1. Basic Concepts in Graph Theory

(1.1) A graph $G$ consists of a nonempty set $V(G)$ of elements called vertices, and a set $E(G)$ of elements called edges, and a relation of incidence that associates with each edge $e$ two vertices $u$ and $v$, called the ends of $e$. Sometimes a graph $G$ with vertex set $V = V(G)$ and edge set $E = E(G)$ is denoted $G(V, E)$, to indicate the vertex and edge sets.

(1.2) If an edge $e$ has ends $u, v$, then $e$ is incident with $u$ and $v$, and $u$ and $v$ are adjacent to each other; and $(u, e)$ and $(v, e)$ are two incidences of $G$. An edge with identical ends is a loop, and one with distinct ends is a link. Two or more edges with the same pair of vertices as their common ends are multiple edges. A graph $G$ is simple if $G$ does not have loops nor multiple edges.

(1.3A) A simple graph is complete if any two of its vertices are joined by an edge. A
complete graph on \( n \) vertices is denoted \( K_n \).

Draw \( K_n \) for \( 1 \leq n \leq 5 \).

(1.3B) Let \( S \) be an \( n \)-set. The **Kneser graph** \( KN_{m,n} \) is the graph whose vertex set is the set of all \( m \)-subsets of \( S \), two vertices being adjacent if they are disjoint. The \( KN_{2,5} \) is called the **Petersen graph**.

Draw the Kneser graphs \( KN_{m,n} \), \( 1 \leq m \leq 3 \) and \( 3 \leq n \leq 5 \).

(1.4) Let \( G \) and \( H \) be graphs. A **homomorphism** from \( G \) to \( H \) is a pair \((\theta, \phi)\) of maps \( \theta : V(G) \to V(H) \) and \( \phi : E(G) \to E(H) \) such that \((v, e)\) is an incidence of \( G \) iff \((\theta(v), \phi(e))\) is an incidence of \( H \). If both \( \theta \) and \( \phi \) are bijections, then \((\theta, \phi)\) is an **isomorphism**.

(1.5) Suppose \( G \) and \( H \) are graphs. If \( V(H) \subseteq V(G) \), \( E(H) \subseteq E(G) \) and for each edge \( e \in E(H) \), the ends of \( e \) in \( H \) are the same ends of \( e \) in \( G \), then \( H \) is a **subgraph** of \( G \), and \( G \) a **supergraph** of \( H \). This is denoted by \( H \subseteq G \). If \( V(G) = V(H) \) and \( H \subseteq G \), then \( H \) is a **spanning** subgraph of \( G \).

(1.5A) If \( G = G(V, E) \) and if \( S \subseteq E \), then \( G'(V, E') \) with \( E' = E - S \) is a subgraph of \( G \), denoted by \( G - S \).

(1.5B) If \( G = G(V, E) \) and if \( R \subseteq V \), then \( G'(V', E') \) with \( V' = V - R \), and \( E' \subseteq E \) being the set of edges whose ends are in \( V' \) is a subgraph of \( G \), denoted by \( G - R \).

(1.5C) Let \( G = G(V, E) \) be a graph, and \( S \subseteq E(G) \) be an edge subset. The subgraph \( H = H(V', S) \) of \( G \) with \( V' \) be the set of vertices in \( G \) that are incident with an edge in \( S \) is called the subgraph **induced** by the edge set \( S \), and will be denoted by \( G[S] \).

(1.5D) Let \( G = G(V, E) \) be a graph, and \( R \subseteq V(G) \) be a vertex subset. The subgraph \( H = H(R, E') \) of \( G \) with \( E' \) be the set of edges in \( G \) whose ends are in \( R \) is called the subgraph **induced** by the vertex set \( R \), and will be denoted by \( G[R] \).

(1.6) A graph \( G \) is **bipartite** if \( V(G) \) can be partitioned into nonempty subsets \( A \) and \( B \) such that \( A \cap B = \emptyset \) and such that each edge \( e \in E(G) \) has one end in \( A \) and the other in \( B \). This partition \((A, B)\) is called a **bipartition** of \( G \), and we sometimes denote the bipartite graph \( G \) by \( G(A; B) \) to indicate the bipartition. A bipartite graph \( G(A; B) \) is **complete** if every vertex in \( A \) is adjacent to every vertex in \( B \). If \( G(A; B) \) is complete and if \(|A| = n \) and \(|B| = m \), then \( G(A; B) \) is denoted by \( K_{n,m} \).

(1.6A) Draw the complete bipartite graphs \( K_{n,m} \) for \( 1 \leq m \leq 3 \) and \( 2 \leq n \leq 4 \).
(1.7) The \textit{n-cube} $Q_n$ is the simple graph whose vertex set is the set of all $n$-tuples of zeros and ones, two vertices being adjacent in $Q_n$ if they differ in exactly one coordinate.

(1.7A) Draw $Q_n$, $1 \leq n \leq 4$.

(1.8) Find a homomorphism from $Q_3$ to $K_2$. Also show that $K_{4,4}$ contains a spanning subgraph isomorphic to $Q_3$.

(1.9) How many vertices and edges are there in $K_n$? in $K_{n,m}$? in $Q_n$?

(1.10) The \textit{degree} of a vertex $v$ in a graph $G$ denoted by $d_G(v)$, is the number of edges in $G$ that are incident with $v$, where a loop counts as two edges. The maximum and the minimum degree of the vertices of a graph $G$ are denoted by $\Delta(G)$ and $\delta(G)$, respectively.

(1.11) Show that $Q_n$ is regular and bipartite.

(1.12) Is $KN_{m,n}$ regular? is it bipartite?

(1.13) (The \textbf{handshaking lemma}: in any party, the total number of hands shaken is even.) Show that, in any graph $G$,

$$\sum_{v \in V(G)} d_G(v) = 2|E(G)|.$$

(1.14) Show that if $G(A;B)$ is a regular bipartite graph, then $|A| = |B|$.

(1.15) Let $x$ and $y$ be vertices of a graph $G$. An \textit{(x, y)-path} (or just called a path, when the ends of the path are not emphasized) in $G$ is a sequence

$$P = \{v_0, e_1, v_1, e_2, v_2, \ldots, v_{k-1}, e_k, v_k\},$$

where $x = v_0, v_1, \ldots, v_k = y$ are in $V(G)$, $e_1, \ldots, e_k$ are in $E(G)$, and $v_{i-1}$ and $v_i$ are the ends of $e_i$, $1 \leq i \leq k$. The vertices $v_1, \ldots, v_{k-1}$ are \textbf{internal vertices} of $P$. The \textbf{length} of $W$ is the number of edges in $P$, namely, $|E(P)| = k$. We allow $k = 0$ and so $v_0$ itself is a walk of length zero.

A path $P$ is an \textbf{edge simple path} if the edges $e_1, \ldots, e_k$ are all distinct, and a \textbf{simple path} if all vertices $v_0, \ldots, v_k$ are distinct. An edge simple path is also called a \textbf{trail}. We then define \textit{(x, y)-trails} and \textbf{simple (x, y)-paths} similarly. A trail with $v_0 = v_k$ is a \textbf{closed trail}. A closed trail with all vertices $v_0 = v_k, v_1, \ldots, v_{k-1}$ being distinct is a \textbf{circuit}.

(1.15A) Find a path of length 16, a path of length 9, and a circuit of length 8 in $KN_{2,5}$.

(1.15B) A graph that itself is a path will be called a \textbf{path} and a graph that itself is a circuit will be called a \textbf{circuit}. We shall use $P_n$ and $C_n$ to denote paths and circuits with
(1.16) Suppose \(x, y \in V(G)\) are distinct vertices. Show that \(G\) has an \((x, y)\)-trail if and only if \(G\) has a simple \((x, y)\)-path.

(1.17) Let \(G\) be a graph. Define a relation on \(V(G)\) by \(x\) is related to \(y\) if \(G\) has an \((x, y)\)-path, \(\forall x, y \in V(G)\). Show that this is an equivalence relation on \(V(G)\).

(1.17A) A graph \(G\) is connected if \(V(G)\) has only one equivalence class. Show that \(G\) is connected if and only if for every pair of vertices \(x, y \in V(G)\), \(G\) has an \((x, y)\)-path.

(1.17B) A subgraph \(H\) of \(G\) is a (connected) component of \(G\) if \(H\) is induced by an equivalence class. Show that \(H\) is a component if and only if \(H\) is a maximal connected subgraph of \(G\) (that is, a connected subgraph of \(G\) which is not contained in other connected subgraphs of \(G\)).

(1.18) A graph \(G\) that does not contain a circuit is called a forest, and a connected forest is a tree.

(1.18A) Draw all trees with at most 5 vertices.

(1.18B) Show that if \(F\) is a forest with \(n\) vertices and \(m\) components, then \(F\) has exactly \(n - m + 1\) edges.

(1.18C) Show that for a graph \(G\), the following are equivalent:

(i) \(G\) is a tree.

(ii) \(G\) is a connected graph but for any edge \(e \in E(G)\), \(G - e\) is not connected.

(iii) For any pair of distinct vertices \(u\) and \(v\), \(G\) has a unique \((u, v)\)-path.

(iv) \(G\) is connected and \(|E(G)| = |V(G)| - 1\).

(iv) \(G\) can be built up from any of its vertices by consecutively adjoining edges so that one end of the currently added edge belongs to the graph having already been constructed while the other endpoint does not.

(1.19) Let \(x, y\) be two vertices of a graph \(G\). The distance in \(G\) between \(x\) and \(y\), denoted by \(\text{dist}_{G}(x, y)\), is the length of a shortest \((x, y)\)-path in \(G\); and \(\text{dist}_{G}(x, y) = \infty\) if \(G\) has no \((x, y)\)-paths. The diameter of \(G\) is the maximum distance between vertices of \(G\).

(1.19A) Show that the diameter of \(K_{N_{2,5}}\) is 2.

(1.19B) Find the diameter and the length of a longest path in \(G\), where \(G\) is

(i) \(K_{n}\);

(ii) \(K_{m,n}\);

(iii) \(K_{N_{2,5}}\);

(iv) \(K_{N_{3,7}}\);

(v) \(Q_{n}\).
Let $n \geq 3$ and $d \geq 1$ be integers. Construct

(i) a simple, connected, 3-regular, and bipartite graph $G$ on $2n$ vertices for each $n \geq 3$;
(ii) a simple, 2-regular graph $G$ of diameter $d$, for each $d \geq 1$;
(iii) a simple, 3-regular graph $G$ of diameter $d$, for each $d \geq 1$.

The **girth** of a graph $G$, denoted by $g(G)$, is the length of a shortest circuit of $G$. If $G$ is a forest, then $g(G) = 0$.

Find the girth of $G$, where $G$ is

(i) $K_n$;
(ii) $K_{m,n}$;
(iii) $KN_2,5$;
(iv) $KN_3,7$;
(v) $Q_n$.

For each $g \geq 1$, construct

(i) a 2-regular graph $G$ of girth $g$;
(ii) a 3-regular graph $G$ of girth $g$.

A **digraph** (directed graph) $D$ is obtained from a graph $G$ by assigning each edge $e \in E(G)$ a direction. We call $D$ an **orientation** of $G$. If $e$ has $u$ and $v$ as ends and if $e$ is oriented from $u$ (tail) to $v$ (head), then we denote the oriented $e$ by $(u, v)$. The oriented edges in a digraph $D$ are called **arcs**. We sometimes denote a directed graph $D$ by $D(V, E)$ to indicate the vertex set $V$ and the arc set $E$ of $D$. **Loops** and **links** in a digraph are defined similar to (1.2). But two or more arcs with the same heads and the same tails are **multiple arcs**. A digraph $D$ is **strict** if it has no loops nor multiple arcs.

