# Notes for Operations Research & More

Lan Peng, PhD Student

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

November 11, 2019

November 11, 2019

# Contents

# Part I

# Preliminary Topics

# Chapter 1

# Introduction to Optimization

## 1.1 Optimization Model

The following is the basic forms of terminology:

$$
\begin{aligned}
\text{(P)} \quad \min \quad & f(x) && (1.1)\\
\text{s.t.} \quad & g_i(x) \leq 0, \quad i = 1, 2, ..., m && (1.2)\\
& h_j(x) = 0, \quad j = 1, 2, ..., l && (1.3)\\
& x \in X && (1.4)
\end{aligned}
$$

We have

- - $x \in R^n \to X \subseteq R^m$
- $g_i(x)$ are called inequality constraints
- $h_j(x)$ are called equality constraints
- $X$ is the domain of the variables (e.g. cone, polygon, $\{0, 1\}^n$, etc.)
- Let $F$ be the feasible region of $(P)$:
  - $x^0$ is a feasible solution iff $x^0 \in F$
  - $x^*$ is an optimized solution iff $x^* \in F$ and $f(x^*) \leq f(x^0), \forall x^0 \in F$ (for minimized problem)

**Notice:** Not every $(P)$ has a feasible region, we can have $F = \emptyset$. Even if $F \neq \emptyset$, there might not be an solution to $P$, e.g. unbounded. If $(P)$ has optimized solution(s), it could be 1) Unique 2) Infinite number of solution 3) Finite number of solution

Types of Optimization Problem

- $m = l = 0$, $x \in R^n$, unconstrained problem
- $m + l > 0$, constrained problem
- $f(x), g_i(x), h_j(x)$ are linear, Linear Optimization
  - If $X = R^n$, Linear Programming
  - If $X$ is discrete, Discrete Optimization
  - If $X \subseteq Z^n$, Integer Programming
  - If $X \in \{0, 1\}^n$, Binary Programming
  - If $X \in Z^n \times R^m$, Mixed Integer Programming

# Chapter 2

# Review of Linear Algebra

## 2.1 Field

**Definition 2.1.1** (Field)**.** Let $F$ denote either the set of real numbers or the set of complex numbers.

- Addition is commutative: $x + y = y + x, \forall x, y \in F$
- Addition is associative: $x + (y + z) = (x + y) + z, \forall x, y, z \in F$
- Element 0 exists and unique: $\exists 0, x + 0 = x, \forall x \in F$
- To each $x \in F$ there corresponds a unique element $(-x) \in F$ such that $x + (-x) = 0$
- Multiplication is commutative: $xy = yx, \forall x, y \in F$
- Multiplication is associative: $x(yz) = (xy)z, \forall x, y, z \in F$
- Element 1 exists and unique: $\exists 1, x1 = x, \forall x \in F$
- To each $x \neq 0 \in F$ there corresponds a unique element $x^{-1} \in F$ that $xx^{-1} = 1$
- Multiplication distributes over addition: $x(y + z) = xy + xz, \forall x, y, z \in F$

Suppose one has a set $F$ of objects $x, y, z, ...$ and two operations on the elements of $F$ as follows. The first operation, called addition, associates with each pair of elements $x, y \in F$ an element $(x + y) \in F$; the second operation, called multiplication, associates with each pair $x, y$ an element $xy \in F$; and these two operations satisfy all conditions above. The set $F$, together with these two operations, is then called a **field**.

**Definition 2.1.2** (Subfield)**.** A **subfield** of the field $C$ is a set $F$ of complex numbers which itself is a field.

**Example.** The set of integers is not a field.

**Example.** The set of rational numbers is a field.

**Example.** The set of all complex numbers of the form $x + y\sqrt{2}$ where $x$ and $y$ are rational, is a subfield of $\mathbb{C}$.

**Notice:** In this note, we (...Lan) assume that the field involved is a subfield of the complex numbers $\mathbb{C}$. More generally, if $F$ is a field, it may be possible to add the unit 1 to itself a finite number of times and obtain 0, which does not happen in the subfield of $\mathbb{C}$. If it does happen in $F$, the least $n$ such that the sum of $n$ 1's is 0 is called **characteristic** of the field $F$. If it does not happen, then $F$ is called a field of **characteristic zero**.

## 2.2 Real Vector Spaces

## 2.3 Linear, Conic, Affine, and Convex Combinations

## 2.4 Determinants

## 2.5 Inner Products

**Definition 2.5.1** (Inner Product)**.** Let $F$ be the field of real numbers or the field of complex numbers, and $V$ a vector space over $F$. An **inner product** on $V$ is a function which assigns to each ordered pair of vectors $\alpha$, $\beta$ in $V$ a scalar $< \alpha|\beta >$ in $F$ in such a way that $\forall \alpha, \beta, \gamma \in V, c \in \mathbb{R}$ that

- $< \alpha + \beta|\gamma > = < \alpha|\gamma > + < \beta|\gamma >$
- $< c\alpha|\beta > = c < \alpha|\beta >$
- $< \alpha|\beta > = \overline{< \beta|\alpha >}$
- $< \alpha|\alpha > \geq 0$, $< \alpha|\alpha > = 0$ iff $\alpha = \mathbf{0}$

Furthermore, the above properties imply that

- $< \alpha|c\beta + \gamma > = \bar{c} < \alpha|\beta > + < \alpha|\gamma >$

**Definition 2.5.2.** On $F^n$ there is an inner product which we call the **standard inner product**. It is defined on $\alpha = (x_1, x_2, ..., x_n)$ and $\beta = (y_1, y_2, ..., y_n)$ by

$$< \alpha|\beta > = \sum_j x_j \bar{y_j} \tag{2.1}$$

For $F = \mathbb{R}^n$

$$< \alpha|\beta >= \sum_j x_j y_j \tag{2.2}$$

In the real case, the standard inner product is often called the dot product and denoted by $\alpha \cdot \beta$

**Example.** For $\alpha = (x_1, x_2)$ and $\beta = (y_1, y_2)$ in $\mathbb{R}^2$, the following is an inner product.

$$< \alpha|\beta >= x_1y_1 - x_2y_1 - x_1y_2 + 4x_2y_2 \tag{2.3}$$

**Example.** For $\mathbb{C}^{n\times n}$,

$$< \mathbf{A}|\mathbf{B} >= trace(\mathbf{B}^*\mathbf{A}) \tag{2.4}$$

is an inner product, where

$$\mathbf{A}^*_{ij} = \bar{\mathbf{A}}_{ji} \quad (\textbf{conjugate transpose}) \tag{2.5}$$

For $\mathbb{R}^{n\times n}$,

$$< \mathbf{A}|\mathbf{B} >= trace(\mathbf{B}^T\mathbf{A}) = \sum_j (AB^T)_{jj} = \sum_j \sum_k A_{jk}B_{jk} \tag{2.6}$$

