|E||V|K)$ where $K = \sum_{e \in A} k_e$. However, there is an algorithm called **scaling algorithm** that uses out-of-kilter as subroutine that runs in $O(R|E|^2|V|)$ where $R = \lceil \max\{\log_2 u_e : e \in A\} \rceil$

## 15.5 Complexity of Different Minimum Weighted Flow Algorithms

Let arc capacities between 1 and $U$, costs between $-C$ and $C$

| Year | Discoverer | Method | Big $O$ |
|---|---|---|---|
| 1951 | Dantzig | Network Simplex | $O(E^2V^2U)$ |
| 1960 | Minty, Fulkerson | Out-of-Kilter | $O(EVU)$ |
| 1958 | Jewell | Successive Shortest Path | $O(EVU)$ |
| 1962 | Ford-Fulkerson | Primal Dual | $O(EV^2U)$ |
| 1967 | Klein | Cycle Canceling | $O(E^2CU)$ |
| 1972 | Edmonds-Karp, Dinitz | Capacity Scaling | $O(E^2 \log U)$ |
| 1973 | Dinitz-Gabow | Improved Capacity Scaling | $O(EV \log U)$ |
| 1980 | Rock, Bland-Jensen | Cost Scaling | $O(EV^2 \log C)$ |
| 1985 | Tardos | $\epsilon$-optimality | poly$(E, V)$ |
| 1988 | Orlin | Enhanced Capacity Scaling | $O(E^2)$ |

# Chapter 16

# Matchings

## 16.1 Maximum Matching

**Definition 16.1.1** (Matching)**.** Let $G = (V, E)$ be a graph, a **matching** is a subset of edges $M \subseteq E$ such that no two elements of $M$ are adjacent. The two ends of an edge in $M$ are said to be **matched under** $M$. A matching $M$ saturates a vertex $v$, and $v$ is said to be **M-saturated** or **M-covered**, if some edge of $M$ is incident with $v$. Otherwise, $v$ is **M-unsaturated** or **M-exposed**.

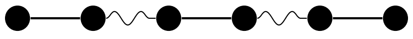
**Definition 16.1.2** (Perfect matching, Maximum matching)**.** If every vertex of $G$ is M-saturated, then the matching is said to be **perfect matching**. $M$ is a **maximum matching** if $G$ has no matching $M'$ with $|M'| > |M|$. Every perfect matching is maximum. The maximum matching does not necessarily to be perfect. Perfect matching and maximum matching may not be unique.

**Definition 16.1.3** (M-alternating)**.** An **M-alternating** path in $G$ is a path whose edges are alternately in $E \setminus M$ and $M$.

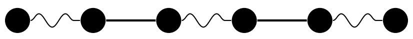
**Definition 16.1.4** (M-augmenting)**.** An **M-augmenting** path in $G$ is an $M$-alternating path whose origin and terminus are $M$-unsaturated.

**Lemma 16.1.** *Every augmenting path $P$ has property that let $M' = P\Delta M = (M \cup P) \setminus (M \cap P)$ then $M'$ contains one more edge then $M$*

The following path is an $M$-augmenting path



The following path is $M' = P\Delta M(M \cup P) \setminus (M \cap P)$ and all the vertices are $M$-saturated.



**Theorem 16.2** (Berge, 1957)**.** *A matching $M$ in a graph $G$ is maximum iff $G$ has no M-augmenting path.*

*Proof.* ($\Rightarrow$) It is clear that if $M$ is maximum, it has no augmenting paths since otherwise by problem claim we can increase by one.
($\Leftarrow$) Suppose $M$ is not maximum and let $M'$ be a bigger matching. Let $A = M\Delta M'$ now no vertex of $G$ is incident to more than two members of $A$. For otherwise either two members of $M$ or two members of $M'$ would be adjacent. Contradict the definition of matching. It follows that every component of the edges incident subgraph $G[A]$ is either an even cycle with edge augmenting in $M\Delta M'$ or else $A$ path with edges alternating between $M$ and $M'$.
Since $|M'| \geq |M|$ then the even cycle cannot help because exchanging $M$ and $M'$ will have same cardinality.
The path case implies that $p$ is alternating in $M$ and since $|M'| > |M|$ the end arc exposed so that $p$ is augmenting. □

**Definition 16.1.5** (Vertex-cover)**.** The **vertex-cover** is a subset of vertices $X$ such that every edge of $G$ is incident to some member of $X$.

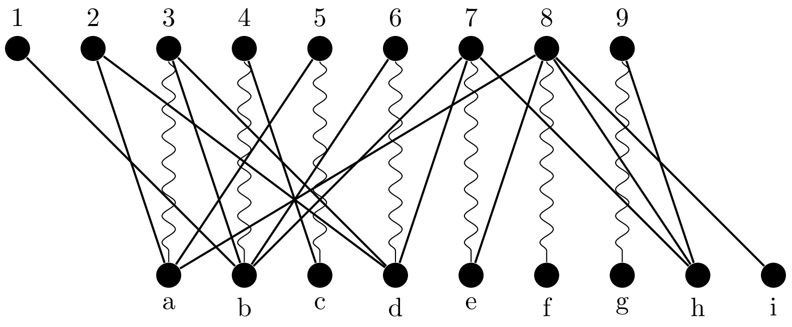
**Lemma 16.3.** *The cardinality of any matching is less than or equal to the cardinality of any vertex cover.*

*Proof.* Consider any matching. Any vertex cover must have nodes that at least incident to the edges in the matching. Since all the edges in the matching are disjointed, so for a single node can at most cover one edge in the matching. If the matching is not perfect, for the edges that not in the tree, they may or may not be possible to be covered by the nodes incident to the edges in the matching, with an easy triangle graph example, we can prove this lemma. □

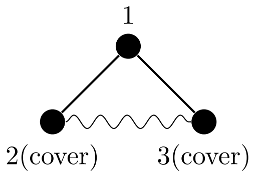
**Theorem 16.4** (König Theorem)**.** *If $G$ is bipartite, the cardinality of the maximum matching is equal to the cardinality of the minimum vertex cover.*

*Proof.* Let $G$ be a bipartite graph, $G = (V, E)$ where $V = X \cup Y$ as $X$ and $Y$ are two disjointed sets of vertices. Let $M$ be a maximum matching on $G$. For each edge in $M$, denoted by $e_i = a_i b_i$ where $e_i \in M$, $a_i \in A$ and $b_i \in B$ and $A = \{a_i : e_i \in M\} \subseteq X$, and $B = \{b_i : e_i \in M\} \subseteq Y$. Therefore, we can partition $X$ by $A$ and $U = X \setminus A$, partition $Y$ by $B$ and $W = Y \setminus B$.
We can further partition the matching $M$ into $M_1$ and $M_2$. For all the edges in $M_1$ can be included into an $M$-alternating path starts from a vertex in $U$ (which includes the edges directly linked to vertices in $U$), and $M_2 = M \setminus M_1$. For edges in $M_1$, we take the ends of edges in $B$ in the vertex cover, denoted by $B_1$, take the ends of edges in $A$ as a subset denoted by $A_1 \subseteq A$. For the edges in $M_2$, we take the ends of edges in $A$ in the vertex cover, denoted by $A_2$, and the ends of edges in $B$ as a subset denoted by $B_2 \subseteq B$.
We claim that all the vertices in $U$ can only be connected to vertices in $B_1$ and vertices in $W$ can only be connected to vertices in $A_2$.
$U \subset X$ connects to vertices in $B_1$ by definition. If vertices in $W \subset Y$ is connected to vertices in $A_1$, then we will have $M$-augmenting path which is contradicted to the assumption that $M$ is maximum matching. □

The following is an example. Where the edge in the matching that accessible from members of $U = \{1, 2\}$ in an $M$-alternating path is edge $3a, 4b, 5c, 6d$.