The **directed circulant** $DC_{m,n}$ is the strict digraph whose vertex set is the set of residues modulo $n$, two vertices $i$ and $j$ being joined by an arc $(i, j)$ if $1 \leq j - i \leq m \,(\text{mod} \, n)$. Draw $DC_{m,n}$, $1 \leq m \leq 3$ and $3 \leq n \leq 6$.

Let $G$ be a graph. The **associate digraph** $D(G)$ of $G$ is the graph obtained from $G$ by replacing each edge of $G$ by two oppositely oriented arcs with the same ends. Note that $D(G)$ is **not** an orientation of $G$.

The **complete digraph** with $n$ vertices is $D(K_n)$. A **tournament** with $n$ vertices is an orientation of $K_n$. Draw $D(K_n)$, $1 \leq n \leq 4$. Also find two distinct orientations of $K_3$ (that is, two distinct tournaments with 3 vertices).

Let $D$ be a digraph. The **underlying graph** $G(D)$ is the graph obtained from
Let $v$ be a vertex of a digraph $D$. The **outdegree** $d^+_D(v)$ of $v$ in $D$ is the number of arcs directed away from $v$, and the **indegree** $d^-_D(v)$ of $v$ in $D$ is the number of arcs directed into $v$. A vertex $v$ is a **sink** if $d^+_D(v) = 0$ and a **source** if $d^-_D(v) = 0$.

(1.27A) Show that, in any digraph $D$,

$$
\sum_{v \in V(D)} d^-_D(v) = \sum_{v \in V(D)} d^+_D(v) = |E(D)|.
$$

(1.27B) Show that if $D$ is an orientation of a tree $T$, then $D$ must have at least one sink and at least one source.

(1.27C) An oriented tree $T$ is an **arborescence** if $T$ has exactly one source $r$ (called the **root**), such that every other vertex in $T$ has indegree one. A vertex disjoint union of arborescences is called a **branching**. (Equivalently, a branching is an oriented forest in which every vertex has in degree at most one.

Draw an arborescence with $n$ vertices, $1 \leq n \leq 5$.

(1.28) Let $D$ be a digraph. The **subdigraphs** can be defined similar to (1.5). For $x, y \in V(D)$, a **directed** $(x, y)$-**walk** (or $(x, y)$-diwalk) in $D$ is a walk, as in (1.15), with an additional requirement that each $e_i$ be directed from $v_{i-1}$ to $v_i$. Directed trails, paths, circuits in a digraph $D$ are defined similarly.

(1.29) A digraph $D$ is **strongly connected** if for any $x, y \in V(D)$, $D$ has an $(x, y)$-dipath.

(1.30) Show that for a digraph $D$, the following are equivalent:

(i) $D$ is an arborescence.

(ii) $D$ has a unique vertex $r$ such that every vertex $v$ of $D$ can be reached from $r$ by a unique $(r, v)$-dipath.

(iii) $D$ contains a vertex $r$ such that every vertex $v$ can be reached from $r$, but deleting any edge yields a vertex that is not reachable from $r$.

(iv) $D$ can be built up from a vertex $r$ by sequentially adjoining arcs so that the tail of the currently added arc belongs to the digraph having already been constructed while the head is a new vertex.
2. Connectedness and Minimum Spanning Trees

(2.1) Let $G(V,E)$ be a graph and let $A \subset V$ be a nonempty proper subset. Then

$$\delta(A) = \{ e = uv \in E : u \in A \text{ and } v \in V - A \}.$$ 

Edge subsets of the form $\delta(A)$ is called an **edge cut** of $G$. A minimal edge cut is called a **minimal edge cut**.

For a digraph $G(V,E)$, and a nonempty proper subset $A \subset V$,

$$\delta^+(A) = \{ e = (u,v) \in E : u \in A \text{ and } v \in V - A \}, \quad \delta^-(A) = \{ e = (v,u) \in E : u \in A \text{ and } v \in V - A \}.$$ 

(2.2) A graph $G$ is not connected if and only if there exists a nonempty proper subset $A \subset V(G)$ such that $\delta(A) = \emptyset$.

(2.3) Let $H$ be a tree and a subgraph of $G$. Then $H$ is a spanning tree of $G$ if and only if for any edge-cut $D$ of $G$, $E(H) \cap D \neq \emptyset$.

(2.4) **Labelling Algorithm**

**Input:** $G$, and a vertex $v_0 \in V(G)$

**Objective:** Determine all vertices $v \in V(G)$ that can be reachable from $v_0$ (That is, $G$ has a $(v_0,v)$-path).

**Initialization:**

- $R := \{v_0\}$ ($v_0$ is reachable from $v_0$).
- $S := \emptyset$ (no vertices have been scanned).

**Iteration:**

- IF $R - S = \emptyset$, STOP.
- OTHERWISE pick $v \in R - S$. IF $\exists w \in V(G) - R$ with $e = vw \in E(G)$, set $R := R \cup \{w\}$.
  - IF no such $w$ exists, set $S := S \cup \{v\}$.

**Efficiency:**

**Remark:** $G$ is connected if and only if this algorithm outputs $R = V(G)$. Therefore, this algorithm can be used to determine if $G$ is connected.

(2.5) Let $G = (V,E)$ be a graph and $c : E \mapsto \mathbb{R}$ be a (cost) function. We will write $(G,c)$ (or just $G$) to denote a graph with a weight function. For a subset $X \subseteq E$, or a subgraph $H$ of $G$, write

$$c(X) = \sum_{e \in X} c(e), \quad \text{and} \quad c(H) = c(E(H)).$$
If $H$ is a spanning tree of a connected graph $(G, c)$ and if
\[ c(H) = \min\{c(F) : F \text{ is a spanning tree of } G\}, \]
then $H$ is a minimum cost tree (MST) of $G$.

An edge subset $X \subseteq E(G)$ (also viewed as the subgraph of $G$ induced by $X$) is extensible to an MST if there exists an MST $H$ such that $X \subseteq E(H)$. ($\emptyset$ is extensible to an MST $\iff$ MST exists).

(2.6) Suppose that $B \subseteq E$ and $B$ is extensible to an MST, and $D$ is an edge cut with $B \cap D = \emptyset$. Let $e_0 \in D$ be such that
\[ c(e_0) = \min\{c(e) : e \in E(H)\}. \]
Then $B \cup \{e_0\}$ is extensible to an MST.

(2.7) **MST Problem:** Given a graph $(G, c)$ with a weight function $c : E \mapsto \mathbb{R}$, find an MST.

**Prim’s Algorithm for MST**

**Input:** A graph $(G, c)$ with $n = |V(G)|$.

**Objective:** Find an MST

**Initialization:** Set $T := K_1$ (single vertex and edgeless graph).

**Iteration:** While $|V(T)| < n - 1$, find $e_0 \in \delta(V(T))$ such that
\[ c(e_0) = \min\{c(e) : e \in \delta(V(T))\}. \]
Set $T := T \cup e$ (the subgraph induced by the edges $E(T) \cup \{e\}$).

**Efficiency:** In each iteration step, at most $m = |E(G)|$ edges are checked, and exactly $n - 1$ iterations.

**Validity:** Does it output an MST?

(2.8) **Kruskal’s Algorithm for MST**

**Input:** A graph $(G, c)$ with $n = |V(G)|$.

**Objective:** Find an MST

**Initialization:** Set $B := \emptyset$ (can be viewed as the subgraph $B = (V, \emptyset)$), and label the edges $E = \{e_1, e_2, \cdots, e_m\}$ so that
\[ c(e_1) \leq c(e_2) \leq \cdots \leq c(e_m). \]
Iteration: FOR $i = 1$ TO $m$, DO

IF $B \cup e_i$ is a forest, set $B := B \cup \{e_i\}$.

IF $|B| = n - 1$, STOP.

Efficiency: To be discussed.

Validity: Does it output an MST?
3. The Shortest Path Problem

1. Introduction

(3.1.1) Let $G(V, E)$ be a digraph in which each arc $e \in E$ has a real valued weight $w(e)$ assigned to it (such a digraph is often called a network, or just a net. Sometimes we denote a net by $(G, w)$ to emphasize that $G$ has a weight function). If $e = (u, v)$ (an arc oriented from $u$ to $v$), we also write $w_{u,v}$ or $w(u, v)$ for $w(e)$. The weight (usually representing the length or the cost) of a path $P = i_0, a_1, v_1, a_2, \cdots, i_{p-1}a_p, i_p$ is

$$w(P) = \sum_{i=1}^{p} w(a_i).$$

More generally, if $H$ is a subgraph of $G$, then

$$w(H) = \sum_{e \in A(H)} w(e).$$

An $(s,t)$-dipath $P$ is a shortest $(s,t)$-path in $G$ if $w(P)$ is minimized, among all $(s,t)$-dipaths in $G$.

(3.1.2) Let $G$ be a digraph with a weight function representing distance and let $s \in V(G)$ be a given vertex. The **Shortest Path Problem (SPP)** asks to determine the shortest distances from $s$ to any vertex $v$ of $G$.

(3.1.3) A dicircuit $C$ of $G$ **negative circuit** if $w(C) < 0$. Note that if $G$ has a negative circuit, then for some pair of vertices $s,t \in V(G), G$ does not have a shortest $(s,t)$-path.

(3.1.4) Let $G$ be a digraph without negative circuits, let $s,t \in V(G)$ be two distinct vertices, and let $P = i_0, e_1, v_1, e_2, \cdots, e_{p-1}v_p, i_p$ be a directed $(s,t)$-path, where $s = i_0$ and $t = i_p$. Each of the following holds:

(i) If $P$ is a shortest $(s,t)$-dipath, then $P$ contains a shortest simple $(s,t)$-path.

(ii) If $P$ is a shortest $(s,t)$-path, then for $i_0 \leq i_a < i_b \leq i_p$, the path

$$P_{i_a,i_b} = i_a, e_{i_a}, v_{i_a+1}, e_{i_{a}+1}, i_{a}+2, \cdots, i_{b-1}, e_{i_b}, i_b$$

is a shortest $(i_a, i_b)$-path.

**Proof:** (i) If $P$ has no dicircuits, then $P$ itself is a shortest $(s,t)$-path. If $P$ has a dicircuit $C$, then $P - C$ is also an $(s,t)$-dipath whose length is $w(P) - w(C)$. Since $D$ has no negative cycles, $w(C) \geq 0$ and so one can replace $P$ by $P - C$. Repeat this process until
the resulting dipath has no dicircuits.

(ii) If $P'$ is a shortest $(i_a, i_b)$-path with $w(P') < w(P_{i_a, i_b})$, then the path

$$s = i_0, e_1, v_1, e_2, \cdots, i_a, P', i_b, \cdots, i_{p-1}, e_p, i_p = t$$

is a shorter $(s, t)$-path, contrary to the assumption that $P$ is a shortest $(s, t)$-path. □

(3.1.5) Exercise: Show that if $G$ is strongly connected, and if $G$ has no negative dicircuits, then for any $s, t \in V(G)$, $G$ has a shortest $(s, t)$-path.