## 2.6 Norms

**Definition 2.6.1** (Norms)**.** A **norm** on a vector space $\mathcal{V}$ is a function $\|\cdot\| : \mathcal{V} \to \mathbb{R}$ for which the following three properties hold for all point $\mathbf{x}, \mathbf{y} \in \mathcal{V}$ and scalars $\lambda \in \mathbb{R}$

- (Absolute homogeneity) $\|\lambda\mathbf{x}\| = |\lambda|\|\mathbf{x}\|$
- (Triangle inequality) $\|\mathbf{x} + \mathbf{y}\| \le \|\mathbf{x}\| + \|\mathbf{y}\|$
- (Positivity) Equality $\|\mathbf{x}\| = 0$ holds iff $\mathbf{x} = 0$

**Definition 2.6.2** ($L_p$-norms)**.** Let $p \ge 1$ be a real number. We define the $p$-norm of vector $\mathbf{v} \in \mathbb{R}^n$ as:

$$\|\mathbf{x}\|_p = (\sum_{i=1}^n |v_i|^p)^{\frac{1}{p}} \tag{2.7}$$

Particularly

$$\|\mathbf{v}\|_1 = \sum_{i=1}^n |v_i| \tag{2.8}$$

$$\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^n v_i^2} \tag{2.9}$$

$$\|\mathbf{v}\|_\infty = \max_{i=1}^n |v_i| \tag{2.10}$$

**Definition 2.6.3** (Frobenius norm)**.** $\mathbf{X} \in \mathbb{R}^{m\times n}$, the **Frobenius norm** is defined as

$$\|\mathbf{X}\|_F = \sqrt{trace(\mathbf{X}^\top\mathbf{X})} \tag{2.11}$$

**Definition 2.6.4** (Dual norm)**.** For an arbitrary norm $\|\cdot\|$ on Euclidean space $\mathbf{E}$, the **dual norm** $\|\cdot\|^*$ on $\mathbf{E}$ is defined by

$$\|\mathbf{v}\|^* = \max\{< \mathbf{v}|\mathbf{x} > | \|\mathbf{x}\| \le 1\} \tag{2.12}$$

For $p, q \in [1, \infty]$, the $l_p$ and $l_q$ norms on $\mathbb{R}^n$ are dual to each other whenever $\frac{1}{p} + \frac{1}{q} = 1$.

## 2.7 Eigenvectors and Eigenvalues

**Definition 2.7.1.** If $\mathbf{A}$ is an $n\times n$ matrix, then a nonzero vector $\mathbf{x} \in \mathbb{R}^n$ is called an **eigenvector** of $\mathbf{A}$ if $\mathbf{Ax}$ is a scalar multiple of $\mathbf{x}$, i.e.

$$\mathbf{Ax} = \lambda\mathbf{x} \tag{2.13}$$

for some scalar $\lambda$. The scalar $\lambda$ is called **eigenvalue** of $\mathbf{A}$ and the vector $\mathbf{x}$ is said to be an **eigenvector corresponding to** $\lambda$

**Theorem 2.1** (Characteristic Equation)**.** *If* $\mathbf{A}$ *is an* $n\times n$ *matrix, then* $\lambda$ *is an eigenvalue of* $\mathbf{A}$ *iff*

$$\det(\lambda I - A) = 0 \tag{2.14}$$

**Corollary 2.1.1.**

$$\sum \lambda_A = tr(\mathbf{A}) \tag{2.15}$$

**Corollary 2.1.2.**

$$\prod \lambda_A = \det(\mathbf{A}) \tag{2.16}$$

**Notice:** Gaussian elimination changes the eigenvalues.

## 2.8 Decompositions

# Chapter 3

# Review of Real Analysis

## 3.1 Sequences and Series

# Chapter 4

# Review of Topology

## 4.1 Open Sets and Closed Sets

**Definition 4.1.1** (Metric space)**.** A **metric space** is a set $X$ where we have a notion of distance. That is, if $x, y \in X$, then $d(x, y)$ is the distance between $x$ and $y$. The particular distance function must satisfy the following conditions:

- $d(x, y) > 0, \forall x, y \in X$
- $d(x, y) = 0 \iff x = y$
- $d(x, y) = d(y, x)$
- $d(x, z) \leq d(x, y) + d(y, z)$

**Definition 4.1.2** (Ball)**.** Let $X$ be a metric space. A **ball** $B$ of radius $r$ around a point $x \in X$ is

$$B = \{y \in X | d(x, y) < r\} \tag{4.1}$$

**Definition 4.1.3** (Open set)**.** A subset $O \subseteq X$ is **open** if $\forall x \in O, \exists r, B = \{x \in X | d(x, y) < r\} \subseteq O$

**Theorem 4.1.** *The union of any collection if open sets is open.*

*Proof.* Sets $S_1, S_2, ..., S_n$ are open sets, let $S = \cup_{i=1}^{n} S_i$, then $\forall i, S_i \subseteq S$. $\forall x \in S, \exists i, x \in S_i$. Given that $S_i$ is an open set, then for $x$, $\exists r$ that $B = \{x \in S_i | d(x, y) < r\} \subseteq S_i \subseteq S$, therefore $S$ is an open set. ☐

**Theorem 4.2.** *The intersection of any finite number of open sets is open.*

*Proof.* Sets $S_1, S_2, ..., S_n$ are open sets, let $S = \cap_{i=1}^{n} S_i$, then $\forall i, S \subseteq S_i$. $\forall x \in S, x \in S_i$. For any $i$, we can define an $r_i$, such that $B_i = \{x \in S_i | d(x, y) < r_i\} \subseteq S_i$. Let $r = \min_i\{r_i\}$. Noticed that $\forall i, B' = \{x \in S_i | d(x, y) < r\} \subseteq B_i \subseteq S_i$. Therefore $S$ is an open set. ☐

*Remark.* The intersection of infinite number of open sets is not necessarily open.

Here we find an example that the intersection of infinite number of open sets can be closed.

**Example.** Let $A_n \in \mathbb{R}$ and $B_n \in \mathbb{R}$ be two infinite series, with the following properties.

- $\forall n, A_n < a, \lim A_n = a$
- $\forall n, B_n > b, \lim B_n = b$
- $a < b$

Then we define infinite number of sets $S_i$, the $i$th set is defined as

$$S_i = (A_i, B_i) \subset \mathbb{R} \tag{4.2}$$

Then

$$S = \cap_{i=1}^{\infty} S_i = [a, b] \subset \mathbb{R} \tag{4.3}$$

and $S$ is a closed set.

**Definition 4.1.4** (Limit point)**.** A point $z$ is a **limit point** for a set $A$ if every open set $U$ that $z \in U$ intersects $A$ in a point other than $z$.