In which $U = \{1, 2\}$, $M_1 = \{3a, 4b, 5c, 6d\}$. $U = \{1, 2, \}, A_1 = \{3, 4, 5, 6\}, A_2 = \{7, 8, 9\}, W = \{h, i\}, B_1 = \{a, b, c, d\}, B_2 = \{e, f, g\}$. The vertex cover is $\{a, b, c, d, 7, 8, 9\}$.
The above theorem does not apply to non-bipartite graph. The following is an example



The maximum matching has one edge, where the minimum cover has two vertices.

## 16.2 Maximum Matching Algorithm

**Definition 16.2.1** (M-alternating tree)**.** An **M-alternating tree** $T$ is a rooted tree satisfied the following condition:

- The root $r$ is $M$-unsaturated
- The unique path from $r$ to any vertex $T$ is $M$-alternating
- Every vertex in $T$, except $r$ is incident to a matching edge of $T$

A vertex $x$ of $T$ is called **inner** if $(r, s)$-path in $T$ has an odd number of edges. Otherwise $x$ is called **outer**.

**Lemma 16.5.** *Let $M$ be a matching in $G$ and let $T$ be an $M$-alternating tree with root $r$, then the following conclusion hold*

- *If $v \neq r$ is an outer vertex and $p$ is the unique $(r, v)$-path in $T$ then the edge of $p$ incident to $v$ is in $M$*
- *The number of inner vertices in $T$ equals the number of matching edges in $T$.*

**Definition 16.2.2** (Alternating forest)**.** An **alternating forest** is a forest of $G$ where every components is an alternating tree. An alternating forest $(F, e)$ is an alternating forest to be then with an edge $e = M, V$ where $M$ and $V$ are outer vertices contained in two distinct components of $F$.

**Example** (Hungarian forest)**.** A **Hungarian forest** $F$ is an alternating forest containing all exposed vertices of $G$ and such that the outer vertices of $F$ are adjacent in $G$ only to inner vertices of $F$

**Definition 16.2.3** (Augmenting forest)**.** An **augmenting forest** $(F, e)$ where $F$ is an augmenting forest and $e = uv$ connects two outer vertices in distinct components of $F$.

The plan is to grow an alternating forest that eventually become augmenting or Hungarian. Augmenting forest will increase the cardinality of the match, Hungarian implies that you have found optimal maximum cardinality matching.

**Theorem 16.6.** *Let $(F, e)$ be an augmenting forest. And let $T_1$ and $T_2$ be the two components of $F$ containing an end of $e$. Let $p_i$ be the unique path in $T_i$ from the root to the end of $e$, then $p_1 e p_2^{-1}$ is an augmenting path.*

**Theorem 16.7.** *If $F$ is a Hungarian forest for some matching $M$ then $M$ is a maximum match.*

The above theorems suggest a method for computing maximum matching. Let $M$ be a matching of $G$ and let $F$ be an alternating forest in $G$ made up of all M-exposing vertices. If $F$ happens to be Hungarian, stop with the max matching $M$. If $(F, e)$ for some $e$ is augmenting then we increase our matching by 1 and start process.
Suppose $F$ is neither Hungarian nor augmenting, by definition, there must be an edge $e$ incident to an outer vertex of $F$ to no inner vertex of $F$. But $e$ cannot be incident to two outer vertices of $F$ in distinct components, since $(F, e)$ is not augmenting. Hence there are only two cases:
Case 1: $e = uv$ where $u$ is outer in $F$ and $v$ is not outer in $F$. The only way its possible if $v$ is covered. Augmenting $F$ to $M$ will increase the matching.
Case 2: $e = uv$ where $u$ and $v$ are outer vertices in $F$. Let $r$ be the root of the component of $F$ containing $u$ and $v$, let $p_u$ and $p_v$ be $(r, u)$-path and $(r, v)$-path in $F$. Let $b$ be the last vertex these two paths have in common and let $p$ be the $(u, v)$-path in $F$, let $p'_u$ be $(b - u)$-path and $p'_v$ be the $(b, v)$-path respectively. Let $n$ be the length of $p_u$, let $m$ be the length of $p_v$, let k be the length of $(r, b)$-path. Let $c$ be the cycle $(p'_u e {p'_v}^{-1})$. Number of vertices in $c$ is $n + m - 2k + 1$, $n, m$ are even, so $c$ is always an odd length cycle. If $G$ has no odd cycles, we call those graph bipartite. To this case can not happen in bipartite graph so algorithm without case 2 will solve bipartite matching.
If a graph has no odd cycles, i.e., bipartite, then we have an algorithm using augmenting forest and Hungarian forest and case 1.
We now have to deal with odd cycles. The idea is to "shrink" add cycles to a super node
Let $S \subseteq E(G)$, denote by $G : S$ the subgraph with edge set $S$

$$G : S = G \setminus (E(G) - S) \tag{16.1}$$

The contraction of $S$ to be the $G \setminus S$ with $E(G/S) = E(G) - S$. $V(G/S)$ to be the components of $G : S$ and if $e \in E(G/S)$ then the ends of $e$ in $G/S$ are in components of $G : S$ containing both ends in $G$
Let network to general case 2. $b$ is outer vertex. Let $B$ be the set of edges of cycle $C$, we call $B$ a **blossom**. We propose to replace $M$ by $M - B$, $G$ by $G/B$ and $F$ by $F/B$

## 16.3 Edmonds's Blossom Algorithm

$O(|V|^4)$ - Non-bipartite matching is one of very few problems in $P$, for which LP relaxation will not provide optimal solution.

## 16.4 Hall's Marriage Theorem

## 16.5 Transversal Theory

## 16.6 Menger's Theorem

## 16.7 The Hungarian Algorithm

# Chapter 17

# Colorings

# Part III

# Integer and Combinatorial Programming

# Chapter 18

# Formulation

## 18.1 Typical Problems

## 18.2 Integer Programming Formulation Skills

### 18.2.1 A Variable Taking Discontinuous Values

In algebraic notation:

$$x = 0, \quad \text{or} \quad l \le x \le u \tag{18.1}$$

Modeling:

$$x \le uy \tag{18.2}$$
$$x \ge ly \tag{18.3}$$
$$y \in \{0, 1\} \tag{18.4}$$

where

$$y = \begin{cases} 0, & \text{if } x = 0 \\ 1, & \text{if } l \le x \le u \end{cases} \tag{18.5}$$

### 18.2.2 Fixed Costs

In algebraic notation:

$$C(x) = \begin{cases} 0 & \text{for } x = 0 \\ k + cx & \text{for } x > 0 \end{cases} \tag{18.6}$$

Modeling:

$$C^*(x, y) = ky + cx \tag{18.7}$$
$$x \le My \tag{18.8}$$
$$x \ge 0 \tag{18.9}$$
$$y \in \{0, 1\} \tag{18.10}$$

where

$$y = \begin{cases} 0, & \text{if } x = 0 \\ 1, & \text{if } x \ge 0 \end{cases} \tag{18.11}$$

### 18.2.3 Either-or Constraints

In algebraic notation:

$$\sum_{j \in J} a_{1j} x_j \le b_1 \text{ or } \sum_{j \in J} a_{2j} x_j \le b_2 \tag{18.12}$$

Modeling:

$$\sum_{j\in J} a_{1j}x_j \le b_1 + M_1 y \tag{18.13}$$

$$\sum_{j\in J} a_{2j}x_j \le b_2 + M_1(1-y) \tag{18.14}$$

$$y \in \{0,1\} \tag{18.15}$$

where

$$y = \begin{cases} 0, & \text{if } \sum_{j\in J} a_{1j}x_j \le b_1 \\ 1, & \text{if } \sum_{j\in J} a_{2j}x_j \le b_2 \end{cases} \tag{18.16}$$

Notice that the sign before $M$ is determined by the inequality $\ge$ or $\le$, if it is "$\ge$", use "$-$", if it "$\le$", use "$+$".