(3.1.6) Exercise: If $T_s$ is an arborescence rooted at $s$, then for any node $t \in V(T_s) - \{s\}$, $T_s$ has a unique $(s, t)$-dipath.

(3.1.7) A tree $T$ of a network $G$ is a shortest path arborescence rooted at $s$ is an arborescence rooted at $s$ such that

(i) every vertex reachable from $s$ in $G$ by a dipath is in $T$,

(ii) for every vertex $v \in V(T)$, the unique $(s, v)$-path in $T$ is a shortest $(s, v)$-path in $D$.

(3.1.8) Exercise: If $G$ has no negative cycles, then for every vertex $s \in V(G)$, there exists a shortest path arborescence rooted at $s$.

(3.1.9) (Existence of Solution to SPP) If $D$ is strongly connected, and if $G$ has no negative cycles, then for every vertex $s \in V(G)$, there exists a spanning shortest path arborescence rooted at $s$.

2. The Bellman Inequalities

Notation: In this section, $d = (d(v_1), d(v_2), \cdots, d(v_n))$ is a function with domain in $V(G)$ which is also expressed as a vector. Note that in this section, $d(v)$ will be the component of $d$ at $v$, not the degree of $v$ in $G$.

(3.2.1) Suppose that $T$ is a shortest path arborescence rooted at $s$ in a net $D$. For any $v \in V(T) - \{s\}$, let $p(v)$, called the predecessor of $v$, denote the unique node such that $(p(v), v) \in E(T)$ and define $p(s) = \emptyset$.

(3.2.2) Suppose that $G$ has a shortest path arborescence. For a given $s \in V(G)$, and $u \in V(G) - \{s\}$, let $d(u)$ denote the length of a shortest $(s, u)$-path, and $d(u) = \infty$ if $G$ has no $(s, u)$-dipaths.
(3.2.3) The labelling $d(u)$ in (3.2.2) satisfying the Bellman inequalities:

$$\begin{align*}
d(s) &= 0 \\
d(v) - d(u) &\leq w_{u,v} \text{ for all } (u, v) \in E
\end{align*}$$

Moreover, for each $vin V - \{s\}$, there is at least one $u$ with $(u, v) \in E$, such that $d(v) - d(u) \leq w_{u,v}$.

**Proof:** Let $P$ be an $(s, v)$-dipath with the last arc $e = (u, v)$ such that the portion from $s$ to $u$ is a shortest $(s, u)$-dipath. Then $w(P) = d(u) + w_{u,v} \geq d(v)$.

Let $Q$ denote a shortest $(s, v)$-dipath with the last arc $e = (u, v)$. By (3.1.4), the section from $s$ to $u$ of $Q$ is a shortest $(s, u)$-dipath whose length is $d(u)$. Therefore, $d(v) = w(Q) = d(u) + w_{u,v}$.

(3.2.4) Let $y : V \mapsto R$ be a function (vertex labelling). If $y$ is a solution of the Bellman’s inequalities, then $y$ is called a feasible potential (of the particular instance of SPP).

(3.2.5) Suppose that $G$ has a feasible potential $y$. Then each of the following holds:

(i) For any $v \in V(G)$, any $(s, v)$-dipath has length at least $y(v)$. In particular, any shortest $(s, v)$-dipath has length at least $y(v)$.

(ii) $G$ has no negative dicircuits.

**Proof:**

(i) Let $P$ be an $(s, v)$-dipath. Write $P = v_0, e_1, \ldots, e_k, v_k$ with $s = v_0$ and $v = v_k$. Then as $y(v_0) = y(s) = 0$,

$$w(P) = \sum_{i=1}^{k} w(e_i) \geq \sum_{i=1}^{k} (y(v_i) - y(v_{i-1})) = y(v_k) - y(v_0) = y(v).$$

(ii) Let $C = v_0, e_1, \ldots, e_k, v_k e_0, v_0$ be a negative dicircuit with $w(C) = -c < 0$. Let $P$ be an $(s, v_0)$-dipath and let $P(k)$ be the $(s, v_0)$-dipath from $P$ by continuing going around $C$ $k$ times. Then by (i),

$$y(v_0) \leq w(P(k)) = w(P) + k(-c) = w(P) - kc.$$  

But the right hand side goes to negative infinity, and so this is impossible.

(3.2.6) Exercise: Let $d$ be a feasible potential, and let $E^- = \{(u, v) \in E : d(v) - d(u) = w_{u,v}\}$. Show that every $(s, u)$-dipath using arcs in $E^-$ is a shortest path. (Apply (3.2.5)(ii) to show that such a path will have the same distance as a shortest path).

(3.2.7) Suppose that $G$ has a spanning arborescence rooted at $s$ and that $G$ has no negative cycles. Each of the following holds:

(i) The Bellman’s inequalities have a solution $d$ such that $d(v)$ is the length of a shortest
(s, v)-path in G.

(ii) A spanning arborescence T rooted at s is a shortest path arborescence if and only if \(d(v) - d(u) = u_{uv},\) for all \((u, v) \in E.\)

**Proof:**

(i). By (3.1.7A), G has a spanning shortest path arborescence \(T'\) rooted at s. Let \(d(u)\) be the length of the unique \((s, u)-\)dipath in \(T'.\) Then \(d\) satisfies the Bellman’s inequalities.

(ii). The if part comes from (3.2.6) and the only if part follows by (3.2.5) and (3.1.6). □

(3.2.8) Suppose that G has a spanning arborescence rooted at s. Then the following are equivalent.

(i) SPP has a solution \(d.\)

(ii) \(G\) has a feasible potential.

(iii) \(G\) has no negative dicircuits.

**Proof:** By (3.2.3), we have (i) \(\implies\) (ii). By (3.2.5)(ii), we have (ii) \(\implies\) (iii). The algorithms in the next section show that if \(G\) has no negative dicircuits, then the algorithms will find a solution of the SPP, and so (i) must follow from (iii). (Another way to see it is to note that (iii) implies that \(d(u) > -\infty,\) while the assumption that \(G\) has a spanning arborescence rooted at s implies that \(d(u) < \infty).\)

3. Dijkstra-Ford’s Algorithm for Networks without Negative Dicircuits

The Algorithms are motivated by (3.2.7). The main idea is to revise the current labelling until it becomes a feasible potential. Throughout this section, we assume that \(G\) is a digraph with an edge weight function \(w\) representing the distance, such that \(G\) has no negative dicircuits.

(3.3.1) **Ford’s Algorithm** for nets without negative dicircuits.

**Input:** A net G with a weight function \(w: E \mapsto R\) and a distinguished vertex \(s\) such that \(G\) has no negative dicircuits.

Two symbols 0, \(-1\) which are not in \(V.\)

**Objective:** A solution \(y\) to the SPP, and a shortest path arborescence rooted at \(s.\)

**Initialization:**

\[
y(s) := 0; y(v) := \infty \text{ for all } v \in V - \{s\}; \\
p(s) := 0, \text{ and } p(v) := -1, \text{ (meaning } p(v) \text{ is not defined yet).}
\]

**Iteration:**

While there is an edge \((t, v) \in E\) such that \(y(v) > y(t) + w(t, v),\) then reset \(y(v) := y(t) + w(t, v)\) and \(p(v) = t.\) (\(p(v)\) is the predecessor of \(v\) in a shortest \((s, v)-\)dipath.)
(3.3.2) Let \( G(V, E) \) be a network such that \( G \) has no negative dicircuits. Let \( s \) be a vertex of \( V(G) \). Suppose that for some set \( S \subseteq V(D) \) with \( s \in S \), the length \( d(u) \) of the shortest distance from \( s \) to \( u \) is known for all \( u \in S \) (with \( d(s) = 0 \), of course). For each \( v \in V - S \), we compute a “temporary label” \( l(v) \) by

\[
l(v) = \begin{cases} 
\min\{d(u) + w(u, v) : u \in S \text{ and } (u, v) \in E\} & \text{if } \exists u \in S \text{ such that } (u, v) \in E \\
\infty & \text{if no such edges exist.}
\end{cases}
\]

If \( v \in V - S \) is so chosen that \( l(v) \) is minimized among all vertices in \( V - S \), then \( d(v) = l(v) \).

**Proof:** Suppose that \( P = v_0, e_1, v_1, \ldots, v_i, \ldots, v \) (with \( s = v_0 \)) is a shortest \((s, v)\)-path. Pick smallest \( k \) such that \( v_k \notin V(P) \cap S \). Then by the choice of \( v \), \( l(v_k) = d(v_{k-1}) + w(v_{k-1}, v_k) \geq l(v) \). As \( l(v) \) represents the length of an \((s, v)\)-dipath, it follows by the definition of \( d \) that \( l(v) \geq d(v) \). Thus

\[
d(v) = w(P) \geq d(v_{k-1}) + w(v_{k-1}, v_k) = l(v_k) \geq l(v) \geq d(v).
\]

(3.3.3) **Dijkstra’s Algorithm** for nets without negative dicircuits.

**Input:** A net \( G \) with a weight function \( w : E \to R \) and a distinguished vertex \( s \) such that \( G \) has no negative dicircuits.

Two symbols \( 0, -1 \) which are not in \( V \).

**Objective:** a solution \( y \) to the SPP, and

a shortest path arborescence rooted at \( s \).

**Initialization:**

\[
S := \{s\}; y(s) := 0; l(v) := \infty \text{ for all } v \in V - S,
\]

\((S \text{ represents the set of all vertices reachable from } s \text{ in } G)\)

\(p(s) := 0\), and \( p(v) := -1\), (meaning \( p(v) \) is not defined yet),

\(t := s\) (\(t \) represents the latest vertex added to \( S \)).

**Iteration:**

**Step 1:** (Revising the temporary label)

For each \((t, v) \in E \text{ with } v \in V - S,\)

if \( l(v) > y(t) + w(t, v) \), then set \( l(v) := y(t) + w(t, v) \) and \( p(v) = t \).

\((p(v) \text{ is the predecessor of } v \text{ in a shortest } (s, v)\)-dipath.)

**Step 2:** (Setting the final label)

Choose a new \( t \) such that \( l(t) = \min\{l(v) : v \in V - S\} \).

If \( l(t) = \infty \), then all vertices reachable from \( s \) are in \( S \), STOP.

Otherwise set \( S := S \cup \{t\} \) and \( y(t) := l(t) \).

If \( S = V \), STOP, otherwise GOTO Step 1.

(3.3.4) Suppose that \( G \) has no negative dicircuits. At any stage of the Algorithms (3.3.1) and (3.3.3), each of the following holds.
(i) For each \((p(v), v) \in E\), \(y(v) \geq y(p(v)) + w(p(v), v)\).

(ii) If \(y(v) \neq \infty\), then \(y(v)\) is the length of a simple \((s, v)\)-dipath.

(iii) If \(p(v) \neq -1\), then \(p\) defines a simple \((s, v)\)-dipath which has length \(y(v)\).

(iv) At the termination, \(y\) is a feasible potential.

**Proof:**

(i) At the initial step, \(y(v) = \infty\) and so the inequality must hold. In the first time when \(y(v)\) and \(y(p(v))\) are assigned finite values, we have \(y(v) = y(p(v)) + w(p(v), v)\). After that, \(y(p(v))\) may be reset, but it will only be getting smaller.

(ii) Let \(y^j(v)\) denote the value of \(y(v)\) after \(j\) iterations. If \(y^j(v) \neq -\infty\), then \(p(v) \neq -1\). Thus \(y(v)\) is finite and so by following the pointer \(p\), we conclude that \(p^j(v)\) is the length of an \((s, v)\)-dipath \(P\).