**Notice:** $z$ is not necessarily in $A$.

**Definition 4.1.5** (Closed set)**.** A set $C$ is **closed** iff it contains all of its limit points.

**Theorem 4.3.** $S \in \mathbb{R}^n$ *is closed* $\iff \forall \{x_k\}_{k=1}^{\infty} \in S, \lim_{k \to \infty} \{x_k\}_{k=1}^{\infty} \in S$

**Theorem 4.4.** *Every intersection of closed sets is closed.*

**Theorem 4.5.** *Every finite union of closed sets is closed.*

*Remark.* The union of infinite number of closed sets is not necessarily closed.

**Theorem 4.6.** *A set $C$ is a closed set if $X \setminus C$ is open*

*Proof.* Let $S$ be an open set, $x \notin S$, for any open set $S_i$ that $x \in S_i$, we can find a correspond $r_i > 0$, such that $B_i = \{x \in S_i | d(x, y) < r_i\}$. Take $r = \min_{\forall i}\{r_i\}$, set $B = \{x \notin S | d(x, y) < r\} \neq \emptyset$. Which means for any $x \notin S$, we can find at least one point $x' \in B$ that for all open set $S_i$, $x' \in S_i$, which makes $x$ a limit point of the complement of the open set. Notice that $x$ is arbitrary, then the collection of $x$, i.e., the complement of $S$ is a closed set. ☐

*Remark.* The empty set is open and closed, the whole space $X$ is open and closed.

# Part II

# Graph and Network Theory

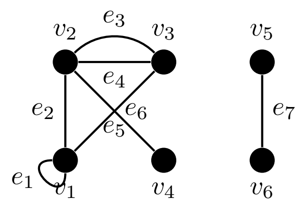
# Chapter 5

# Graphs and Subgraphs

## 5.1 Graph

**Definition 5.1.1** (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

$$V = V(G) = \{v_1, v_2, v_3, v_4, v_5, v_6\} \tag{5.1}$$
$$E = E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\} \tag{5.2}$$
$$e_1 = v_1v_2, \quad e_2 = v_2v_4, \quad \dots \tag{5.3}$$

**Definition 5.1.2** (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 5.1.3** (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.

## 5.2 Graph Isomorphism

## 5.3 The Adjacency and Incidence Matrices

## 5.4 Subgraph

**Definition 5.4.1** (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 5.4.2** (Spanning). A subgraph $H$ on $G$ is **spanning** if $V(H) = V(G)$

**Definition 5.4.3** (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.

## 5.5 Degree

**Definition 5.5.1** (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 5.1.** *For any graph $G = (V, E)$*

$$\sum_{v \in V} d(v) = 2|E| \tag{5.4}$$

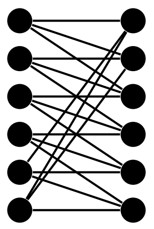
*Proof.* $\forall$ edge $e = uv$ with $u \neq v$, $e$ is counted once for $u$ and once for $v$, a total of two altogether. If $e = uu$, a loop, then it is counted twice for $u$. □

**Problem 5.1.** 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

$$2|E| = \sum_{v \in V} d(v) \geq 3\times|N| = 3\times 13 \Rightarrow |E| \geq 19.5 > 19 \tag{5.5}$$

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

□

**Corollary 5.1.1.** *Every graph has an even number of odd degree vertices.*

*Proof.*

$$V = V_E \cup V_O \Rightarrow \sum_{v \in V} d(v) = \sum_{v \in V_E} d(v) + \sum_{v \in V_O} d(v) = 2|E| \tag{5.6}$$

□

## 5.6 Special Graphs

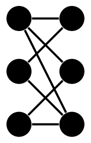
**Definition 5.6.1** (Complete Graph)**.** A **complete** graph $K_n (n \geq 1)$ is a simple graph with $n$ vertices and with exactly one edge between each pair of distinct vertices.

**Definition 5.6.2** (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 5.6.3** (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 $C_n$

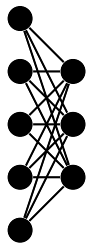
**Definition 5.6.4** (Bipartite Graph)**.** A simple graph is said to be **bipartite** if the vertex set can be expressed as the union of two disjoint non-empty subsets $V_1$ and $V_2$ such that every edges has one end in $V_1$ and another end in $V_2$

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



**Definition 5.6.5** (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 5.2.** *(König Theorem) A graph $G$ is bipartite iff every cycle is even.*

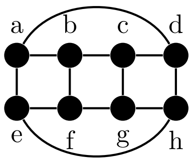
*Proof.* Hereby we prove the ⇒ and ⇐

- (⇒) 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 subgraph that has more that one vertices, we already proved that it has to be bipartite. For the subgraph

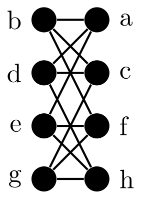
$G_i \subset G, i = 1, 2, ..., n$, the vertices can be partition into $V_{i1} \in V(G_i)$ and $V_{i2} \in V(G_i)$, where $V_{i1} \cap V_{i2} = \emptyset$, the union of those subgraphs are bipartite too because $V_1 = \cup_{i=1}^{n} V_{i1} \in V(G)$ and $V_2 = \cup_{i=1}^{n} V_{i2} \in V(G)$ satisfied 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.

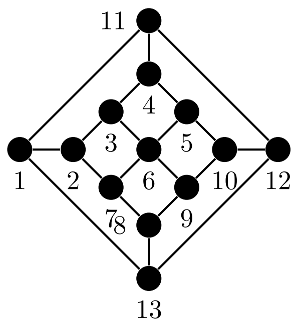


We can rearrange the graph to be more clear as following



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

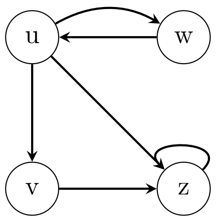
**Example.** The following graph is not bipartite



The cycle $c = v_1 v_1 1 v_4 v_3 v_2$ have odd number of vertices.

## 5.7 Directed Graph

**Definition 5.7.1.** 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 5.7.2.** We call directed graphs **digraphs**, we call edges in a digraph are called **arcs**, and vertices in a digraph **nodes**

**Definition 5.7.3.** Similar as in the undirected case we have walks, traces, paths and cycles in digraphs.

**Definition 5.7.4.** 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 5.7.5.** A digraph is strongly connected if every pair of vertices are strongly connected.