### 18.2.4 Conditional Constraints

If constraint A is satisfied, then constraint B must also be satisfied

$$\text{If} \quad \sum_{j\in J} a_{1j}x_j \le b_1 \text{ then } \sum_{j\in J} a_{2j}x_j \le b_2 \tag{18.17}$$

The key part is to find the opposite of the first condition. We are using $A \Rightarrow B \Leftrightarrow \neg B \Rightarrow \neg A$
Therefore it is equivalent to

$$\sum_{j\in J} a_{1j}x_j > b_1 \text{ or } \sum_{j\in J} a_{2j}x_j \le b_2 \tag{18.18}$$

Furthermore, it is equivalent to

$$\sum_{j\in J} a_{1j}x_j \ge b_1 + \epsilon \text{ or } \sum_{j\in J} a_{2j}x_j \le b_2 \tag{18.19}$$

Where $\epsilon$ is a very small positive number.
Modeling:

$$\sum_{j\in J} a_{1j}x_j \ge b_1 + \epsilon - M_2 y \tag{18.20}$$

$$\sum_{j\in J} a_{2j}x_j \le b_2 + M_2(1-y) \tag{18.21}$$

$$y \in \{0,1\} \tag{18.22}$$

### 18.2.5 Special Ordered Sets

Out of a set of yes-no decisions, at most one decision variable can be yes. Also known as SOS1.

$$x_1 = 1, x_2 = x_3 = \cdots = x_n = 0 \tag{18.23}$$

$$\text{or} \tag{18.24}$$

$$x_2 = 1, x_1 = x_3 = \cdots = x_n = 0 \tag{18.25}$$

$$\text{or ...} \tag{18.26}$$

Modeling:

$$\sum_{i} x_i = 1, \quad i \in N \tag{18.27}$$

Out of a set of binary variables, at most two variables can be nonzero. In addition, the two variables must be adjacent to each other in a fixed order list. Also known as SOS2. Modeling: If $x_1, x_2, ..., x_n$ is a SOS2, then

$$\sum_{i=1}^{n} x_i \le 2 \tag{18.28}$$

$$x_i + x_j \le 1, \forall i \in \{1,2,...,n\}, j \in \{i+2, i+3, ..., n\} \tag{18.29}$$

$$x_i \in \{0,1\} \tag{18.30}$$

There is another type of definition, that is out of a set of nonnegative variables **not binary here**, at most two variables can be nonzero. In addition, the two variables must be adjacent to each other in a fixed order list. All variables summing to 1.
This definition of SOS2 is used in Piecewise Linear Formulations.

### 18.2.6 Piecewise Linear Formulations

The objective function is a sequence of line segments, e.g. $y = f(x)$, consists $k-1$ linear segments going through $k$ given points $(x_1, y_1), (x_2, y_2), ..., (x_k, y_k)$.
Denote

$$d_i = \begin{cases} 1, & x \in (x_i, x_{i+1}) \\ 0, & \text{otherwise} \end{cases} \tag{18.31}$$

Then the objective function is

$$\sum_{i \in \{1,2,...,k-1\}} y = d_i f_i(x) \tag{18.32}$$

Modeling: Given that objective function as a piecewise linear formulation, we can have these constraints

$$\sum_{i \in \{1,2,...,k-1\}} d_i = 1 \tag{18.33}$$
$$d_i \in \{0,1\}, i \in \{1,2,...,k-1\} \tag{18.34}$$
$$x = \sum_{i \in \{1,2,...,k\}} w_i x_i \tag{18.35}$$
$$y = \sum_{i \in \{1,2,...,k\}} w_i y_i \tag{18.36}$$
$$w_1 \le d_1 \tag{18.37}$$
$$w_i \le d_{i-1} + di, i \in \{2,3,...,k-1\} \tag{18.38}$$
$$w_k \le d_{k-1} \tag{18.39}$$

In this case, $w_i \in SOS2$ (second definition)

### 18.2.7 Conditional Binary Variables

Choose at most $n$ binary variable to be 1 out of $x_1, x_2, ...x_m, m \ge n$. If $n = 1$ then it is SOS1.
Modeling:

$$\sum_{k \in \{1,2,...,m\}} x_k \le n \tag{18.40}$$

Choose exactly $n$ binary variable to be 1 out of $x_1, x_2, ...x_m, m \ge n$
Modeling:

$$\sum_{k \in \{1,2,...,m\}} x_k = n \tag{18.41}$$

Choose $x_j$ only if $x_k = 1$
Modeling:

$$x_j = x_k \tag{18.42}$$

"and" condition, iff $x_1, x_2, ..., x_m = 1$ then $y = 1$
Modeling:

$$y \le x_i, i \in \{1,2,...,m\} \tag{18.43}$$
$$y \ge \sum_{i \in \{1,2,...,m\}} x_i - (m-1) \tag{18.44}$$

### 18.2.8 Elimination of Products of Variables

For variables $x_1$ and $x_2$,

$$y = x_1 x_2 \tag{18.45}$$

Modeling: If $x_1, x_2$ are binary, it is the same as "and" condition of binary variables.
If $x_1$ is binary, while $x_2$ is continuous and $0 \leq x_2 \leq u$, then

$$y \leq u x_1 \tag{18.46}$$
$$y \leq x_2 \tag{18.47}$$
$$y \geq x_2 - u(1 - x_1) \tag{18.48}$$
$$y \geq 0 \tag{18.49}$$

If both $x_1$ and $x_2$ are continuous, it is non-linear, we can use Piecewise linear formulation to simulate.

# Chapter 19

# Polyhedral Analysis

## 19.1 Polyhedral and Dimension

### 19.1.1 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 $d$

### 19.1.2 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** $\partial S$ 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 ">"

### 19.1.3 Hyperplane and half-space

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

### 19.1.4 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.

### 19.1.5 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

## 19.2 Face and Facet

### 19.2.1 Valid Inequalities and Faces

- The inequality denoted by $(\pi, \pi_o)$ is called a **valid inequality** for $P$ if $\pi x \leq \pi_0, \forall x \in P$
- Note that $(\pi, \pi_0)$ is a valid inequality iff $P$ lies in the half-space $\{x \in \mathbb{R}^n | Ax \leq b\}$
- If $(\pi, \pi_0)$ 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 $(\pi, \pi_0)$

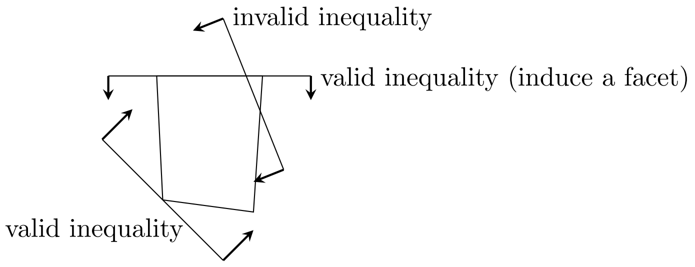


Figure 19.1: Example of valid/invalid inequality

**represents** or **defines** $F$
- A face is said to be **proper** if $F \neq \emptyset$ and $F \neq P$
- The face represented by $(\pi, \pi_0)$ 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

$$F = \{x \in P | \pi^T x = \pi_0\} \tag{19.1}$$

is not empty, then $F$ is a polyhedron. Let

$$M^= \subseteq M_F^=, M_F^\leq = M \setminus M_F^= \tag{19.2}$$

then

$$F = \{x | a_i^T x = b_i, \forall i \in M_F^=, a_i^T x \leq b_i, \forall i \in M_f^\leq\} \tag{19.3}$$

### 19.2.2 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 $P$

### 19.2.3 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 $\sum_i a_i x_i = b_i$

### 19.2.4 Domination

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

$$\begin{cases} \Pi \geq \mu M, \mu > 0 \\ \Pi_0 \leq \mu M_0, \mu > 0 \\ (\Pi, \Pi_0) \neq (M, M_0) \end{cases} \tag{19.4}$$

# Chapter 20

# Branch and Bound

## 20.1 LP based Branch and Bound

### 20.1.1 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

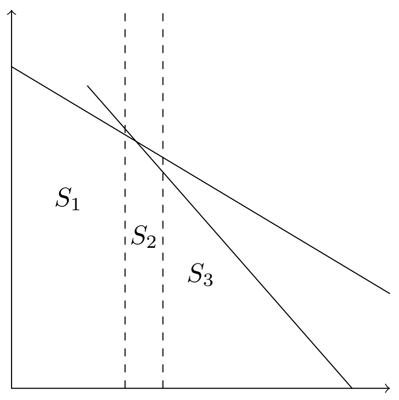


Figure 20.1: Divide and Conquer

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.