If \(P\) is a simple path, then done. Otherwise we may assume that \(P\) contains a dicircuit \(C = v_0, e_1, v_1, \ldots, e_k, v_k\) with \(v_0 = v_k\), and iterations numbers \(q_0 < q_1 < \cdots < q_k\) such that at iteration \(q_i\), \(v_i\) was set to be \(p(v_{i+1})\). Thus, for each \(i = 1, 2, \ldots, k\),

\[
y^{q_i-1}(v_{i-1}) + w(e_i) = y^{q_i}(v_i).
\]

It follows that

\[
w(C) = \sum_{i=1}^{k} w(e_i) = y^{q_i}(v_i) - y^{q_i-1}(v_{i-1}) = y^{q_k}(v_k) - y^{q_0}(v_0).
\]

But at the \(q_k\) iteration, \(y^{q_k}(v_k)\) replaces \(y^{q_0}(v_0)\) only if \(y^{q_k}(v_k) < y^{q_0}(v_0)\), and so \(w(C) < 0\), a contradiction.

(iii) If \(p(v) \neq -1\), then \(y(v) \neq -\infty\). By (ii), \(y(v)\) is the length of a simple \((s, v)\)-dipath \(P\).

From the argument above, \(P\) can be obtained backwardly by

\[
v, p(v), p^2(v), \ldots, p^{k'}(v), \text{ where } p^{k'}(v) = s.
\]

In fact, if the path \(P = p^{k'}(v), p^{k'-1}(v), \ldots, p(v), v\) is not simple path, then this sequence has a dicircuit \(C = v_0, e_1, v_1, \ldots, e_k, v_k\) with \(v_0 = v_k\). Again, as \(y(v_k)\) replaces \(y(v_0)\) in the algorithm, we must have \(y(v_k) > y(v_{k-1}) + w(e_k) = y(v_k)\). It follows that by (i),

\[
w(C) = \sum_{i=1}^{k} w(e_i) \leq \sum_{i=1}^{k} y(v_i) - y(v_{i-1}) = y(v_k) - y(v_0) < 0,
\]

contrary to the assumption that \(G\) has no negative dicircuits.

Therefore \(p\) gives rise to a simple \((s, v)\)-dipath \(P\). It remains to show that \(w(P) \leq y(v)\).

Let \(P = v_0, e_1, v_1, \cdots, e_k, v_k\) with \(s = v_0\) and \(v = v_k\). Then by (i) and by \(y(s) = 0\),

\[
w(P) = \sum_{i=1}^{k} w(e_i) \leq \sum_{i=1}^{k} y(v_i) - y(v_{i-1}) = y(v_k) - y(v_0) = y(v).
\]
(iv) If $y$ is not a feasible potential, then there would be an edge $(t, v) \in E$ such that $y(v) > y(t) + w(t, v)$, and so the iteration step will not stop.

(3.3.5) Suppose that $G$ has a spanning arborescence rooted at $s$ and $G$ has no negative dicircuits. At the termination of Algorithms (3.3.1) and Algorithm (3.3.3), $y$ is a solution of the SPP.

**Proof:** (For Algorithm (3.3.3), this is assured by (3.3.2).)

Let $d(v)$ denote the length of a shortest $(s, v)$-dipath. By (3.3.2), $G$ has an $(s, v)$-dipath $P$ such that $w(P) \leq y(v)$. By (3.3.2)(iv) and (3.2.5)(i), we must have $w(P) = y(v)$. Thus $d(v) \leq y(v)$. Now let $P'$ be a shortest $(s, v)$-dipath. By (3.3.2)(iv), $y$ is a feasible potential and by (3.2.5)(ii), $d(v) = w(P) \geq y(v)$, and so $y(v) = d(v)$.

(3.3.6) Each of the following holds.

(i) Algorithm (3.3.1) and Algorithm (3.3.3) will stop in a finite number of steps.

(ii) At termination, $\forall v \in V(G) - \{s\}$, $p$ defines a shortest $(s, v)$-dipath.

**Proof:** (i). For each given $v$, $G$ has finitely many $(s, v)$-dipaths in $G$, and so by (3.3.4)(ii), there are only finitely many possible values for $y(v)$. At each iteration step, $y(v)$ decreases from one of the possible value to another (and smaller) such possible values; and as the program runs, no values of $y(v)$ is increased. Therefore, the algorithm must stop in a finitely many steps.

(ii). By (3.3.4)(iii), $p$ defines an $(s, v)$-dipath $P$ with $w(P) \leq y(v)$. Since $y$ is a feasible potential, it follows by (3.2.5)(i) that $w(P) = y(v)$.

(3.3.7) The value $p(v)$ constructed are valid predecessor pointers for a shortest path arborescence. (This follows from (3.3.6)(ii).)

(3.3.8) Exercise: Show that if we arrange the vertices of $G$ in the sequence $s = v_0, v_1, \ldots, v_n$ corresponding to the order in which the value $y(v_i)$ were computed in (3.3.3), then

$$y(v_0) \leq y(v_1) \leq \cdots \leq y(v_n).$$

4. **Acyclic Digraph**

(3.4.1) A digraph $G$ is **acyclic** if $G$ has no dicircuits.

(3.4.1A) Exercise: Find an acyclic digraph $G$ whose underlying graph is not a tree.

(3.4.2) Exercise: If a digraph $G$ does not have a vertex of out degree equal to zero (or
dually, if $G$ does not have a vertex of indegree equal to zero), then $G$ has a dicircuit.

(3.4.3) A digraph $G(V,E)$ with $n = |V|$ is acyclic if and only if the vertices of $G$ can be ordered $v_1, \ldots, v_n$ such that we have $i < j$, $\forall (v_i, v_j) \in E$. (Such an ordering of an acyclic digraph is called a canonical ordering).

**Proof:** Any directed circuit will fail the condition $i < j$ for every $(v_i, v_j) \in E$. Hence “if”.

Assume that $G$ is acyclic. By (3.4.2), $G$ must have a vertex $z$ such that the out degree of $z$ in $G$ is zero. Label $z$ as $v_n$. Now consider the graph $G' = G - v_n$. Since $G$ is acyclic, $G'$ is also acyclic, and so by induction on $|V|$, we can write $V(G') = \{v_1, v_2, \ldots, v_{n-1}\}$, such that $(v_i, v_j) \in E$ only if $i < j$. Since the out degree of $v_n$ is zero, the labelling $V(G) = \{v_1, v_2, \ldots, v_{n-1}, v_n\}$ satisfies the requirement of (3.4.3).

**Remark:** The “only if” part also follows from the Algorithm (3.4.4) below.

(3.4.4) **Algorithm: Topological Sort**

**Input:** A digraph $G(V,E)$.

**Output:** A dicircuit in $G$, or an ordering $v_1, \ldots, v_n$ of $V$ such that if $(v_i, v_j) \in E$, then $i < j$.

**Initialization:** $j = 1$, (counter of the subscripts, the next vertex is to be $v_j$).

**Iteration:**

**Step 1** If $G$ has a source $v'$, then DO

$v_j := v'$, $j := j + 1$, $G := G - v'$, and repeat Step 1.

**Step 2** Otherwise use (3.4.2) to find a dicircuit.

(3.4.5) **Validity** of Algorithm (3.4.4). (This is in fact follows from (3.4.3)).

Suppose that $G$ does not have a dicircuit. Then any subgraph of $G$ is also acyclic. By (3.4.2), any subgraph of $G$ must have a source (a vertex with indegree zero). Hence we can label the vertices in $V(G) = \{v_1, v_2, \ldots, v_n\}$ such that for each $i = 1, 2, \cdots, n - 1$, $v_i$ is a source of the subgraph of $G$ induced by $\{v_{i+1}, \ldots, v_n\}$.

But this is precise the same way that Algorithm (3.4.4) labels the vertices in Step 1. Therefore, if $(v_i, v_j) \in E$ and if $j < i$, then $v_j$ will be a source in $G[\{v_j + 1, \cdots, v_i, \cdots, v_n\}]$ (subgraph induced by $\{v_j + 1, \cdots, v_i, \cdots, v_n\}$), contrary to the assumption that $(v_i, v_j) \in E$.

(3.4.6) **Efficiency**
Bellman's Algorithm for Acyclic Networks

Input: An acyclic network $G$ with a canonical ordering $v_1, \cdots, v_n$.

Output: A shortest path arborescence rooted at $s$ for each $s \in V(G)$.

Initialization: Suppose $s = v_k$. Set $d(v_i) = \infty$ for $1 \leq i \leq k - 1$; $d(v_k) = 0$; $p(v_i) = \emptyset$, for $1 \leq i \leq k$.

Iteration: For $i = k + 1, n$, DO

find $j < i$ such that $d(v_j) + w(v_j, v_i) = \min\{d(v_h) + w(v_h, v_i) : 1 \leq h \leq i\}$

set $d(v_i) := d(v_j) + w(v_j, v_i)$;

IF $d(i) < \infty$, THEN $p(v_i) := v_j$, ELSE $p(i) = \emptyset$.

Exercise: Show that Algorithm (3.4.7) is correct, and discuss its efficiency.

5. Circuits and Negative Weights

In this section we shall assume that it is possible for $G$ to have a negative dicircuit. Thus our algorithm should either find a solution for SPP or detect a negative dicircuit.

Ford-Bellman Algorithm

Input: A network $G$ with $n$ vertices and a fixed vertex $s \in V(G)$.

Output: A shortest path arborescence of $G$ rooted at $s$ OR a negative dicircuit of $G$.

Initialization: Set $p(s) = \emptyset$, $d^0(s) = 0$, and $d^0(v) = \infty$, $\forall v \in V - \{s\}$.

Iteration:

(i) FOR $k = 1$ to $n$, DO

FOR each $v \in V$, DO

compute $d^k(v) = \min\{d^{k-1}(v), \min\{d^{k-1}(u) + w(u, v) : (u, v) \in E\}\}$

IF $d^k(v) < d^{k-1}(v)$,

THEN $p(v) = u'$ where $(u', v)$ gives the minimum above.

(ii) IF $d^n(v) = d^{n-1}(v)$, $\forall v \in V$,

THEN use the predecessor pointers $p(\cdot)$ to construct a shortest path arborescence rooted at $s$.

IF $d^n(v) < d^{n-1}(v)$ for some vertex $v$,

THEN use the predecessor pointers $p(\cdot)$ to trace down a negative dicircuit.

Exercise: Show that in Algorithm (3.5.1), for each $k = 0, 1, \cdots, n$, $d^k(v)$ is the length of a shortest $(s, v)$-dipath using at most $k$ arcs. (Hint: Induction on $k$.)
(3.5.3) Exercise: Show that if $G$ has a negative dicircuit, then Algorithm (3.5.1) will correctly discover it.

(3.5.4) Exercise: Prove the correctness of Algorithm (3.5.1) and discuss its efficiency.

(3.5.5) **Floyd-Warshall Algorithm**

Input: A network $G(V, E)$ with $n$ vertices and a fixed vertex $s \in V(G)$.

Output: A shortest path arborescence of $G$ rooted at $s$ OR a negative dicircuit.

Initialization:

Set $d^0(u, v) = w(u, v), \forall (u, v) \in E$;
set $d^0(u, u) = 0, \forall u \in V$;
set $d^0(u, v) = \infty, \forall (u, v) \notin E$.