**Definition 5.7.6.** A digraph is **strict** if it has no loops and whenever $e$ and $f$ are parallel, $h(e) = t(f)$

**Definition 5.7.7.** 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

$$n_{ij} = \begin{cases} 1, & \text{if } v_i = h(e_j) \\ -1, & \text{if } v_i = t(e_j) \\ 0, & \text{otherwise} \end{cases} \tag{5.7}$$

$$\begin{bmatrix} -1 & 0 & -1 & -1 & 1 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{5.8}$$

## 5.8 Sperner's Lemma

# Chapter 6

# Paths, Trees, and Cycles

## 6.1 Walk

**Definition 6.1.1** (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.**

$$w = v_2e_4v_3e_4v_2e_5v_3 \tag{6.1}$$

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

**Definition 6.1.2** (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 $w$.

$$w^{-1} = v_ke_kv_{k-1}e_{k-1}...e_2v_1 \tag{6.2}$$

(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.

## 6.2 Path and Cycle

**Definition 6.2.1** (trail)**.** A **trail** is a walk with no repeating edges. e.g., $v_3e_4v_2e_5v_3$

**Definition 6.2.2** (path)**.** A **path** is a trail with no repeating vertices. e.g., $v_3e_4v_2$

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

**Definition 6.2.3** (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 6.2.4** (even/odd cycle)**.** A cycle is **even** if it has a even number of edges otherwise it is **odd**.

**Problem 6.1.** 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

$$C_1 = v_{11}e_{11}v_{12}e_{12}v_{13}e_{13}...v_{1n}e_{1n}v_{11} \tag{6.3}$$
$$C_2 = v_{21}e_{21}v_{22}e_{22}v_{23}e_{23}...v_{2k}e_{2k}v_{21} \tag{6.4}$$

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.

**Algorithm 1** 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
```

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 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 6.2.5** (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 6.2.6** (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 6.2.** 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.

□

## 6.3 Tree and forest

**Definition 6.3.1** (acyclic graph). A graph is called **acyclic** if it has no cycles

**Definition 6.3.2** (forest, tree). A acyclic graph is called a **forest**. A connected forest is called a **tree**.

**Theorem 6.1.** *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.)

**Algorithm 2** 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
```

The above algorithm is guaranteed to have an end because a tree is acyclic by definition

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)

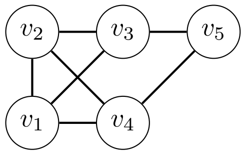
3. Do the leaf removal process recursively to $T^{'}$ if Situation 2 happens until Situation 1 happens.

□

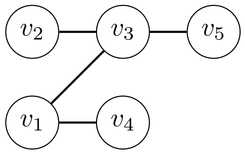
## 6.4 Spanning tree

**Definition 6.4.1** (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 6.3.** 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 6.2.** *Every connected graph has a spanning tree.*

*Proof.* Prove by constructing algorithm: □

**Algorithm 3** 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. □

## 6.5 Cayley's Formula

## 6.6 Connectivity

## 6.7 Blocks

# Chapter 7

# Euler Tours and Hamilton Cycles

## 7.1 Euler Tours

## 7.2 Hamilton Cycles

# Chapter 8

# Planarity

## 8.1 Plane and Planar Graphs

## 8.2 Dual Graphs

## 8.3 Euler's Formula

## 8.4 Bridges

## 8.5 Kuratowski's Theorem

## 8.6 Four-Color Theorem

## 8.7 Graphs on other surfaces

# Chapter 9

# Minimum Spanning Tree Problem

## 9.1 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 9.1.1** (Cut vertex)**.** A vertex $v$ of a connected graph $G$ is a **cut vertex** if $G \setminus v$ is not connected.

**Definition 9.1.2** (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 9.1.** *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 9.1.3** (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.

## 9.2 Kroskal's Algorithm

**Algorithm 4** 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
```

## 9.3 Prim's Algorithm

**Algorithm 5** 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
```

## 9.4 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.

## 9.5 Extensible MST

**Definition 9.5.1** (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 9.5.2.** We also define $\gamma(A) = \{e \in E : \text{both ends of } e \text{ are in } A\}$

**Theorem 9.2.** *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 9.5.3.** 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 9.3.** *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.*

## 9.6 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

$$X_e = \begin{cases} 1, \text{if edge } e \text{ is in the optimal solution} \\ 0, \text{otherwise} \end{cases} \tag{9.1}$$

The formulation is as following

$$\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e && (9.2)\\
\text{s.t.} \quad & \sum_{e \in E} x_e = |V| - 1 && (9.3)\\
& x_e \geq 0 && (9.4)\\
& e \in E && (9.5)\\
& \sum_{e \in E(S)} x_e = |S| - 1, \forall S \subseteq V, S \neq \emptyset && (9.6)\\
& && (9.7)
\end{aligned}$$

# Chapter 10

# Shortest-Path Problem

## 10.1 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 10.1.1** (feasible potential)**.** We call $y = (y_v : v \in V)$ a **feasible potential** if it satisfies

$$y_v + c_{vw} \geq y_w \quad \forall (v, w) \in E \tag{10.1}$$

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

$$C(P) = \sum_{i=1}^{k} C_{e_i} \geq \sum_{i=1}^{k} (y_{v_i} - y_{v_{i-1}}) = y_{v_k} - y_{v_0} \tag{10.2}$$

□

## 10.2 Breadth-First Search Algorithm

## 10.3 Ford's Method

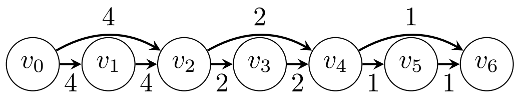
Define a predecessor function $P(w), \forall w \in V$ and set $P(w)$ to $v$ whenever $y_w$ is set to $y_v + c_{vw}$

**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.

**Algorithm 6** 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
```



## 10.4 Ford-Bellman Algorithm

**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 $v_i v_j$ precedes $v_k v_i$ if $i < k$ based on topological order then ford algorithm will terminate in one pass.

**Algorithm 7** 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
```

## 10.5 SPFA Algorithm

## 10.6 Dijkstra Algorithm

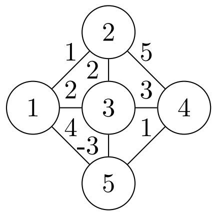
**Algorithm 8** Dijkstra Algorithm

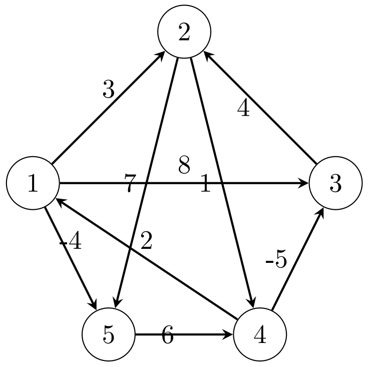
```
Ensure: Shortest Paths from $r$ to all other nodes in $V$
Require: A digraph with arc costs,starting node $r$
  Initialize $y$ and $p$
  $S \leftarrow V$
  while $S \neq \emptyset$ do
    Choose $v \in S$ with minimum $y_v$
    $S \leftarrow S \setminus v$
    for $\forall w, (v, w) \in E$ do
      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
```