### 20.1.2 Relation Between LP Relaxation and IP

Let

$$Z_{IP} = \max_{x \in S} cx, \quad \text{where } s \text{ is a set of integer solutions} \tag{20.1}$$

$$Z_{LP} = \max cx, \quad \text{the LP relaxation of IP} \tag{20.2}$$

then

$$Z_{IP} = \max_{1\leq i\leq k}\{\max_{x\in S_i} cx\} \tag{20.3}$$

$$\text{iff} \quad S = \bigcup_{1\leq i\leq k} S_i \tag{20.4}$$

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*

$$\hat{x} \in S \rightarrow Z_{IP} \geq c\hat{x} \tag{20.5}$$

- 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 $u(S_i) \leq L$, 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)

### 20.1.3 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, $Z_{LP} \geq Z_{IP}$

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

## 20.2 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

## 20.3 Bounding

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

### 20.3.1 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

### 20.3.2 Lower Bound

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

## 20.4 Branch and Bound Algorithm

**Algorithm 14** 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 $U = u(\hat{S})$
      end if
    end if
  end if
end while
if Lower bound exists then
  find the optimal at $L$
else
  Infeasible
end if
```

## 20.5 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.

## 20.6 Choose Branching Variables

### 20.6.1 The Most Violated Integrality constraint

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

### 20.6.2 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 $D_j^-$,

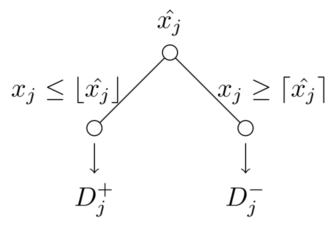

$\hat{x_j}$
$x_j \leq \lfloor \hat{x_j} \rfloor$
$x_j \geq \lceil \hat{x_j} \rceil$
$D_j^+$
$D_j^-$

Figure 20.2: Strong Branching

### 20.6.3 pseudo-cost Branching

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

$$\begin{cases} P_j^+, & \text{bound reduction if rounded up} \\ P_j^-, & \text{bound reduction if rounded down} \end{cases} \tag{20.6}$$

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

$$\begin{cases} D_j^+ = P_j^+(1 - f_j) \\ D_j^- = P_j^- f_j \end{cases} \tag{20.7}$$

## 20.7 Choose the Node to Branch

### 20.7.1 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.

### 20.7.2 Branch on Important Variables First

Branch on variables that affects many decisions.

### 20.7.3 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”

## 20.8 Types of Branching

### 20.8.1 Traditional Branching

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

$$\hat{x_j} - \lfloor \hat{x_j} \rfloor > 0 \tag{20.8}$$

Create two sub-problems

### 20.8.2 Constraint Branching

Use parallel constraints to branch, e.g.

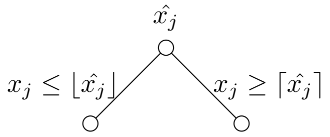

$\hat{x}_j$
$x_j \leq \lfloor \hat{x}_j \rfloor$
$x_j \geq \lceil \hat{x}_j \rceil$

Figure 20.3: Traditional Branching

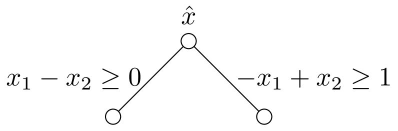


Figure 20.4: Traditional Branching

### 20.8.3 SOS

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

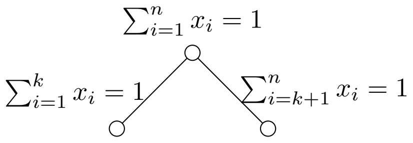


Figure 20.5: Traditional Branching

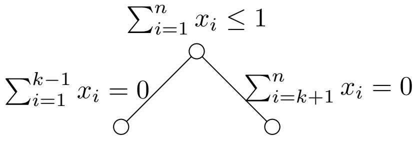

$\sum_{i=1}^{n} x_i \leq 1$
$\sum_{i=1}^{k-1} x_i = 0$
$\sum_{i=k+1}^{n} x_i = 0$

Figure 20.6: Traditional Branching

### 20.8.4 GUB

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

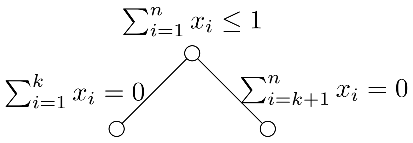


Figure 20.7: Traditional Branching

### 20.8.5 Ryan-Foster

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

$$\min \quad \sum_{i \in C} x_i \tag{20.9}$$

$$\text{s.t.} \quad \sum_{i \in C} a_{ij} x_i \geq 1, \quad \forall j \in U \tag{20.10}$$

$$x_i \in \{0, 1\}, \quad \forall i \in C \tag{20.11}$$

**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 $i$

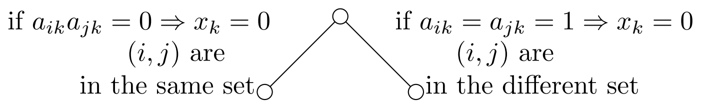

if $a_{ik}a_{jk} = 0 \Rightarrow x_k = 0$
$(i, j)$ are
in the same set
if $a_{ik} = a_{jk} = 1 \Rightarrow x_k = 0$
$(i, j)$ are
in the different set

Figure 20.8: Traditional Branching

# Chapter 21

# Branch and Cut

## 21.1 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.

### 21.1.1 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

$$y_i = \begin{cases} 1, & \text{if } i \in C \\ 0, & \text{otherwise} \end{cases} \tag{21.1}$$

Find the maximum clique via:

$$\max \quad \sum \bar{x}_i y_i \tag{21.2}$$

$$\text{s.t.} \quad y_i + y_j \leq 1, \forall \{i,j\} \notin E \tag{21.3}$$

**Option 2 - Heuristic:**

**Algorithm 15** 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: end while
5: return C
```