Iteration: FOR $k = 1$ to $n$, DO

FOR each $u, v \in V$, DO

$$d^k(u, v) = \min\{d^{k-1}(u, v), d^{k-1}(u, v') + d^{k-1}(v', v) : v' \in V\}$$

IF $d^n(u, u) < 0, \forall u \in V$, THEN $G$ has a negative dicircuit.

IF $d^n(u, u) = 0, \forall u \in V$, THEN

$d^n(u, u)$ is the length of a shortest $(s, u)$-path.

(3.5.6) Exercise: Prove the correctness of Algorithm (3.5.5) and discuss its efficiency.

6. **Additional Exercises**

(3.6.1) Mr. Dow Jones, 50 years old, wishes to place his IRA (Individual Retirement Account) funds in various investment opportunities so that at the age of 65 years, when he withdraws the funds, he has accrued maximum possible amounts of money. Assume that Mr. Jones knows the investment alternatives for the next 15 years: their maturity (in years) and the appreciation they offer. How would you formulate this investment problem as a shortest path problem, assuming that at any point in time, Mr. Jones invests all his funds in a single investment alternatives?

(3.6.1A) (Continuation of (6.1)) The following is a chart made by Mr. Jones summarizing the information of the investment alternatives (Inv. Alt.) available:

<table>
<thead>
<tr>
<th>Inv. Alt.</th>
<th>From</th>
<th>To</th>
<th>Appreciation</th>
<th>Inv. Alt.</th>
<th>From</th>
<th>To</th>
<th>Appreciation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Money Market</td>
<td>Year 1</td>
<td>Year 2</td>
<td>4.5%</td>
<td>Money Market</td>
<td>Year 9</td>
<td>Year 10</td>
<td>4.5%</td>
</tr>
<tr>
<td>3-year CD</td>
<td>Year 1</td>
<td>Year 3</td>
<td>5.0%</td>
<td>3-year CD</td>
<td>Year 9</td>
<td>Year 11</td>
<td>5.0%</td>
</tr>
<tr>
<td>5-year CD</td>
<td>Year 1</td>
<td>Year 5</td>
<td>5.5%</td>
<td>5-year CD</td>
<td>Year 9</td>
<td>Year 13</td>
<td>6.25%</td>
</tr>
</tbody>
</table>
Find a most profitable investment strategy for Mr. Dow Jones.

(3.6.2) Beverly owns a vacation home in Cape Cod that she wishes to rent for the period May 1 to August 31. She has solicited a number of bids, each having the following
form: the day the rental starts (the rental starts at 3pm.), the day the rental ends (checkout time is noon), and the total number of the bid (in dollars). Beverly wants to identify a selection of bids that would maximize her total revenue.

(3.6.2A) Use Shortest Path model to find a most profitable selection.
(3.6.2B) Randomly generate 160 bids, each covering at least 3 days, such that there is a bid that starts on every day of the four months (May, June, July and August). Then find an optimal selection.

(3.6.3) The money changing problem requires that we determine whether we can change a given number \( p \) into coins of known denominations \( a_1, a_2, \cdots, a_k \). For example, if \( k = 3 \), \( a_1 = 3 \), \( a_2 = 5 \) and \( a_3 = 7 \), then we can change all numbers in the set \( \{8, 10, 12, 13, 54\} \); on the other hand, we cannot change the number 4. In general, the money changing problem asks whether \( p = \sum_{i=1}^{k} a_i x_i \) for some nonnegative integer \( x_1, x_2, \cdots, x_k \). Suppose that \( a_1, a_2, \cdots, a_k \) are known. Answer each of the following.

(6.3A) Describe a method for identifying all numbers in a given range of numbers \([l, u]\) that we can change.
(6.3B) Describe a method that identifies whether we can change a given number \( p \), and if so, the identifies a demolition? with the least number of coins.

(3.6.4) Cluster Analysis. Consider a given integer \( p \geq 1 \) and a set of \( n \) scalar numbers \( a_1, a_2, \cdots, a_n \) arranged in nondecreasing order of their values. We wish to partition these numbers into clusters (or groups) so that
(i) each cluster contains at least \( p \) numbers;
(ii) each cluster contains consecutive numbers from the list \( a_1, a_2, \cdots, a_n \); and
(iii) the sum of the squared deviation of the numbers from their cluster means is as small as possible.
Here, if \( S \) is a set of numbers, then the mean of \( S \) is \( m(S) = (\sum_{a_i \in S} a_i)/|S| \) and the squared deviation of the numbers from their mean is \((a_k - m(S))^2\), for all \( a_k \in S \).

(3.6.4A) Show how to formulate this problem as a shortest path problem.
(3.6.4B) Illustrate your solution to (6.4A) by using these data: \( p = 2, n = 6, a_1 = 0.5, a_2 = 0.8, a_3 = 1.1, a_4 = 1.5, a_5 = 1.6 \) and \( a_6 = 2.0 \).

(3.6.5) (Malik, Mittal and Gupta) Consider a network \( G(V, E) \) without any negative di-circuits. For every vertex \( j \in V \), let \( d^s(j) \) denote the length of a shortest path from vertex \( s \) to vertex \( j \), and \( d^t(j) \) denote the length of a shortest path from vertex \( t \) to vertex \( t \).
(i) Show that an arc \((i, j)\) is in a shortest \((s, t)\)-path if and only if \( d^s(t) = d^s(i) + w(i, j) + d^t(j) \).
(ii) Show that \( d^s(t) = \min\{d^s(i) + w(i, j) + d^t(j) : (i, j) \in E\} \).
(iii) Randomly generate a network with at least 5 vertices and randomly generate a set
of nonnegative numbers \( w(i, j) \) for each pair of \( (i, j) \in E \). Specify two distinct vertices \( s \) and \( t \) and then verify (ii) by direct computation.

(iv) Does (ii) give you an idea to compute shortest path lengths? Write an algorithm using (ii) to find a shortest \((s, t)\)-path. What is the complexity of your algorithm?

(3.6.6) Which of the following claims are true and which are false? Justify your answers by giving proofs or constructing counterexamples.

(i) If all the arc length \( w(i, j) \)'s in a network \( D = D(V, E) \) are distinct, then \( D \) has a unique shortest path arborescence.

(ii) If we eliminate the directions of all the arcs in a network \( G \) with positive arc lengths, the shortest path length will not change.

(iii) In a shortest path problem, if each arc length increases by \( k \) units, the shortest path distances increase by a multiple of \( k \).

(iv) In a shortest path problem, if each arc length decreases by \( k \) units, the shortest path distances decrease by a multiple of \( k \).

(v) Among all shortest paths in a network, Dijkstra-Ford’s Algorithm always finds a shortest path with the least number of arcs.

(3.6.7) Suppose that you are given a shortest path problem in which all arc lengths are the same. How will you solve this problem in the least possible time? (Describe your solution and explain why it uses the least possible time.)

(3.6.8) Show that in the shortest path problem if the length of some arc decreases by \( k \) units, then the shortest path length between any pair of vertices decreases at most \( k \) units.

(3.6.9) A **longest** \((s, t)\)-path is an \((s, t)\)-dipath with the maximum length. Suggest an \(O(m)\) algorithm for determining a longest path in an acyclic network with nonnegative arc lengths. Will your algorithm work if the network contains a dicircuit?

(3.6.10) Suppose that every dicircuit in a digraph \( G \) has positive length. Show that a shortest \((s, t)\)-dipath is always a simple dipath. Construct an example for which the first shortest dipath is a simple path but the second one is not.
4 Maximum Flow and Minimum Cut

1. Introduction

(4.1.1) Let \( G(V,E) \) be a network with a weight function \( u : E \to \mathbb{R}^+ \cup \{0\} \), the set of non-negative real numbers (\( u \) is called the capacity of the net \( (G,u) \)). We shall denote \( V = \{1,2,\cdots,n\} \), where \( n = |V| \). For a vertex \( v \in V(G) \), let

\[
N^+(v) = \{ z \in V(G) : (v,z) \in E(G) \}, \quad N^-(v) = \{ z \in V(G) : (z,v) \in E(G) \}
\]

and

\[
N(v) = N^+(v) \cup N^-(v).
\]

Notation:

(i) For a function \( f : E \to \mathbb{R} \) written in the vector form, we also use \( f(e) \) to denote \( f(e) \). Thus the net \( (G,f) \) will also be written as \( (G,f) \).

(ii) For (not necessarily distinct) subsets \( X,Y \subseteq V \), we write

\[
x(X,Y) = \sum_{i \in X, j \in Y, (i,j) \in E} x(i,j).
\]

(iii) Let \( G = (V,E) \) be a digraph. For a function \( f : E \to \mathbb{R} \), the boundary of \( f \) is a function \( \partial f : V \to \mathbb{R} \) such that for any vertex \( v \in V \), \( (\partial f(v) \) counts the net in amount of \( f \))

\[
\partial f(v) = f(V - \{v\}, \{v\}) - f(\{v\}, V - \{v\}).
\]

Definition: Let \( s,t \in V(G) \). An \( (s,t) \)-flow in \( G \) is a vector

\[
x = (x(i,j) : (i,j) \in E),
\]
satisfying the local balanced condition:

\[(4.1.1A) \quad \partial x(v) = x(V - \{j\}, \{j\}) - x(\{j\}, V - \{j\}) = 0, \quad \forall j \in V - \{s,t\}.\]

An \( (s,t) \)-flow \( x \) is feasible (in \( (G,u) \)) if it also satisfies (the feasibility conditions):

\[(4.1.1B) \quad 0 \leq x(i,j) \leq u(i,j), \quad \forall (i,j) \in E.\]

The value of an \( (s,t) \)-flow \( x \) is

\[(4.1.1C) \quad v(x) = x(V - \{t\}, \{t\}) - x(\{t\}, V - \{t\}).\]

(4.1.2) A cut is a set of arcs directed from vertices in a subset \( S \subset V \) to vertices in \( V - S \). Such a cut is denoted by \( (S,\bar{S}) \), where \( \bar{S} = V - S \). The capacity of a cut \( (S,\bar{S}) \) is

\[
u(S,\bar{S}) = \sum_{i \in S,j \in S,(i,j) \in E} u(i,j).
\]
An \((s,t)\)-cut is a cut \((S, \bar{S})\) such that \(s \in S\) and \(t \in \bar{S}\).

**Notation:** Let \(S \subset V\) be a subset. Then both \((S, \bar{S})\) and \(\delta(S)\) denote the same edge cut

\[
\delta(S) = (S, \bar{S}) = \{(i, j) \in E : i \in S, j \in \bar{S}\}.
\]

(4.1.2A) Let \(x\) be an \((s,t)\)-flow and let \((S, \bar{S})\) be an \((s,t)\)-cut. Then

\[
v(x) = x(S, \bar{S}) - x(\bar{S}, S).
\]

**Proof:** For the given \(S\), let \(S^o = \{v \in \bar{S} - \{t\} : v\) is not incident with an edge in \(\delta(S) \cup \delta(\bar{S})\}\).