## 10.7 A* Algorithm

## 10.8 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 $O(n^3)$ 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

$$d_{ij}^0 = \begin{cases} c_{ij} & \text{distance from } i \text{ to } j \text{ if } (i,j) \in E \\ 0 & \text{if } i = j \\ \infty & \text{if } (i,j) \notin E \end{cases} \tag{10.3}$$

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 $D^n$

$$D^0 = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \tag{10.4}$$

$$\Pi^0 = \begin{bmatrix} & 1 & 1 & & 1 \\ & & & 2 & 2 \\ & 3 & & & \\ 4 & & 4 & & \\ & & & 5 & \end{bmatrix} \tag{10.5}$$

$$D^1 = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \mathbf{5} & -5 & 0 & \mathbf{-2} \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \tag{10.6}$$

$$\Pi^1 = \begin{bmatrix} & 1 & 1 & & 1 \\ & & & 2 & 2 \\ & 3 & & & \\ 4 & \mathbf{1} & 4 & & \mathbf{1} \\ & & & 5 & \end{bmatrix} \tag{10.7}$$

$$D^2 = \begin{bmatrix} 0 & 3 & 8 & \mathbf{4} & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \mathbf{5} & \mathbf{11} \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \tag{10.8}$$

$$\Pi^2 = \begin{bmatrix} & 1 & 1 & \mathbf{2} & 1 \\ & & & 2 & 2 \\ & 3 & & \mathbf{2} & \mathbf{2} \\ 4 & 1 & 4 & & 1 \\ & & & 5 & \end{bmatrix} \tag{10.9}$$

$$D^3 = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & \mathbf{-1} & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \tag{10.10}$$

$$\Pi^3 = \begin{bmatrix} & 1 & 1 & 2 & 1 \\ & & & 2 & 2 \\ & 3 & & 2 & 2 \\ 4 & \mathbf{3} & 4 & & 1 \\ & & & 5 & \end{bmatrix} \tag{10.11}$$

$$D^4 = \begin{bmatrix} 0 & 3 & \mathbf{-1} & 4 & -4 \\ \mathbf{3} & 0 & \mathbf{-4} & 1 & \mathbf{-1} \\ \mathbf{7} & 4 & 0 & 5 & \mathbf{3} \\ 2 & -1 & -5 & 0 & -2 \\ \mathbf{8} & \mathbf{5} & \mathbf{1} & 6 & 0 \end{bmatrix} \tag{10.12}$$

$$\Pi^4 = \begin{bmatrix} & 1 & \mathbf{4} & 2 & 1 \\ \mathbf{4} & & \mathbf{4} & 2 & \mathbf{1} \\ \mathbf{4} & 3 & & 2 & \mathbf{1} \\ 4 & 3 & 4 & & 1 \\ \mathbf{4} & \mathbf{3} & \mathbf{4} & 5 & \end{bmatrix} \tag{10.13}$$

$$D^5 = \begin{bmatrix} 0 & \mathbf{1} & \mathbf{-3} & \mathbf{2} & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix} \tag{10.14}$$

$$\Pi^5 = \begin{bmatrix} & \mathbf{3} & 4 & \mathbf{5} & 1 \\ 4 & & 4 & 2 & 1 \\ 4 & 3 & & 2 & 1 \\ 4 & 3 & 4 & & 1 \\ 4 & 3 & 4 & 5 & \end{bmatrix} \tag{10.15}$$

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.

## 10.9 Johnson's Algorithm

# Chapter 11

# Maximum Flow Problem

## 11.1 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.

$$\max \quad v \tag{11.1}$$

$$\text{s.t.} \quad \sum_{h(e)=i} x_e - \sum_{t(e)=i} x_e = \begin{cases} -v, & \text{if } i = s \\ v, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases} \tag{11.2}$$

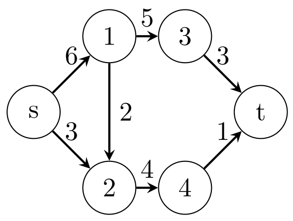
$$0 \le x_e \le u_e, \quad \forall e \in A \tag{11.3}$$

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.

## 11.2 Solving Maximum Flow Problem in LP

**Theorem 11.1.** *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 $\delta^-(S) = \{e \in A : t(e) \in S \text{ and } h(e) \in \bar{S}\}$*

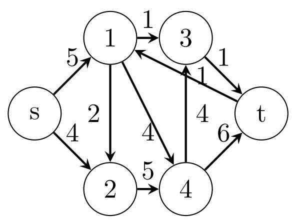
**Example.** For the following graph:



Let $S = \{1, 2, 3, s\}$, $\bar{S} = \{4, t\}$
then $\delta^-(S) = \{(2,4), (3,t)\} \Rightarrow u(S, \bar{S}) = 7$

**Definition 11.2.1.** If $(S, \bar{S})$ has minimum capacity of all $(s,t)$-cuts, then it is called **minimum cut**.

**Definition 11.2.2.** 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 11.2.** *If $x$ is a $(s,t)$ flow of value $v$ and $(S, \bar{S})$ is a $(s,t)$-cut, then*

$$v = \sum_{e \in \delta^-(S)} x_e - \sum_{e \in \delta^+(S)} x_e \tag{11.4}$$

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

$$\sum_{i \in S} \left( \sum_{h(e)=i} x_e - \sum_{t(e)=i} x_e \right) = -v \tag{11.5}$$

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

$$\sum_{e \in \delta^+(S)} x_e - \sum_{e \in \delta^-(S)} x_e = -v \tag{11.6}$$

□

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

**Corollary 11.2.1.** *If $x$ is a feasible flow of value $v$, and $(S, \bar{S})$ is an $(s,t)$-cut, then*

$$v \le u(S, \bar{S}) \quad \textit{(Weak duality)} \tag{11.7}$$

**Definition 11.2.3.** Define an arc $e$ to be **saturated** if $x_e = u_e$, and to be **flowless** if $x_e = 0$

**Corollary 11.2.2.** *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

$$\sum_{e\in\delta^-(S)} x_e = \sum_{e\in\delta^-(S)} u_e \tag{11.8}$$

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

$$\sum_{e\in\delta^+(S)} x_e = 0 \tag{11.9}$$

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

## 11.3 Prime and Dual of Maximum Network Flow Problem

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

$$\max \quad f = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ f \end{bmatrix} \tag{11.10}$$

$$\text{s.t.} \quad \begin{bmatrix} \mathbf{A} & \mathbf{F} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ f \end{bmatrix} = \mathbf{0} \tag{11.11}$$

$$\mathbf{Ix} \le \mathbf{u} \tag{11.12}$$

$$\begin{bmatrix} \mathbf{x} \\ f \end{bmatrix} \ge 0 \tag{11.13}$$

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.