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

### 21.1.2 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.

## 21.2 Optional v.s. Essential Inequalities

### 21.2.1 Valid (Optional) Inequalities

See Figure 21.1

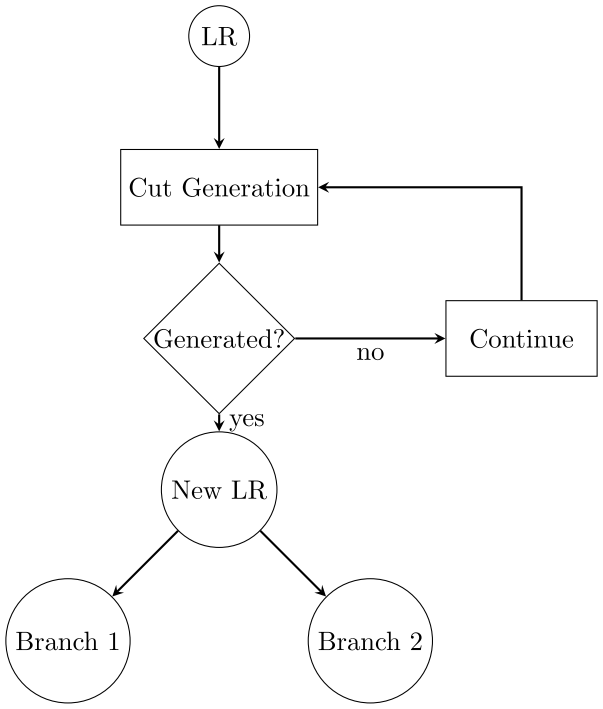


Figure 21.1: Branch and Cut for Optional Inequality

### 21.2.2 Essential Inequalities (Lazy Cuts)

See Figure 21.2

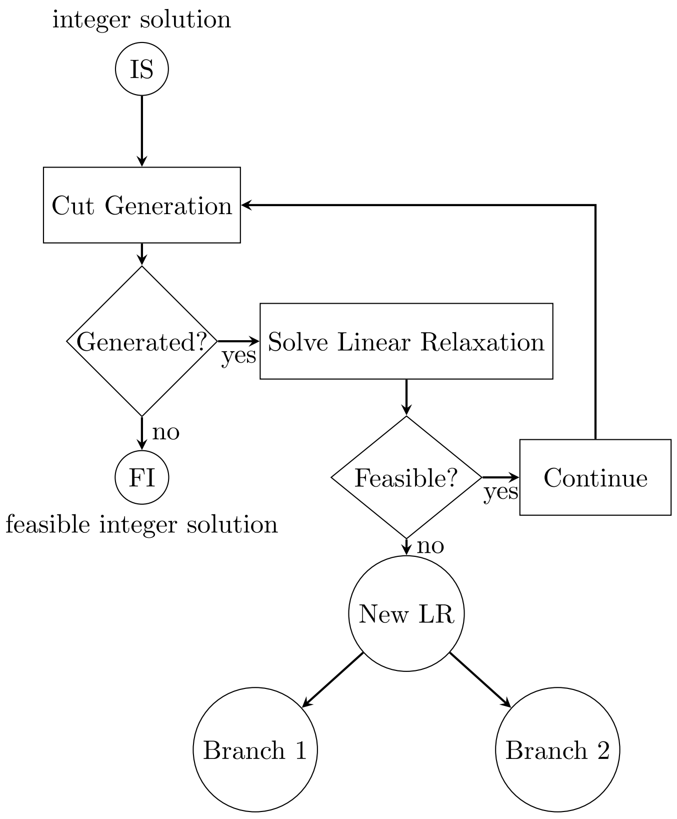


Figure 21.2: Branch and Cut for Essential Inequality

## 21.3 Chvatal-Gomory Cut

### 21.3.1 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

$$\sum_{j=1}^{n} ua_j x_j \leq ub \tag{21.4}$$

is valid
- Therefore the inequality

$$\sum_{j=1}^{n} \lfloor ua_j \rfloor x_j \leq ub \tag{21.5}$$

is valid
- Furthermore, the inequality

$$\sum_{j=1}^{n} \lfloor ua_j \rfloor x_j \leq \lfloor ub \rfloor \tag{21.6}$$

is valid.

### 21.3.2 Gomory Cutting Plane

For a IP problem

$$\begin{aligned} \max \quad & cx && (21.7)\\ \text{s.t.} \quad & Ax = b && (21.8)\\ & x \in \mathbb{B}^n && (21.9) \end{aligned}$$

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

$$\bar{x} = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix} = \begin{bmatrix} x_B \\ x_N \end{bmatrix} \tag{21.10}$$

We have

$$\begin{aligned} & Bx_B + Nx_N = b && (21.11)\\ \Rightarrow \quad & x_B + B^{-1}Nx_N = B^{-1}b && (21.12)\\ \Rightarrow \quad & x_B + [\bar{a}_1, \bar{a}_2, ...]x_N = \bar{b} && (21.13)\\ \Rightarrow \quad & x_i + \sum_{j \in NB} \bar{a}_{ij} x_j = \bar{b}_i \quad \text{(for the } i\text{th row)} && (21.14) \end{aligned}$$

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

$$x_i + \sum_{j \in NB} \lfloor \bar{a}_{ij} \rfloor x_j \leq \lfloor \bar{b}_i \rfloor \tag{21.15}$$

is a valid constraint for $P$, furthermore,

$$(\bar{b}_i - \sum_{j \in NB} \bar{a}_{ij} x_j) + \sum_{j \in NB} \lfloor \bar{a}_{ij} \rfloor x_j \leq \lfloor \bar{b}_i \rfloor \tag{21.16}$$

Move the item, we get a new Gomory Cutting Plane

$$\sum_{j \in NB} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor) x_j \geq \bar{b}_i - \lfloor \bar{b}_i \rfloor \tag{21.17}$$

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 22

# Packing and Matching

## 22.1 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.

$$PACK(G) = \{x \in \mathbb{B}^n | x_i + x_j \leq 1, \forall (i,j) \in E\} \tag{22.1}$$

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

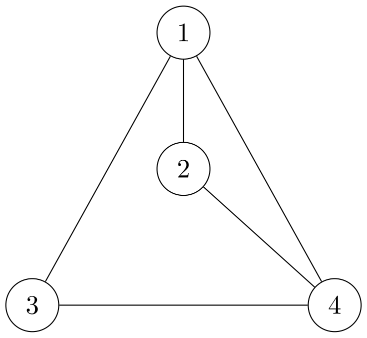


Figure 22.1: Example of vertices packing problem

$$PACK = conv\left(\begin{pmatrix}0\\0\\0\\0\end{pmatrix}, \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\\1\end{pmatrix}, \begin{pmatrix}0\\1\\1\\0\end{pmatrix}\right) \tag{22.2}$$

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.

$$MATCH(G) = \{\sum_{e \in \delta(i)} x_e \leq 1 | i \in V\} \tag{22.3}$$

## 22.2 Dimension of PACK(G)

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

$$dim(PACK(G)) = |V| \tag{22.4}$$

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

*Proof.*

$$rank\left(\begin{bmatrix}0 & I_{|V|}\\ 1 & 1\end{bmatrix}\right) = |V| + 1 \tag{22.5}$$

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

## 22.3 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.

## 22.4 Inequalities and Facets of conv(VP)

**Example:**

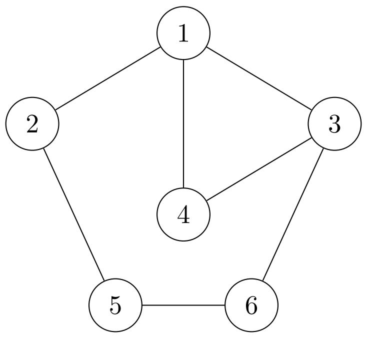


Figure 22.2: Example

### 22.4.1 Type 1 (Nonnegative Constraints)

$x_i \geq 0$ induce facets.
**Proof:**

$$rank\left(\begin{bmatrix}0 & 0\\ 0 & I_{|V|}\end{bmatrix}\right) = |V| + 1 \tag{22.6}$$

### 22.4.2 Type 2 (Neighborhood Constraints)

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

### 22.4.3 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,

$$\sum_{i \in H} x_i \leq \frac{|H| - 1}{2} \tag{22.7}$$

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

### 22.4.4 Type 4 (Maximum Clique)

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

$$\sum_{i \in C} x_i \leq 1 \tag{22.8}$$

$\boxed{\textbf{Proof:}}$
First, if $C = V$

$$rank\,([I]) = |C| = |V| \tag{22.9}$$

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.