By (4.1.1A) and (4.1.1C),

\[
x(S, \bar{S}) - x(\bar{S}, S) = \sum_{i \in S, j \in \bar{S}, (i,j) \in E} x(i,j) - \sum_{i' \in \bar{S}, j' \in \bar{S}, (i',j') \in E} x(i',j') \\
= \sum_{i \in S, j \in \bar{S}, (i,j) \in E} x(i,j) - \sum_{i' \in \bar{S}, j' \in \bar{S}, (i',j') \in E} x(i',j') \\
+ \sum_{v \in S^o} \left[ \sum_{i \in V - \{t\}} x(v,i) - \sum_{(i,v) \in E} x(i,v) \right] \\
= \sum_{v \in S \cup N(t) - \{t\}} \left[ \sum_{i \in V - \{t\}} x(v,i) - \sum_{(i,v) \in E} x(i,v) \right] \\
= \sum_{i \in V - \{t\}} x(i,t) - \sum_{i \in V - \{t\}} x(t,i) \\
= x(V - \{t\}, \{t\}) - x(\{t\}, V - \{t\}) = v(x).
\]

(4.1.2B) Exercise: Use (4.1.1A) to show that the value of an \((s,t)\)-flow \(x\) is

\[
v(x) = x(\{s\}, V - \{s\}) - x(V - \{s\}, \{s\}).
\]

(4.1.3) Let \(x\) be a feasible \((s,t)\)-flow and \((S, \bar{S})\) be an \((s,t)\)-cut. Then

(i) \(v(x) \leq u(S, \bar{S})\).

(ii) \(v(x) = u(S, \bar{S})\) if and only if

\[
x(i,j) = u(i,j) \ \forall (i,j) \in (S, \bar{S}), \text{ and } x(i,j) = 0 \ \forall (i,j) \in (\bar{S}, S).
\]

24
Proof: By (4.1.2A) and since $x$ is a feasible $(s,t)$-flow,

$$v(x) = x(S,\overline{S}) - x(\overline{S},S) = \sum_{i \in S, j \in \overline{S}, (i,j) \in E} x(i,j) - \sum_{i \in \overline{S}, j \in S, (i',j') \in E} x(i',j')$$

$$\leq \sum_{i \in S, j \in \overline{S}, (i,j) \in E} x(i,j) \leq \sum_{i \in S, j \in \overline{S}, (i,j) \in E} u(i,j) = u(S,\overline{S}).$$

(4.1.4) Let $G(V,E)$ be a network with a capacity function $u$ and let $x$ be a feasible flow of $G$. The residual network $G(x)$ is a network with the same vertices and same arcs as $G$, where the residual capacity $r_{ij} = u_{ij} - x_{ij} + x_{ji}$, $\forall (i,j) \in E$. Note that $r_{ij}$ has two components:

(i) $u_{ij} - x_{ij}$, the unused capacity of the arc $(i,j)$;

(ii) the current flow $x_{ji}$ on arc $(j,i)$ which can be cancelled to increase the flow from vertex $i$ to vertex $j$.

2. Maximum $(s,t)$-Flow and Minimum $(s,t)$-Cut Problems.

(4.2.1) The maximum $(s,t)$-flow problem is to find a feasible $(s,t)$-flow for which $v(x)$ is maximum. Such a maximized $x$ is called a maximum $(s,t)$-flow. The minimum $(s,t)$-cut problem is to find an $(s,t)$-cut $(S,\overline{S})$ for which $u(S,\overline{S})$ is minimum. Such a minimized $(S,\overline{S})$ is called a minimum $(s,t)$-cut.

(4.2.2) (This follows from (4.1.3)(i).) If $x^*$ is a maximum $(s,t)$-flow and $(S^*,\overline{S}^*)$ is a minimum $(s,t)$-cut, then

$$v(x^*) \leq u(S^*,\overline{S}^*).$$

(4.2.3) Let $x$ be an $(s,t)$-flow. Let $v \in V(G)$. An undirected $(s,v)$-path $s = v_0e_1v_1e_2 \cdots e_kv_k = v$ in the corresponding residual graph of $x$ is called an $(s,v)$-flow augmenting path (or just an $(s,v)$-fap) if

(i) for each forward arc $e_i$, ($e_i$ is directed in $G$ from $v_{i-1}$ to $v_i$), $x(v_{i-1},v_i) < u(v_{i-1},v_i)$;

(ii) for each backward arc $e_i$, ($e_i$ is directed in $G$ from $v_i$ to $v_{i-1}$), $x(v_{i-1},v_i) > 0$.

An $(s,t)$-fap in the corresponding residual graph of $x$ is also called an $(s,t)$-fap with respect to $x$ in $G$.

(4.2.4) A feasible $(s,t)$-flow $x$ is maximum if and only if there is no $(s,t)$-fap.

Proof: (Only If) If there is an $(s,t)$-fap, then we can increase the amount of flow along this fap and so $x$ is not maximum.

(If) Let $S = \{v \in V(G) : G$ has an $(s,v)$-fap$\}$. Then $s \in S$ and, as $G$ has no $(s,t)$-fap,
\( t \in \bar{S} \). By the definition of \( S \), for any \((u, v) \in \delta(S)\), we must have \( x(u, v) = u(u, v) \); for any \((u, v) \in \delta(\bar{S})\), we must have \( x(u, v) = 0 \), as otherwise \( v \in S \), contrary to the fact that \( v \in \bar{S} \). Therefore, by (4.1.2A)
\[
v(x) = x(S, \bar{S}) - x(\bar{S}, S) = u(S, \bar{S}) - 0 = u(S, \bar{S}) .
\]

By (4.2.2), \( x \) must be maximum.

(4.2.5) (The Idea of a Flow Augmentation Algorithm)

Input: A network \( G \), a non-negative capacity \( u \),
two vertices \( s, t \in V(G) \).

Output: A maximum \((s, t)\)-flow \( x^* \)
and a minimum \((s, t)\)-cut \((S^*, \bar{S}^*)\).

Initialization: \( x := 0, S := \{s\} \) (as defined in the proof of (4.2.4)).

Iteration:
Step 1 Find an \((s, t)\)-fap with respect to the current flow \( x \).
If none exists, STOP (the current flow is maximum).

Step 2 Augment the current flow along the found \((s, t)\)-fap.
Update the current flow and \( S \) and GOTO Step 1.

(4.2.5A) Using residual networks (4.1.4), one can develop a labelling algorithm based on
the idea in (4.2.5):

Input: A network \( G(V, E) \) with a capacity function, \( s, t \in V \).

Output: a maximum \((s, t)\) flow and a minimum \((s, t)\)-cut.

Initialization: label \( t \).

Iteration: WHILE \( t \) is labelled, DO
BEGIN unlabel all vertices;
set \( \text{pred}(j) := 0 \), for each \( j \in V \);
label \( s \) and set \( \text{LIST}: = \{s\} \);
WHILE \( \text{LIST} \neq \emptyset \) or \( t \) is unlabelled, DO
BEGIN remove a vertex \( i \) from \( \text{LIST} \);
FOR each arc \((i, j)\) in the residual network, DO
IF \( r_{ij} > 0 \) and vertex \( j \) is unlabelled,
THEN set \( \text{pred}(j) := i \), label vertex \( j \),
and set \( \text{LIST} := \text{LIST} \cup \{j\} \);
END;
IF \( t \) is labelled, THEN argument
END;
PROCEDURE argument
BEGIN
use the predecessor labels to track back from \( t \) to \( s \) to find an augmenting \((s,t)\)-path \( P \);
\( \delta := \min \{ r_{ij} : (i, j) \in P \} \);
augment \( \delta \) units of flow along \( P \) and update the residual capacities;
END

(4.2.5B) At termination, Algorithm (4.2.5A) will output a maximum \((s,t)\)-flow.

Proof: Note the when the Algorithm terminates, \( G \) will have no \((s,t)\)-fap, and so this follows by (4.2.4).

(4.2.6) (Max Flow-Min Cut Theorem)
Let \( G \) be a digraph with a capacity weight function \( u \geq 0 \). Let \( s, t \in V(G) \) be two distinct vertices. If \( \mathbf{x}^* \) is a maximum \((s,t)\)-flow and \((S^*, \tilde{S}^*)\) is a minimum \((s,t)\)-cut, then
\[
v(\mathbf{x}^*) = u(S^*, \tilde{S}^*).
\]

Proof: By (4.2.2) and (4.2.5).
\( (4.2.7) \) (Ford-Fulkerson Labelling Algorithm)

**Input**  
A network \( G \), a non-negative capacity \( u \),  
two vertices \( s, t \in V(G) \).

**Output**  
A maximum \((s,t)\)-flow \( \vec{x}^* \) and a minimum \((s,t)\)-cut \((S^*, \bar{S}^*)\).

**Initialization:**  
\( l(s) : -\infty, S := \{s\} \) (the labelled vertices).

**Iteration:**  

\textbf{Step 1}  
[Optimality Test]  
IF all labelled vertices have been scanned, THEN  
\( \vec{x} \) is maximum, and \((S, V - S)\) is a minimum \((s,t)\)-cut.  
ELSE choose a unscanned vertex \( i \) with label \((h^\pm, \epsilon)\),  
GOTO Step 2.

\textbf{Step 2}  
[Scanning]  
FOR any unlabelled vertex \( j \) such that \((i,j) \in E\) and \( x(i,j) < u(i,j) \),  
give \( j \) the label \((i^+, \min\{\epsilon, u(i,j) - x(i,j)\})\).  
FOR any unlabelled vertex \( j \) such that \((j,i) \in E\) and \( x(j,i) > 0 \),  
give \( j \) the label \((i^-, \min\{\epsilon, x(j,i)\})\).  
(this completes the scanning of \( j \).)  
IF \( t \) has a label, THEN GOTO Step 3, ELSE GOTO Step 1.

\textbf{Step 3}  
[Breakthrough]  
IF \( t \) has label \((j^\pm, \epsilon)\), THEN DO  
\( h := t \), WHILE \( h \neq s \), DO  
IF \( h \) has label \((k^+, \epsilon)\), THEN  
\( x(k,h) := x(k,h) + \epsilon \) and \( h := k \),  
ELSE \( (h \) has label \((k^-, \epsilon)\),)  
\( x(h,k) := x(h,k) - \epsilon \) and \( h := k \).  
GOTO Step 1.

\( (4.2.8) \) Suppose that in the Maximum Flow problem, the capacity function \( u : E \mapsto \mathbb{Z} \) is integer valued. Then Algorithms (4.2.6) and (4.2.7) will terminate in a finite number of steps, and output a maximum flow \( x \). (This remains valid if \( u \) is rational valued.)

**Proof:** Suppose that \( u \) is integral. If we start with an integral feasible flow (such as \( x = 0 \)),  
then in each iteration step, the amount of augmentation is also an integer. Moreover, after  
each iteration, \( x \) remains as an integer valued function and \( v(x) \) is increased by an integral  
amount. Since \( u \) is integer valued, the algorithm will stop in at most  
\[ \max\{u(e) : e \in E(G)\}. \]

number of iterations. At termination of the algorithm, \( x \) will be an integer valued flow \( G \)  
has no \((s,t)\)-faps with respect to \( x \). By (4.2.4), the outputted \( x \) must be a maximum flow.
Suppose that \( u \) is rational valued. Since \( G \) is a finite net, there exists an integer \( d > 0 \) such that for any \( e \in E(G) \), \( d \cdot u(e) \in \mathbb{Z} \), and so we can apply the result on integer valued maximum flow problem to the net \((G, d \cdot u)\).

3. Improvement of a Slow Case.

(4.3.1) A slow example.

(4.3.2) (Edmonds and Karp Theorem)
If each flow augmentation is made along a fap containing the minimum possible number of arcs, then a maximum flow is obtained after at most \( \frac{mn}{2} \leq \frac{n^3 - n^2}{2} \) augmentations, where \( m = |E| \) and \( n = |V| \).