$$\mathbf{A} = \mathbf{A}_{|E|\times|V|} = [a_{ij}], \text{ where } a_{ij} = \begin{cases} 1, & \text{if } v_i = h(e_j) \\ -1, & \text{if } v_i = t(e_j) \\ 0, & \text{otherwise} \end{cases} \tag{11.14}$$

$$\mathbf{F} = \begin{bmatrix} -1 & \cdots & 0 & \cdots & 1 \end{bmatrix}^\top \tag{11.15}$$

$$\mathbf{u} = \begin{bmatrix} u_1 & u_2 & \cdots & u_{|E|} \end{bmatrix}^\top \tag{11.16}$$

Then, we take the dual of LP

$$\min \quad \mathbf{uw_E} \tag{11.17}$$

$$\text{s.t.} \quad \begin{bmatrix} \mathbf{w_V} & \mathbf{w_E} \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ \mathbf{I} \end{bmatrix} \ge 0 \tag{11.18}$$

$$\begin{bmatrix} \mathbf{w_V} & \mathbf{w_E} \end{bmatrix} \begin{bmatrix} \mathbf{F} \\ \mathbf{0} \end{bmatrix} = 1 \tag{11.19}$$

$$\mathbf{w_V} \quad \text{unrestricted} \tag{11.20}$$

$$\mathbf{w_E} \ge \mathbf{0} \tag{11.21}$$

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.

$$\mathbf{w_V} = \begin{bmatrix} w_1 & w_2 & \cdots & w_{|V|} \end{bmatrix}^\top \tag{11.22}$$

$$\mathbf{w_E} = \begin{bmatrix} w_{|V|+1} & w_{|V|+2} & \cdots & w_{|V|+|E|} \end{bmatrix}^\top \tag{11.23}$$

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

$$\min \quad \sum_{e\in E} u_e w_e \tag{11.24}$$

$$\text{s.t.} \quad w_i - w_j + w_{|V|+e} \ge 0, \forall e = (i,j) \in E \tag{11.25}$$

$$-w_1 + w_{|V|} = 1 \tag{11.26}$$

$$\mathbf{w_V} \quad \text{unrestricted} \tag{11.27}$$

$$\mathbf{w_E} \ge \mathbf{0} \tag{11.28}$$

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.

## 11.4 Maximum Flow Minimum Cut Theorem

**Definition 11.4.1.** 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 11.3.** *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):

$$D_1 = \min\{u_e - x_e : e \text{ forward in } P\} \tag{11.29}$$

$$D_2 = \min\{x_e : e \text{ backward in } P\} \tag{11.30}$$

$$D = \min\{D_1, D_2\} \tag{11.31}$$

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

$$\hat{x_e} = \begin{cases} x_e + D & \text{If } e \text{ is forward in } P \\ x_e - D & \text{If } e \text{ is backward in } P \\ x_e & otherwise \end{cases} \tag{11.32}$$

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 11.4.** *(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*

## 11.5 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.

FIXME This algorithm is incomplete/incorrect, needs to be fixed

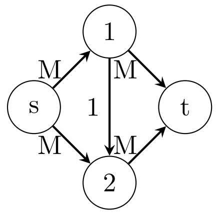
**Algorithm 9** Labeling algorithm

```
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.
```

Labeling algorithm can be exponential, the following is an example

**Algorithm 10** 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.
```



## 11.6 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

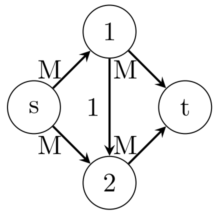
Layer algorithm (Dinic 1977) Input: A network $(D,s,t,u)$ and a feasible flow $x$ Output: The **x-layers** $V_0, V_1, ..., V_l$ where $V_i \cap V_j = \emptyset \forall i \neq j$

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 $V_{i+1} \leftarrow R \setminus \cup_{0 \le j \le i} V_j$, $l \leftarrow i + 1, l(x) = i$, goto Step 2. If $V_{i+1} = \emptyset$, set $l(x) = i$ and Stop.

**Example.** For the following graph



$$\text{Second iteration} \tag{11.33}$$
$$V_0 = \{s\}, i = 0, l(x) = 0 \tag{11.34}$$
$$R = \{1, 2\} \tag{11.35}$$
$$V_1 \leftarrow \{1, 2\}, i = 1, l(x) = 1 \tag{11.36}$$
$$R = \{3, 4, 5\} \tag{11.37}$$
$$V_2 \leftarrow \{3, 4\}, i = 2, l(x) = 2 \tag{11.38}$$
$$R = \{1, 5, 6, 3\} \tag{11.39}$$
$$V_3 \leftarrow \{5, 6\}, i = 3, l(x) = 3 \tag{11.40}$$
$$R = \{4, t\} \tag{11.41}$$
$$V_4 = \{t\} \tag{11.42}$$
$$A_1 = \{(s, 1), (s, 2)\} \tag{11.43}$$
$$A_2 = \{(1, 3), (2, 4)\} \tag{11.44}$$
$$A_3 = \{(3, 5), (4, 6)\} \tag{11.45}$$
$$A_4 = \{(5, t), (6, t)\} \tag{11.46}$$

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

$$x'_e = x_e + \hat{x_e}, \quad \text{If } t(e) \in V_{i-1}, h(e) \in V_i \tag{11.47}$$
$$x'_e = x_e - \hat{x_e}, \quad \text{If } h(e) \in V_{i-1}, t(e) \in V_i \tag{11.48}$$

## 11.7 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.

**Theorem 11.5.** *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 12

# Minimum Weight Flow Problem

## 12.1 Transshipment Problem

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

$$\begin{aligned} \min \quad & wx && (12.1)\\ \text{s.t.} \quad & Nx = b && (12.2)\\ & x \geq 0 && (12.3) \end{aligned}$$

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

$$\sum_{h(e)=i} x_e - \sum_{t(e)=i} x_e = b_i \qquad (12.4)$$

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.

## 12.2 Network Simplex Method

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

**Lemma 12.1.** *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. □

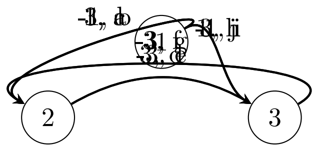
**Theorem 12.2.** *Let $T$ be a spanning tree of $G$. And let $e \in E \setminus T$ then $T + e$ contains a unique cycle $C$ and for any edge $f \in C$, $T + e - f$ is a spanning tree of $G$*

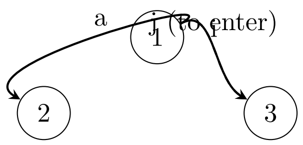
Let $(D, b, w)$ be a transshipment problem. A feasible solutions $x$ is a **feasible tree solution** if there is a spanning tree $T$ such that $||x|| = \{e \in A, x_e \neq 0\} \subseteq T$.
For any tree $T$ of $D$ and for $e \in A \setminus T$, it follows from above theorem that $T+e$ contains a unique cycle. Denote that cycle $C(T, e)$ and orient it in the direction of $e$, define $w(T, e) = \sum\{w_e : e \text{ forward in } C(T, e)\} - \sum\{w_e : e \text{ reverse in } C(T, e)\}$.
We think of $w(T, e)$ as the weight of $C(T, e)$.

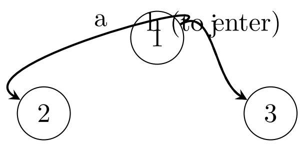
**Algorithm 11** Network Simplex Method Algorithm

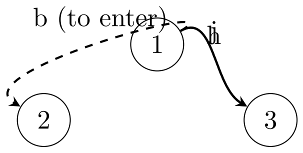
```
Ensure: An optimal solution or the conclusion that
    $(D, b, w)$ is unbounded
Require: A transshipment problem $(D, b, w)$ and a fea-
    sible tree solution $x$ containing to a spanning tree $T$