## 22.5 Gomory Cut in Set Covering

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

$$\sum_{e \in \delta(i)} x_e \leq 1, i \in V, x_e \in \{0,1\}, e \in E \tag{22.10}$$

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

$$\sum_{i \in T} \sum_{e \in \delta(i)} x_e \leq |T| \tag{22.11}$$

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

$$2 \sum_{e \in E(T)} x_e + \sum_{e \in \delta(V \setminus T, T)} x_e \leq |T| \tag{22.12}$$

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

$$\sum_{e \in E(T)} x_e \leq \lfloor \frac{|T|}{2} \rfloor \tag{22.13}$$

# Chapter 23

# Traveling Salesman Problem

## 23.1 TSP Formulation (Asymmetric)

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

$$x_{ij} = \begin{cases} 1, & \text{if goes from } i \text{ to } j \\ 0, & \text{otherwise} \end{cases} \tag{23.1}$$

**Dantzig-Fulkerson-Johnson Formulation:**

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{23.2}$$

$$\sum_{j\in N,(i,j)\in A} x_{ij} = 1 \tag{23.3}$$

$$\sum_{i\in N,(i,j)\in A} x_{ij} = 1 \tag{23.4}$$

$$\sum_{j\notin S, i\in S,(i,j)\in A} x_{ij} = 1 \text{ or } \sum_{i,j\in S,(i,j)\in A} x_{ij} \leq |S| - 1 \tag{23.5}$$

$$\forall S \subset N, S \neq \emptyset, 2 \leq |S| \leq n-1 \tag{23.6}$$

**Miller-Tucker-Zemlin Formulation:**

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{23.7}$$

$$\sum_{j\in N,(i,j)\in A} x_ij = 1 \tag{23.8}$$

$$\sum_{i\in N,(i,j)\in A} x_ij = 1 \tag{23.9}$$

$$u_i - u_j + nx_{ij} \leq n-1 \quad i, j \in 2, ..., n, (i,j) \in A \tag{23.10}$$

$$u_1 = 1 \tag{23.11}$$

$$2 \leq u_i \leq n, i \in N, i > 1 \tag{23.12}$$

## 23.2 Sub-tour Searching Algorithm

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

$$\bar{G} = (G, \bar{A}), \bar{A} = \{(i,j) \in A | \bar{x_{ij}} = 1\} \tag{23.13}$$

denote

$$\bar{FS}(i) = \{(i,j) \in \bar{A}\} \tag{23.14}$$

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

**Algorithm 16** Sub-tour Searching Algorithm

```
$K = \emptyset$
$d_i = 0, \forall i \in N$
**for** $i \in N$ **do**
    $C = \emptyset$
    $Q = \emptyset$
    **if** $d_i == 0$ **then**
        $d_i = 1$
        $C = C \cup \{i\}$
        Q.append(i)
        **while** $Q \neq \emptyset$ **do**
            v = Q.pop()
            **for** $u \in \bar{F}S(v)$ **do**
                **if** $d_u == 0$ **then**
                    $d_u = 1$
                    $C = C \cup \{u\}$
                    Q.append(u)
                **end if**
            **end for**
        **end while**
    **end if**
    $K = K \cup C$
**end for**
```

Chapter 24

# Knapsack Problem

## 24.1 Knapsack Problem Formulation

Consider the knapsack set KNAP

$$conv(KNAP) = conv(\{x \in \mathbb{B}^n | \sum_{j \in N} a_j x_j \le b\}) \tag{24.1}$$

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

## 24.2 Valid Inequalities for a Relaxation

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

$$P_i = \{x \in \mathbb{B}^n | a_i^T x \le b_i\} \tag{24.2}$$

is a relaxation of $P$, therefore,

$$P \subseteq P_i, \forall i = 1, 2, ..., m \tag{24.3}$$

$$P \subseteq \cap_{i=1}^m P_i \tag{24.4}$$

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

## 24.3 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

$$C \subseteq N | \sum_{j \in C} a_j > b, \sum_{j \in C \setminus k} a_j < b, \forall k \in C \tag{24.5}$$

For a cover $C$, we can have the cover inequality

$$\sum_{j \in C} x_j \le |C| - 1 \tag{24.6}$$

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

$$E(C) = C \cup \{j \in N | a_j \ge a_i, \forall i \in C\} \tag{24.7}$$

is called an extended cover. Then we have,

$$\sum_{i \in E(C)} x_i \le |C| - 1 \text{ dominates } \sum_{i \in C} x_i \le |C| - 1 \tag{24.8}$$

and

$$\sum_{i \in E(C)} x_i \leq |C| - 1 \text{ dominates } \sum_{i \in E(C)} x_i \leq |E(C)| - 1 \tag{24.9}$$

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

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

$$x_j^R = \begin{cases} 1, & \text{if } j \in R \\ 0, & \text{otherwise} \end{cases} \tag{24.10}$$

Then

$$\sum_{j \in E(C)} x_j^R \geq |C| \Rightarrow |R \cap E(C)| \geq |C| \tag{24.11}$$

therefore

$$\sum_{j \in R} a_j \geq \sum_{j \in R \cap E(C)} a_j \geq \sum_{j \in C} a_j > b \tag{24.12}$$

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

## 24.4 Dimension of KNAP

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

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

## 24.5 Inequalities and Facets of conv(KNAP)

### 24.5.1 Type 1 (Lower Bound and Upper Bound Constraints):

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

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

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

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

### 24.5.2 Type 2 (Extended Cover)

Order the variables so that $a_1 \geq a_2 \geq \cdots \geq a_n$, 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 $a_{j_1} \geq a_{j_2} \geq \cdots \geq a_{j_r}$
Let $p = \min\{j | j \in N \setminus E(C)\}$
Then

$$\sum_{j \in E(C)} x_j \leq |C| - 1 \tag{24.13}$$

is a facet of $conv(KNAP)$ if

- $C = N$

$\boxed{\textbf{Proof:}}$

$$R_k = C \setminus k, \forall k \in C = N \setminus k, \forall k \in N \tag{24.14}$$

have $|N|$ affinely independent vectors

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

$\boxed{\textbf{Proof:}}$ ($j_1, j_2$ are two heaviest elements in $C$)

$$S_k = C \setminus \{j_1, j_2\} \cup \{k\}, \forall k \in E(C) \setminus C \tag{24.15}$$

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

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

$\boxed{\textbf{Proof:}}$ ($j_1$ is the heaviest element in $C$, $k$ is the lightest element outside extended cover)

$$T_k = C \setminus j_i \cup \{k\}, \forall k \in N \setminus E(C) \tag{24.16}$$

$R_k \cup T_k$ 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 $\sum_{j \in C \setminus j_1} a_j + a_p \leq b$

$\boxed{\textbf{Proof:}}$ $S_k \cup T_k$ have $|E(C) \setminus C| + |N \setminus E(C)| = |N|$ affinely independent vectors

## 24.6 Lifting

### 24.6.1 Up Lifting

For KNAP problem

$$KNAP = \{x \in \mathbb{B}^n | \sum_j a_j x_j \leq b\} \tag{24.17}$$

For $P = conv(KNAP)$ denote

$$\begin{aligned} &P_{k_1,k_2,\ldots,k_m} && (24.18)\\ &= conv(KNAP \cap \{x \in \mathbb{B}^n | x_{k_1} = x_{k_2} = \cdots = x_{k_m} = 0\}) && (24.19)\end{aligned}$$

Therefore

$$\begin{aligned} &P_{k_1,k_2,\ldots,k_m} && (24.20)\\ &= conv(KNAP \cap \{x \in \mathbb{B}^n | \sum_{j \in N \setminus \{k_1,k_2,\ldots,k_m\}} a_j x_j \leq b\}) && (24.21)\end{aligned}$$

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

$$\sum_{j \in N \setminus \{k_1,k_2,\ldots,k_m\}} x_j \leq n - m - 1 \tag{24.22}$$

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

$$\alpha_m x_m + \sum_{j \in N \setminus \{k_1,k_2,\ldots,k_m\}} x_j \leq n - m - 1 \tag{24.23}$$

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

$$\begin{aligned} \gamma &= \max\{ \sum_{j \in N \setminus \{k_1,k_2,\ldots,k_m\}} x_j | x \in P_{k_1,k_2,\ldots,k_{m-1}}, x_m = 1\} && (24.24)\\ &= \max\{ \sum_{j \in N \setminus \{k_1,k_2,\ldots,k_m\}} x_j | \sum_{j \in N \setminus \{k_1,k_2,\ldots,k_m\}} a_j x_j \leq b - a_m\} && (24.25)\end{aligned}$$