**Remark:** If we viewed the weight of the edges in the residual net is the constant function \( w \equiv 1 \), then here we are looking for a shortest fap.

(4.3.3) Let
\[
\sigma_i^{(k+1)} = \text{the minimum number of arcs in an } (s, i)\text{-fap after } k \text{ flow augmentations},
\]
\[
\tau_i^{(k+1)} = \text{the minimum number of arcs in an } (i, t)\text{-fap after } k \text{ flow augmentations}.
\]
If each flow augmentation is made along a fap with a minimum number of arcs, then
\[
\sigma_i^{(k+1)} \geq \sigma_i^{(k)}, \quad \tau_i^{(k+1)} \geq \tau_i^{(k)}.
\]

**Proof:** Suppose \( \sigma_i^{(k+1)} < \sigma_i^{(k)} \), for some \( i \) and \( k \). Choose \( i \) and \( k \) such that
\[
\sigma_i^{(k+1)} = \min_j \{ \sigma_j^{(k+1)} \mid \sigma_j^{(k+1)} < \sigma_i^{(k)} \}.
\]
Then \( \sigma_i^{(k+1)} \geq 1 \) (only \( \sigma_i^{(k+1)} = 0 \)). Let \((j, i)\) or \((i, j)\) be the final arc in a shortest \((s, i)\)-fap after the \((k+1)\)st flow augmentation.

If \((j, i)\) is a forward arc with \( u(j, i) > x(j, i) \), then \( \sigma_i^{(k+1)} = \sigma_j^{(k+1)} + 1 \) and by the choice of \( \sigma_i^{(k+1)} \), \( \sigma_j^{(k+1)} \geq \sigma_j^{(k)} \) + 1. Therefore, \( x(j, i) = u(j, i) \) after the \( k \)th augmentation since otherwise \( \sigma_i^{(k)} \leq \sigma_j^{(k)} + 1 \leq \sigma_i^{(k+1)} \), a contradiction.

But if \( x(j, i) = u(j, i) \) after \( k \)th augmentation, and \( x(j, i) < u(j, i) \) after \((k+1)\)st augmentation, it follows that \((j, i)\) is a backward arc in the \((k+1)\)st fap. Since that fap contains the minimum number of arcs, \( \sigma_j^{(k)} = \sigma_i^{(k)} + 1 \), and so
\[
\sigma_i^{(k)} + 2 = \sigma_j^{(k)} + 1 \leq \sigma_i^{(k+1)},
\]
a contradiction.
The case when \((j, i)\) is a backward arc with \(x(j, i) > 0\) can be proved similarly.

The proof for \(\tau_i^{(k+1)} \geq \tau_i^{(k)}\) is also similar.

(Proof for (4.3.2)) Each time an augmentation is made, at least one arc is critical in the sense that it limits the amount of augmentation possible. The flow through a critical arc is either raised to capacity (forward arc) or reduced to zero (backward arc).

Suppose that \((i, j)\) is a critical arc in the \((k + 1)\)st fap. The number of arcs in this path is \(\sigma_i^{(k)} + \tau_i^{(k)} = \sigma_j^{(k)} + \tau_j^{(k)}\). The next time \((i, j)\) appears in a fap, (say the \((l + 1)\)st), it will be with the opposite direction (forward in the \((k + 1)\)st and backward in the \((l + 1)\)st or vice versa).

Assume that \((i, j)\) is forward in the \((k + 1)\)st path. Then

\[
\sigma_j^{(k)} = \sigma_i^{(k)} + 1 \quad \text{and} \quad \sigma_i^{(l)} = \sigma_j^{(l)} + 1.
\]

By (3.3), \(\sigma_j^{(l)} \geq \sigma_j^{(k)}\) and \(\tau_i^{(l)} \geq \tau_i^{(k)}\), and so

\[
\sigma_i^{(k)} + \tau_i^{(l)} = \sigma_j^{(l)} + 1 + \tau_i^{(l)} \geq \sigma_j^{(k)} + 1 + \tau_i^{(l)} = \sigma_i^{(k)} + 1 + \tau_i^{(k)}.
\]

Thus each succeeding fap in which \((i, j)\) is critical at least two arcs longer than the preceding one.

Since every fap has length at most \(n - 1\), no arc may be critical for more than \(n/2\) times. But each fap has at least one critical arc, and there are \(m \leq n^2 - n\) distinct arcs. Hence there can be at most \(\frac{mn}{2} \leq \frac{n^3 - n^2}{2}\) augmentations.


(4.4.1) Let \(G\) be a digraph. A circulation (also called a flow) in \(G\) is a real valued vector indexed by \(E\)

\[
x = (x(i, j) : (i, j) \in E(D)),
\]

satisfying

(4.4.1A) \(x(V - \{j\}, \{j\}) - x(\{j\}, V - \{j\}) = 0, \forall j \in V(G)\).

(4.4.2) Let \(G(V, E)\) be a digraph with two weight (capacity) functions \(l, u\) from \(E\) to the reals such that \(\forall (i, j) \in E, l(i, j) \leq u(i, j)\). A circulation \(x\) is feasible (in the net \(*((G, l, u))\)) if

\[
l(i, j) \leq x(i, j) \leq u(i, j), \forall (i, j) \in E.
\]

(4.4.3) The Circulation Problem: Given a net \((G, l, u)\), does there exist a feasible circulation?
(4.4.4) If a feasible circulation exists, then $\forall S \subset V(G)$, $l(S, \bar{S}) \leq u(S, \bar{S})$.

(4.4.5) **(Hoffman's Theorem)** There is a feasible circulation if and only if $\forall S \subset V$, $l(S, \bar{S}) \leq u(S, \bar{S})$.

**Proof:** Construct a new digraph $G'$ from $G$ by adding two new vertices $s$ and $t$ and by adding new arcs $(s, i), (i, t), \forall i \in V$. Define a capacity function for $G'$ as follows:

$$u'(i, j) = \begin{cases} 
    l(V - \{j\}, \{j\}) & \text{if } i = s \\
    l(\{i\}, V - \{i\}) & \text{if } j = t \\
    u(i, j) - l(i, j) & \text{if } i \neq s \text{ and } j \neq t
\end{cases}$$

Then solve the maximum $(s, t)$-flow problem for $G'$ with capacity $u'$. Let $x'$ be a maximum flow obtained.

**Case 1** $v(x') = u'(\{s\}, V \cup \{t\}) = l(V, V)$

The define $x$ by $x(i, j) = l(i, j) + x'(i, j)$. By (4.1.3), $\forall i \in V$,

$$x'(s, i) = l(V - \{i\}, \{i\}) \text{ and } x'(i, t) = l(\{i\}, V - \{i\})$$

Since $x'$ is a feasible $(s, t)$-flow of $G'$,

$$x'(\{i\}, V - \{i\}) - x'(V - \{i\}, \{i\}) = 0, \forall i \in V,$$

and so

$$x(\{i\}, V - \{i\}) - x(V - \{i\}, \{i\}) = l(\{i\}, V - \{i\}) + x'(\{i\}, V - \{i\}) - l(V - \{i\}, \{i\}) - x'(V - \{i\}, \{i\})$$

Thus $x$ is a circulation of $G$.

**Case 2** $v(x') < u'(\{s\}, V \cup \{t\}) = l(V, V)$

Let $(S, \bar{S})$ be a minimum $(s, t)$-cut (with respect to $DG'$ and $u'$) and let $X = \bar{S} - \{t\}$ and so $\bar{X} = S - \{s\}$. Then

$$u'(S, \bar{S}) = u'(\{s\}, V \cup \{t\}) + u'(X, \bar{X}) + u'(\bar{X}, \{t\})$$

$$= l(V, X) + u(\bar{X}, X) - l(\bar{X}, X) + l(\bar{X}, \{t\})$$

Since $u'(S, \bar{S}) < l(V, V)$,

$$u(X, \bar{X}) < l(V, V) - l(V, X) - l(\bar{X}, V) + l(\bar{X}, X)$$

$$= l(X, \bar{X}).$$

31
Thus by (4.4.4), no circulation can exist and the subset $X$ violates the condition of the theorem.

(4.4.6) Based on the proof of (4.4.5), find an algorithm that for any given $(G, l, u)$, either outputs a circulation or finds a subset $X$ that violates the necessary condition (4.4.4).

(4.4.7) (**A special case of the flow feasibility problem**) Let $G = (V, E)$ be a digraph, $u : E \mapsto \mathbb{R}^+$ and $b : V \mapsto \mathbb{R}$ be two functions. We want to find a function $x : E \mapsto \mathbb{R}$ such that

(i) $\forall v \in V, \partial x(v) = b(v)$.
(ii) $0 \leq x \leq u$ (that is, $x(e) \leq u(e), \forall e \in E$).

(4.4.8) Suppose that $\exists x$ satisfying $\partial x = b$, then

$$b(V) : = \sum_{v \in V} b(v) = \sum_{v \in V} \partial x(v)$$

$$= \sum_{v \in V} \left[ \sum_{(u, v) \in E} x(u, v) - \sum_{(v, w) \in E} x(v, w) \right]$$

$$= \sum_{(i, j) \in E} x(i, j) - \sum_{(i', j') \in E} x(i', j') = 0$$

(4.4.9) The problem (4.4.7) has a solution if and only if each of the following holds.

(i) $b(V) = 0$.
(ii) $\forall A \subseteq V, b(A) \leq u(\delta(A))$.

Moreover, when both $b$ and $u$ are integral, then (4.4.7) has an integral solution $x$ if and only if (4.4.9)(i) and (ii) hold.

**Proof:** (**Necessity**) As $x$ must satisfy (4.4.7)(i) and (ii), $\forall A \subseteq V$,

$$b(A) = \partial x(A) = \sum_{v \in A} \left[ \sum_{(u, v) \in E} x(u, v) - \sum_{(v, w) \in E} x(v, w) \right] \leq \sum_{v \in A, u \in \delta(A), (u, v) \in E} x(u, v) \leq u(\delta(A)).$$

**Sufficiency** Construct a new digraph $G'$ with $V(G') = V(G) \cup \{s, t\}$, where $\{s, t\} \cap V(G) = \emptyset$.

**Edges of $G'$:** For each $v$ with $b(v) < 0$, there is an arc $(s, v) \in E(G')$; and for each $w$ with $b(w) > 0$, there is an arc $(w, t) \in E(G')$.

**Capacity $u'$ of $G'$:** For each old edge $e \in E(G)$, $u'(e) = u(e)$; for each new edge $(s, v) \in E(G')$, $u'(s, v) = -b_v$, and for each new edge $(w, t) \in E(G')$, $u'(w, t) = b_v$. 

32
Note that the following are equivalent:

(A) (4.4.7) has a solution if and only if \((G', u')\) has an \((s, t)\)-flow \(x\) with

\[
v(x) = \sum_{v \in V, b(v) > 0} b(v) = u'((V - \{t\}, \{t\})).
\]

(B) In \(G'\), \((V - \{t\}, \{t\})\) is a minimum cut. (By the Max-Flow-Min-Cut Theorem).

(C) \(\forall A' \subseteq V\), in \(G'\),

\[
\sum_{v \in V, b(v) > 0} b(v) = b(\delta(\{t\})).
\]

(D) \(\forall A \subseteq V\), \(b(A) \leq u(\delta(A))\).