  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
```

The following is an example of cycling
Then for
$w(T, j) = w_j - w_i = -3 - 3 = -6$
$w(T, h) = w_h + w_j - w_i = 3 - 3 - 1 = -1$
Keep going and we will find cycling.
To Avoid cycling we will introduce the modified network Simplex Method. Let $T$ be a **rooted** spanning tree. Let









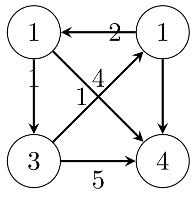
$f$ be an arc in $T$, we say $f$ is **away** from the root $r$ if $t(f)$ is the component of $T - f$. Otherwise we say $f$ is **towards** $r$.
Let $x$ be a feasible tree solution associated with $T$, then we say $T$ is a **strong feasible tree** if for every arc $f \in T$ with $x_f = 0$ then $f$ is away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

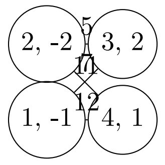
- The algorithm is initialed with a strong feasible tree.
- $f$ in pivot phase is chosen to be the first reverse arc of $C(T, e)$ having $x_f = \theta$. By “first”, we mean the first arc encountered in traversing $C(T, e)$ in the direction of $e$, starting at the vertex $i$ of $C(T, e)$ that minimizes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

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.

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





## 12.3 Out-of-Kilter algorithm

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

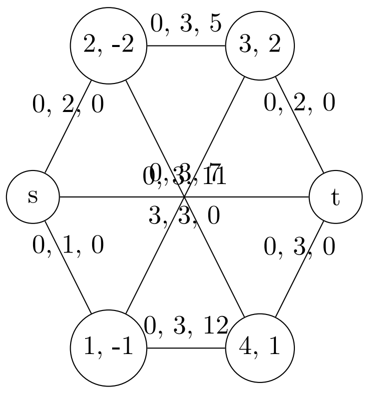
**Definition 12.3.1.** The minimum weight circulation problem is defined as follows:

$$\min \quad wx \tag{12.5}$$
$$\text{s.t.} \quad Nx = 0 \tag{12.6}$$
$$l \leq x \leq u \tag{12.7}$$

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

We will show how to transform any transshipment into circulation. (The reverse is also possible.)
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 $w_{ts} = 0, l_{ts} = u_{ts} = \sum\{b_x : \forall x, xisdemandvertex\}$. Each original arc is given a $l_x = 0, u_x = \sum\{b_x : \forall x, xisademandvertex\}$, $w_x$ remains unchanged.

**Theorem 12.3.** *(For LP optimality conditions we need primal feasibility, dual feasibility and complementary slackness. Primal and dual feasibility are obvious so we need to show complementary slackness through following theorem.)*



*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 $e \in A$*

$$y_{h(e)} - y_{t(e)} > w_e \text{ implies } x_e = u_e \quad (12.8)$$
$$y_{h(e)} - y_{t(e)} < w_e \text{ implies } x_e = l_e \quad (12.9)$$

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

*Proof.* For each $e \in A$ define $\gamma_e = \max\{y_{h(e)} - y_{t(e)} - w_e, 0\}$, $\mu_e = \max\{w_e - y_{h(e)} + y_{t(e)}, 0\}$, $\gamma_e - \mu_e = y_{h(e)} - y_{t(e)} - w_e$.

$$\sum_{e\in A}(\mu_e l_e - \gamma_e u_e) = \sum_{e\in A}(\mu_e l_e - \gamma_e u_e) + \sum_{i\in V} y_i(\sum_{h(e)=i} x_e - \sum_{t(e)=i} x_e) \quad (12.10)$$
$$= \sum_{e\in A}(\mu_e l_e - \gamma_e u_e + x_e(y_{h(e)} - y_{t(e)})) \quad (12.11)$$
$$= \sum_{e\in A}(\mu_e l_e - \gamma_e u_e + x_e(\gamma_e - \mu_e + w_e)) \quad (12.12)$$
$$= \sum_{e\in A}(\gamma_e(x_e - u_e) + \mu_e(l_e - x_e) + x_e w_e) \quad (12.13)$$
$$\le \sum_{e\in A} x_e w_e \quad (12.14)$$

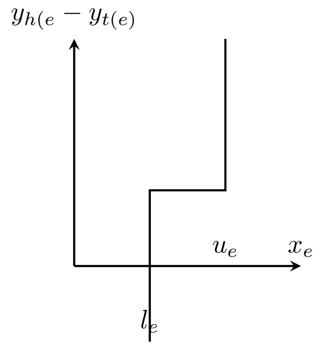
The last inequality will be satisfied as equality iff the first two hold. □

The following is the formulation of circulation problem

$$\begin{array}{llll}
\text{(P)} & \min & wx & (12.15)\\
& \text{s.t.} & Nx = 0 \quad y & (12.16)\\
& & x \ge l \quad z^l & (12.17)\\
& & -x \le -u \quad z^u & (12.18)\\
\text{(D)} & \max & lz^l - uz^u & (12.19)\\
& \text{(s.t.)} & yN^{-1} + z^l - z^u \le w & (12.20)\\
& & y \quad free & (12.21)\\
& & z^l, z^u \ge 0 & (12.22)\\
& \text{(CS)} & y_{h(e)} - y_{t(e)} > w_e \Rightarrow x_e = u_e & (12.23)\\
& & y_{h(e)} - y_{t(e)} < w_e \Rightarrow x_e = l_e & (12.24)
\end{array}$$

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 12.4.** *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 $\forall e \in A, l_e \le x_e \le u_e$. 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
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 $\Delta_1 = \min\{u_e - x_e : eisforwardandy_e \geq w_e\}$ Set $\Delta_2 = \min\{l_e - x_e : eisforwardandy_e < w_e\}$ Set $\Delta_3 = \min\{x_e - l_e : eisrevereseandy_e \leq w_e\}$ Set $\Delta_4 = \min\{x_e - u_e : eisrevereseandy_e > w_e\}$ $\Delta = \min\{\Delta_i, i = 1,2,3,4\}$
Increase $x_e$ by $\Delta$ on each forword 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 veertices 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 $R = \lceil \max\{\log_2 u_e : e \in A\} \rceil$