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 $conv(KNAP)$

### 24.6.2 Down Lifting

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

$$\begin{aligned} &P'_{k_1,k_2,\ldots,k_m} && (24.26)\\ &= conv(KNAP \cap \{x \in \mathbb{B}^n | x_{k_1} = x_{k_2} = \cdots = x_{k_m} = 1\}) && (24.27)\end{aligned}$$

## 24.7 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

$$\sum_{i\in C} x_i \leq |C| - 1 \tag{24.28}$$

$$\Rightarrow \quad |C| - \sum_{i\in C} x_i = \sum_{i\in C}(1 - x_i) \geq 1 \tag{24.29}$$

$$\tag{24.30}$$

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:

$$y_i = \begin{cases} 1, & \text{if } i \in C \\ 0, & \text{otherwise} \end{cases} \tag{24.31}$$

$$\min \quad \sum_{i\in N}(1 - \bar{x_i})y_i = z \tag{24.32}$$

$$\text{s.t.} \quad \sum_{i\in N} a_i y_i \geq b + 1 \tag{24.33}$$

$$y_i \in \{0,1\}, i \in N \tag{24.34}$$

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

# Chapter 25

# Network Flow Problem

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

## 25.1 Shortest Path Problem

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

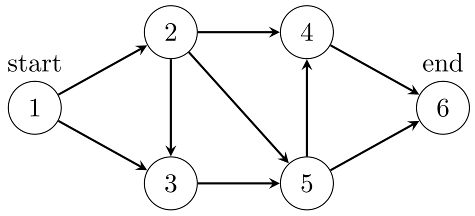


Figure 25.1: Example of directed graph

$$x_{ij} = \begin{cases} 1, & \text{if goes from } i \text{ to } j \\ 0, & \text{otherwise} \end{cases} \tag{25.1}$$

The shortest path problem can be formulated as the following:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{25.2}$$

$$\sum_{i\in N\setminus(\{S\}\cup\{E\}),(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = 0 \tag{25.3}$$

$$\sum_{i=\{S\},(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = 1 \tag{25.4}$$

$$\sum_{i=\{E\},(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = -1 \tag{25.5}$$

$$x_{ij} \in [0,1], (i,j) \in A \tag{25.6}$$

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

## 25.2 Maximum Flow Problem

$$\min \sum_{(i,j)\in A} F \tag{25.7}$$
$$\sum_{i\in N\backslash(\{S\}\cup\{E\}),(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = 0 \tag{25.8}$$
$$\sum_{i=\{S\},(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = F \tag{25.9}$$
$$\sum_{i=\{E\},(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = -F \tag{25.10}$$
$$l_{ij} \le x_{ij} \le u_{ij}, (i,j) \in A \tag{25.11}$$

In where $F$ means the flow from source to target.

## 25.3 Minimum Cost Flow

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

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{25.12}$$
$$\sum_{i\in N\backslash(\{S\}\cup\{E\}),(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = 0 \tag{25.13}$$
$$\sum_{i=\{S\},(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = 1 \tag{25.14}$$
$$\sum_{i=\{E\},(i,j)\in A} x_{ij} - \sum_{j\in N,(i,j)\in A} x_{ji} = -1 \tag{25.15}$$
$$x_{ij} \in [0,1], (i,j) \in A \tag{25.16}$$

## 25.4 Unimodularity

### 25.4.1 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.

### 25.4.2 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.
$\boxed{\textbf{Proof:}}$ By Cramer's Rule

$$x_i = \frac{detM_i}{detM} \tag{25.17}$$

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

### 25.4.3 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

### 25.4.4 Construct a New Unimodular Matrix

Let $F$ be a unimodular matrix, then

$$\begin{bmatrix} F \\ I \end{bmatrix} \tag{25.18}$$

is a unimodular matrix, also

$$\begin{bmatrix} F & 0 \\ I & I \end{bmatrix} \tag{25.19}$$

is a unimodular matrix.

# Part IV

# Nonlinear Programming

# Chapter 26

# Convex Analysis

## 26.1 Convex Sets

**Definition 26.1.1.** A set $S \in \mathbb{R}^n$ is said to be convex if $\forall x_1, x_2 \in S, \lambda \in (0,1) \Rightarrow \lambda x_1 + (1-\lambda)x_2 \in S$

The following are some familes of convex sets.

**Example.** Empty set is by convention considered as convex.

**Example.** Polyhedrons are convex sets.

**Example.** Let $P = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{x}^\top \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ where $\mathbf{A} \in \mathbb{S}_+^{n\times n}$ and $\mathbf{b} \in \mathbb{R}_+$. The set $P$ is a convex subset of $\mathbb{R}^n$.

**Example.** Let $\|.\|$ be any norm in $\mathbb{R}^n$. Then, the unit ball $B = \{\mathbf{x} \in \mathbb{R}^n | \|\mathbf{x}\| \leq b, b > 0\}$ is convex.

Let $S_1, S_2$ be convex set, then:

- $S_1 \cap S_2$ is convex set
- $S_1 \oplus S_2$ (Minkowski addition) is convex set, where
$$S_1 \oplus S_2 = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{x} = \mathbf{x_1} + \mathbf{x_2}, \mathbf{x_1} \in S_1, \mathbf{x_2} \in S_2\} \tag{26.1}$$
- $S_1 \ominus S_2$ is convex set, where
$$S_1 \oplus S_2 = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{x} = \mathbf{x_1} - \mathbf{x_2}, \mathbf{x_1} \in S_1, \mathbf{x_2} \in S_2\} \tag{26.2}$$
- $f(S_1)$ is convex iff $f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}, \mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m$

**Theorem 26.1** (Carathéodory's Theorem)**.** *Let $S \subseteq \mathbb{R}^n$. Then $\forall \mathbf{x} \in conv(S)$, there exists $\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^p \in S$, where $p \leq n+1$ such that $\mathbf{x} \in conv\{\mathbf{x}^1, \mathbf{x}^2, ...\mathbf{x}^p\}$.*

**Notice:** This theorem means, any point $\mathbf{x} \in \mathbb{R}^n$ in a convex hull of $S$, i.e., $conv(S)$, can be included in a convex subset $S' \subseteq conv(S)$ that has $n+1$ extreme points.

**Theorem 26.2.** *Let $S$ be a convex set with nonempty interior. Let $\mathbf{x}_1 \in cl(S)$ and $\mathbf{x}_2 \in int(S)$, then $\mathbf{y} = \lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \in int(S), \forall\lambda \in (0,1)$*

## 26.2 Convex Functions

**Definition 26.2.1.** Let $C \in \mathbb{R}^n$ be a convex set. A function $f : C \to \mathbf{R}$ is (resp. strictly) convex if

$$f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2) \tag{26.3}$$
$$\forall \mathbf{x}_1, \mathbf{x}_2 \in C, \forall\lambda \in (0,1) \tag{26.4}$$

(resp.)

$$f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) < \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2) \tag{26.5}$$
$$\forall \mathbf{x_1} \neq \mathbf{x}_2 \in C, \forall\lambda \in (0,1) \tag{26.6}$$

**Notice:** When calling a function convex, we imply that its domain is convex.

**Example.** Given any norm $\|.\|$ on $\mathbb{R}^n$, the function $f(x) = \|x\|$ is convex over $\mathbb{R}^n$.

**Definition 26.2.2.** Let $S$ be a nonempty convex subset of $\mathbb{R}^n$, $f : S \to \mathbb{R}$ is (resp. strictly) **concave** if $-f(x)$ is (resp. strictly) convex.

**Notice:** A function may be neither convex nor concave.

**Theorem 26.3.** *Consider $f : \mathbb{R}^n \to \mathbb{R}$. $\forall \bar{\mathbf{x}} \in \mathbb{R}^n$ and a nonzero direction $\mathbf{d} \in \mathbb{R}^n$. Define $F_{\bar{\mathbf{x}},d}(\lambda) = f(\bar{\mathbf{x}} + \lambda\mathbf{d}$. Then $f$ is (resp. strictly) convex iff $F_{\bar{\mathbf{x}},d}(\lambda)$ is (resp. strictly) convex for all $\bar{\mathbf{x}} \in \mathbb{R}^n, \forall \mathbf{d} \in \mathbb{R}^n \setminus \{0\}$.*

**Definition 26.2.3** (Level-set)**.** Given a function $f : \mathbb{R}^n \to \mathbb{R}$ and a scalar $\alpha \in \mathbb{R}$, we refer to the set $S_\alpha = \{\mathbf{x} \in S | f(\mathbf{x}) \leq \alpha\} \subseteq \mathbb{R}^n$ as the $\alpha$**-level-set** of $f$.

**Lemma 26.4.** *Let $S$ be a nonempty convex set in $\mathbb{R}^n$. Let $f : S \to \mathbb{R}^n$ be a convex function, then the $\alpha$**-level-set** of $f$ is a convex set for each value of $\alpha \in \mathbb{R}$.*

**Notice:** The converse is not necessarily true.

**Definition 26.2.4** (Epigraphs, Hypographs)**.** Let $S \in \mathbb{R}^n$ be such that $S \neq \emptyset$. The **epigraph** of $f$, denoted by $epi(f)$ is

$$epi(f) = \{(\mathbf{x}, y) \in S | \mathbf{x} \in S, y \in \mathbb{R}, y \geq f(x)\} \in \mathbb{R}^{n+1} \tag{26.7}$$

The **hypograph** of $f$, denoted by $hypo(f)$ is

$$hypo(f) = \{(\mathbf{x}, y) \in S | \mathbf{x} \in S, y \in \mathbb{R}, y \leq f(x)\} \in \mathbb{R}^{n+1} \tag{26.8}$$

**Theorem 26.5.** *Let $S$ be a nonempty convex subset in $\mathbb{R}^n$. Let $f : S \to \mathbb{R}$. Then $f$ is convex iff $epi(f)$ is convex.*

**Theorem 26.6.** *Let $S$ be a nonempty convex subset in $\mathbb{R}^n$. Let $f : S \to \mathbb{R}$ be a convex function on $S$. Then $f$ is continuous in $int(S)$.*

## 26.3 Subgradients and Subdifferentials

**Definition 26.3.1** (Subgradient)**.** Let $S$ be a nonempty convex set in $\mathbb{R}^n$. Let $f : S \to \mathbb{R}$ be a convex function, then $\xi$ is a **subgradient** of $f$ at $\bar{\mathbf{x}}$ if

$$f(\mathbf{x}) \geq f(\bar{\mathbf{x}}) + \xi^\top(\mathbf{x} - \bar{\mathbf{x}}), \forall \mathbf{x} \in S \tag{26.9}$$

**Definition 26.3.2** (Subdifferential)**.** The set of all subgradients of $f$ at $\bar{\mathbf{x}}$ is called **subdifferential** of $f$ at $\bar{\mathbf{x}}$, denoted as $\partial f(\bar{\mathbf{x}})$

**Theorem 26.7.** *Let $S$ be a nonempty convex set in $\mathbb{R}^n$. Let $f : S \to \mathbb{R}$ be a convex function. Then for $\bar{\mathbf{x}} \in int(S)$, there exists a vector $\xi$ such that*

$$f(\mathbf{x}) \geq f(\bar{\mathbf{x}}) + \xi^\top(\mathbf{x} - \bar{\mathbf{x}}), \forall \mathbf{x} \in S \tag{26.10}$$

*In particular, the hyperplane*

$$\mathcal{H} = \{(\mathbf{x}, y) | y = f(\bar{\mathbf{x}}) + \xi^\top(\mathbf{x} - \bar{\mathbf{x}})\} \tag{26.11}$$

*is a supporting plane of $epi(f)$ at $(\bar{\mathbf{x}}, f(\bar{\mathbf{x}}))$*

**Theorem 26.8.** *Let $S$ be a nonempty convex set in $\mathbb{R}^n$. Let $f : S \to \mathbb{R}$ be a convex function. Suppose that for each $\bar{\mathbf{x}} \in S$, there exists $\xi$ such that*

$$f(\mathbf{x}) \geq f(\bar{\mathbf{x}}) + \xi^\top(\mathbf{x} - \bar{\mathbf{x}}), \forall \mathbf{x} \in S \tag{26.12}$$

*Then $f$ is convex on $int(S)$*

**Notice:** Not all convex functions are continuous, it has to be continuous in its interior, but it may not be continuous at the boundary.

## 26.4 Differentiable Functions

**Definition 26.4.1** (Differentiable Functions)**.** Let $S$ be a nonempty subset of $\mathbb{R}^n$. Let $f : S \to \mathbb{R}$.Then $f$ is said to be **differentiable** at $\bar{\mathbf{x}} \in int(S)$ if there exists a vector $\nabla f(\bar{\mathbf{x}})$ and a function $\alpha : \mathbb{R}^n \to \mathbb{R}$ such that

$$f(\mathbf{x}) = f(\bar{\mathbf{x}}) + \nabla f(\bar{\mathbf{x}})^\top (\mathbf{x} - \bar{\mathbf{x}}) + \alpha(\bar{\mathbf{x}}, \mathbf{x} - \bar{\mathbf{x}})\|\mathbf{x} - \bar{\mathbf{x}}\| \tag{26.13}$$

for all $\mathbf{x} \in S$ where $\lim_{\mathbf{x} - \bar{\mathbf{x}}} \alpha(\bar{\mathbf{x}}, \mathbf{x} - \bar{\mathbf{x}}) = 0$

*Remark.* If function $f$ is differentiable, then $\nabla f(\bar{\mathbf{x}}) = (\frac{\partial f(\bar{\mathbf{x}})}{\partial \mathbf{x}_1}, \frac{\partial f(\bar{\mathbf{x}})}{\partial \mathbf{x}_2}, \cdots, \frac{\partial f(\bar{\mathbf{x}})}{\partial \mathbf{x}_n})$, and the gradient is unique.

**Lemma 26.9.** *Let $S \neq \emptyset$ be a convex set of $\mathbb{R}^n$. Let $f : S \to \mathbb{R}$ be convex. If $f$ is differentiable at $\bar{\mathbf{x}} \in int(S)$, then the subdifferential of $f$ at $\bar{\mathbf{x}}$ is the singleton, $\{\nabla f(\bar{\mathbf{x}})\}$*

**Theorem 26.10.** *Let $S$ be a nonempty subset of $\mathbb{R}^n$. Let $f : S \to \mathbb{R}$ be differentiable on $S$. Then $f$ is (resp. strictly) convex on $S$ iff $\forall \bar{\mathbf{x}} \in S$*

$$f(\mathbf{x}) \geq f(\bar{\mathbf{x}}) + \nabla f(\bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}}), \forall \mathbf{x} \in S \tag{26.14}$$

*(resp.)*

$$f(\mathbf{x}) > f(\bar{\mathbf{x}}) + \nabla f(\bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}}), \forall \mathbf{x} \neq \bar{\mathbf{x}} \in S \tag{26.15}$$

**Theorem 26.11** (Mean-value Theorem)**.** *Let $S$ be a nonempty subset of $\mathbb{R}^n$. Let $f : S \to \mathbb{R}$ be differentiable on $S$. Then for all $\mathbf{x}_1, \mathbf{x}_2 \in S$, there exists $\lambda \in (0, 1)$ such that*

$$f(\mathbf{x}_2) = f(\mathbf{x}_2) + \nabla f(\hat{\mathbf{x}})(\mathbf{x}_2 - \mathbf{x}_1) \tag{26.16}$$

*where*

$$\hat{\mathbf{x}} = \lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2 \tag{26.17}$$

Chapter 27

# KKT Optimality Conditions

# Chapter 28

# Lagrangian Duality

# Chapter 29

# Unconstrained Optimization

Chapter 30

# Penalty and Barrier Functions

and Meta-heuristic Methods