# Chapter 13

# Matchings

## 13.1 Hall's "Marriage" Theorem

## 13.2 Transversal Theory

## 13.3 Menger's Theorem

## 13.4 The Hungarian Algorithm

# Chapter 14

# Colorings

## 14.1 Edge Chromatic Number

## 14.2 Vizing's Theorem

## 14.3 The Timetabling Problem

## 14.4 Vertex Chromatic Number

## 14.5 Brooks' Theorem

## 14.6 Hajós' Theorem

## 14.7 Chromatic Polynomials

## 14.8 Girth and Chromatic Number

# Chapter 15

# Independent Sets and Cliques

## 15.1 Independent Sets

## 15.2 Ramsey's Theorem

## 15.3 Turán's Theorem

## 15.4 Schur's Theorem

# Chapter 16

# Matroids

# Part III

# Nonlinear Programming

# Chapter 17

# Convex Analysis

## 17.1 Convex Sets

**Definition 17.1.1.** 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
$$S_1 \oplus S_2 = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{x} = \mathbf{x_1} + \mathbf{x_2}, \mathbf{x_1} \in S_1, \mathbf{x_2} \in S_2\} \tag{17.1}$$
- $S_1 \ominus S_2$ is convex set, where
$$S_1 \oplus S_2 = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{x} = \mathbf{x_1} - \mathbf{x_2}, \mathbf{x_1} \in S_1, \mathbf{x_2} \in S_2\} \tag{17.2}$$
- $f(S_1)$ 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 17.1** (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 17.2.** *Let $S$ be a convex set with nonempty interior. Let $\mathbf{x}_1 \in cl(S)$ and $\mathbf{x}_2 \in int(S)$, then $\mathbf{y} = \lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \in int(S), \forall \lambda \in (0,1)$*

## 17.2 Convex Functions

**Definition 17.2.1.** Let $C \in \mathbb{R}^n$ be a convex set. A function $f : C \to \mathbf{R}$ is (resp. strictly) convex if

$$f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda)f(\mathbf{x}_2) \tag{17.3}$$
$$\forall \mathbf{x}_1, \mathbf{x}_2 \in C, \forall \lambda \in (0,1) \tag{17.4}$$

(resp.)

$$f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) < \lambda f(\mathbf{x}_1) + (1-\lambda)f(\mathbf{x}_2) \tag{17.5}$$
$$\forall \mathbf{x_1} \neq \mathbf{x}_2 \in C, \forall \lambda \in (0,1) \tag{17.6}$$

**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 17.2.2.** 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 17.3.** *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 17.2.3** (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 17.4.** *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 17.2.4** (Epigraphs, Hypographs)**.** Let $S \in \mathbb{R}^n$ be such that $S \neq \emptyset$. The **epigraph** of $f$, denoted by $epi(f)$ is

$$epi(f) = \{(\mathbf{x}, y) \in S | \mathbf{x} \in S, y \in \mathbb{R}, y \geq f(x)\} \in \mathbb{R}^{n+1} \tag{17.7}$$

The **hypograph** of $f$, denoted by $hypo(f)$ is

$$hypo(f) = \{(\mathbf{x}, y) \in S | \mathbf{x} \in S, y \in \mathbb{R}, y \leq f(x)\} \in \mathbb{R}^{n+1} \tag{17.8}$$

**Theorem 17.5.** *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 17.6.** *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 $int(S)$.*

## 17.3 Subgradients and Subdifferentials

**Definition 17.3.1** (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

$$f(\mathbf{x}) \geq f(\bar{\mathbf{x}}) + \xi^\top(\mathbf{x} - \bar{\mathbf{x}}), \forall \mathbf{x} \in S \tag{17.9}$$

**Definition 17.3.2** (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 17.7.** *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*

$$f(\mathbf{x}) \geq f(\bar{\mathbf{x}}) + \xi^\top(\mathbf{x} - \bar{\mathbf{x}}), \forall \mathbf{x} \in S \tag{17.10}$$

*In particular, the hyperplane*

$$\mathcal{H} = \{(\mathbf{x}, y) | y = f(\bar{\mathbf{x}}) + \xi^\top(\mathbf{x} - \bar{\mathbf{x}})\} \tag{17.11}$$

*is a supporting plane of $epi(f)$ at $(\bar{\mathbf{x}}, f(\bar{\mathbf{x}}))$*

**Theorem 17.8.** *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*

$$f(\mathbf{x}) \geq f(\bar{\mathbf{x}}) + \xi^\top(\mathbf{x} - \bar{\mathbf{x}}), \forall \mathbf{x} \in S \tag{17.12}$$

*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.

## 17.4 Differentiable Functions

**Definition 17.4.1** (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

$$f(\mathbf{x}) = f(\bar{\mathbf{x}}) + \nabla f(\bar{\mathbf{x}})^\top(\mathbf{x} - \bar{\mathbf{x}}) + \alpha(\bar{\mathbf{x}}, \mathbf{x} - \bar{\mathbf{x}})\|\mathbf{x} - \bar{\mathbf{x}}\| \tag{17.13}$$

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 17.9.** *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 17.10.** *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 $\forall \bar{\mathbf{x}} \in S$*

$$f(\mathbf{x}) \geq f(\bar{\mathbf{x}}) + \nabla f(\bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}}), \forall \mathbf{x} \in S \tag{17.14}$$

*(resp.)*

$$f(\mathbf{x}) > f(\bar{\mathbf{x}}) + \nabla f(\bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}}), \forall \mathbf{x} \neq \bar{\mathbf{x}} \in S \tag{17.15}$$

**Theorem 17.11** (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*

$$f(\mathbf{x}_2) = f(\mathbf{x}_2) + \nabla f(\hat{\mathbf{x}})(\mathbf{x}_2 - \mathbf{x}_1) \tag{17.16}$$

*where*

$$\hat{\mathbf{x}} = \lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2 \tag{17.17}$$

# Chapter 18

# KKT Optimality Conditions

# Chapter 19

# Lagrangian Duality

# Chapter 20

# Unconstrained Optimization

# Chapter 21

# Penalty and Barrier Functions