(Reason: As \(u'(\delta(A' \cup \{s\})) = \sum_{v \in \delta(A' \cup \{s\})} b(v) + u(\delta(A)) - \sum_{v \in \delta(A) - \{s\}} b(v) = \sum_{v \in \delta(A) - \{s\}} b(v) - u(\delta(A)) - b(A)\), therefore (C) \(\iff u(\delta(A)) - b(A) \geq 0\).

(4.4.10) **Flow Feasibility Problem** Given a bipartite graph \(G = (V, E)\) with a vertex bipartition \(\{V_1, V_2\}\), and vectors \(a : V_1 \mapsto \mathbb{Z}^+, b : V_2 \mapsto \mathbb{Z}^+\), find a vector \(x : E \mapsto \mathbb{Z}\) satisfying each of the following:

(i) for each \(i \in V_1\), \(x(\{i\}, V_2) \leq a(i)\),

(ii) for each \(j \in V_2\), \(x(V_1, \{j\}) = b(j)\),

(iii) for each \(e \in E\), \(x(e) \geq 0\).

(4.4.11) We can apply (4.4.9) to give another proof of (Hoffman’s Theorem), restated as follows.

Let \(G\) be a digraph, \(l : E \mapsto \mathbb{R} \cup \{-\infty\}\) and \(u : E \mapsto \mathbb{R} \cup \{\infty\}\) be two maps satisfying \(l \leq u\). Then a feasible circulation \(x\) exists if and only if

\[
\forall A \subseteq V, u(\delta(A)) - l(\delta(A)).
\]

Moreover, if both \(l\) and \(u\) are integral, then an integral feasible circulation exists if and only if the same condition above holds.

**Proof:** \(\forall A\), as \(x\) is a circulation, we have \(x(\delta(A)) = x(\delta(A))\), and so by the feasibility,

\[
u(\delta(A)) - x(\delta(A)) = l(\delta(A)).
\]

Let \(l' = 0\) and \(u' = u - l\). (Therefore, \(x' = x - l\). Note that the following are equivalent:

(A) \(x'\) is a feasible circulation.

(B) \(\partial x' = \partial x - \partial l = -\partial l\).

Consider Problem (4.4.7) with \(l' = 0\), \(u' = u - l\), and \(b = -\partial l\). Then by (4.4.9), a solution \(x'\) exists if and only if

\[
\forall A \subseteq V, (u - l)(\delta(A)) = -\partial l(A) = l(\delta(A)) - l(\delta(A)).
\]
This implies (by adding \( l(\delta(A)) \) both sides)

\[
\forall A \subseteq V, u(\delta(A)) \geq l(\delta(A)).
\]

When \( u \) and \( l \) are integral, \( u - l \) is also integral. Apply (4.4.9).

5. Supply-Demand Flows

(4.5.1) (The problem). Let \((G, u)\) be a net with a capacity \( u \). Suppose that \( V = S \cup T \cup R \) is a partition, where \( S \) is a set of \textit{supply vertices}, \( T \) a set of \textit{demand vertices} and \( R \) a set of \textit{transfer vertices}. Suppose also that each \( i \in S \) is associated with a \textit{possible maximum available supply} \( a(i) \), and each \( i \in T \) is associated with a \textit{positive demand} \( b(i) \). A \textit{supply-demand} flow is a solution \( x \) (a vector indexed by the arcs in \( E \)) to

\[
\begin{align*}
(4.5.1A) & \quad 0 \leq x(i, j) \leq u(i, j), \forall (i, j) \in E, \\
(4.5.1B) & \quad x(V - \{i\}, \{i\}) - x(\{i\}, V - \{i\}) \geq b(i), \forall i \in T, \\
(4.5.1C) & \quad x(V - \{i\}, \{i\}) - x(\{i\}, V - \{i\}) = 0, \forall i \in R, \\
(4.5.1D) & \quad x(V - \{i\}, \{i\}) - x(\{i\}, V - \{i\}) \leq a(i), \forall i \in S
\end{align*}
\]

(4.5.2) Notation: \( \forall X \subseteq V, a(X) = \sum_{i \in X} a(i), \) and \( b(X) = \sum_{i \in X} b(i) \).

(4.5.3) If there is a supply-demand flow, then

\[
b(T \cap X) - a(S \cap X) \leq u(\bar{X}, X), \forall X \subseteq V.
\]

**Proof.** In (4.5.1), (B) + (C) - (D) for vertices in \( X \), then apply (A).

(4.5.4) (Gale) The necessary condition is (4.5.3) is also sufficient. That is: There exists a supply-demand flow if and only if

\[
b(T \cap X) - a(S \cap X) \leq u(\bar{X}, X), \forall X \subseteq V.
\]

**Proof:** Reduce the problem to a circulation problem as follows: Add a new vertex \( v \), add new arcs \((v, i), \forall i \in S\), add new arcs \((i, v), \forall i \in T\). For each \((i, j) \in E\). Define

\[
l(i, j) = \begin{cases} 
0 & \text{if } (i, j) \in E \text{ or } i = v, j \in V \\
b(i) & \text{if } i \in V, j = v,
\end{cases} \quad \text{and } u(i, j) = \begin{cases} 
0 & \text{if } (i, j) \in E \\
a(i) & \text{if } i = v, j \in V \\
\infty & \text{if } i \in V, j = v.
\end{cases}
\]

Let the new network be \( G' \).
(4.5.5) Show that if \( G' \) in (4.5.4) has a feasible circulation, then \( G \) has a supply-demand flow.

**Proof of (4.5.4), continued.** Suppose that \( G' \) does not have a feasible circulation. Then by Hoffman’s Theorem (4.4.5), there exists a set \( X \subseteq V \cup \{v\} \) such that \( l(X, \bar{X}) > u(\bar{X}, X) \).

5pt **Case 1** Suppose \( v \in X \). Then \( T \subset X \), for otherwise \( u(\bar{X}, X) = \infty \). But this would mean \( l(i, j) = 0 \) for all \((i, j) \in (X, \bar{X})\), and so \( l(X, \bar{X}) = 0 \), a contradiction.

**Case 2.** Thus \( v \in \bar{X} \). Then \( l(X, \bar{X}) = b(X \cap T) \) and \( u(\bar{X}, X) = a(X \cap S) + u(\bar{X} - \{v\}, X) \), and so (4.5.3) is violated.

6. Flow Decomposition Theorems

(4.6.1) Let \( G(V, E) \) be a digraph. Let \( C \) be a dicircuit of \( D \). A **flow circuit** with support \( C \) is a circulation \( f \) such that \( x(i, j) \neq 0 \) if and only if \((i, j) \) is an arc of \( C \). For convenience, we also regard the zero vector \( 0 \) as a flow circuit.

(4.6.2) If \( f \) is a flow circuit with support \( C \), then \( \bar{f} \) is a constant on the arcs of \( C \). In other words, there is a constant \( \delta \) such that \( f(i, j) = \delta, \forall (i, j) \in E(C) \). (This constant \( \delta \) is denoted by \( |f| \).)

(4.6.3) Exercise: Let \( f_1, f_2, \ldots, f_k \) be flow circuits. Then \( \sum_{i=1}^{k} f_i \) is a circulation.

(4.6.4) A vector \( x \geq 0 \) indexed by \( E \) is a circulation of \( G(V, E) \) if and only if there are flow circuits \( f_1, f_2, \ldots, f_k \), such that \( x = \sum_{i=1}^{k} f_i \).

**Proof:** The “if” part is Exercise (4.6.3). We shall prove the “only if” part. For a vector \( x \geq 0 \) indexed by \( E \), define \( \text{support}(x) = \{ (i, j) \in E : x(i, j) > 0 \} \). Choose a counterexample circulation \( x \) such that \( |\text{support}(x)| \) is minimized.

Pick \((i, j) \in E \) such that \( x(i, j) > 0 \), and set \( v_0 = i \) and \( v_1 = j \). Since \( x \) is a circulation, there is some arc \((v_1, v_2) \) such that \( x(v_1, v_2) > 0 \). Continue this process until a dicircuit \( C \) is found, and \( x(i, j) > 0, \forall (i, j) \in E(C) \). Let \( \delta = \min\{x(i, j) : (i, j) \in E(C)\} \). Then \( \delta > 0 \). Construct a flow circuit \( f_1 \) with \( |f_1| = \delta \), and let \( x_1 = x - f_1 \). Then \( |\text{support}(x_1)| < |\text{support}(x)| \), and so by the minimality of \( x \), \( x_1 = \sum_{i=2}^{k} f_i \).

(4.6.5) Let \( G(V, E) \) be a network with distinct vertices \( s, t \in V \). Let \( P \) be an \((s, t)\)-dipath in \( G \). An \((s, t)\)-**flow path** with support \( P \) is a vector \( f \) indexed by \( E \) such that \( \text{support}(f) = E(P) \).

(4.6.5A) Exercise: Show that if \( f \) is an \((s, t)\)-flow path with support \( P \), then there is a
constant $\delta$ such that for any $(i, j) \in E(P)$, $f(i, j) = \delta$.

(4.6.5B) Exercise: Show that an $(s, t)$-flow can be expressed as the sum of a finite number of $(s, t)$-flow paths and flow circuits.

(4.6.5C) Exercise: Show that for any maximum $(s, t)$-flow problem, there is an optimal solution which is the sum of $(s, t)$-flow paths.

7. Some Applications

(4.7.1) Let $G$ be a graph, $M$ be a matching of $G$ and $W$ be a vertex cover of $G$. Then $|M| \leq |W|$.

Proof: Every edge $e \in M$ must be incident with a vertex in $v \in W$.

(4.7.2) (König’ Theorem).

Let $G = (V_1 \cup V_2, E)$ be a bipartite graph with bipartition $\{V_1, V_2\}$. If $M^*$ is a maximum cardinality matching and if $W^*$ is a minimum cardinality vertex cover, then $|M^*| = |W^*|$.

Proof: Construct a digraph $G'$ with $V(G') = V(G) \cup \{s, t\}$, where $\{s, t\} \cap V(G) = \emptyset$, and with $E(G') = (V_1, V_2) \cup \{\{s\}, V_1\} \cup (V_2, \{t\})$. Let the capacity of $G'$ be $u'$ such that $u'(i, j) = \begin{cases} 1 & \text{if } (i, j) \in (\{s\}, V_1) \cup (V_2, \{t\}) \\ \infty & \text{if } (i, j) \in (V_1, V_2) \end{cases}$.

Let $x$ be an integral maximum $(s, t)$-flow. Since $(\{s\}, V_1)$ and $(V_2, \{t\})$ are $(s, t)$-cuts, $x$ is $\{0, 1\}$-valued. Let

$$M = \{(i, j) \in (V_1, V_2) : x(i, j) = 1\}.$$ 

Suppose that $e', e''$ are both incident with the same vertex $v \in V_1$, then since $x$ is $\{0, 1\}$-valued,

$$x(V(G') - \{v\}, \{v\}) - x(\{v\}, V(G') - \{v\}) = 1 - x(\{v\}, V(G') - \{v\}) \leq 1 - 2 < 0,$$

contrary to the local balanced condition (4.1.1A). Thus $M$ must be a matching of $G$.

The Max-Flow-Min-Cut Algorithm also outputs a minimum cut in the form $\delta(\{s\} \cup A)$ for some $A \subseteq V_1 \cup V_2$. If

$$\exists e = (i, j) \in (V_1, V_2) \cap (A \cap V_1, A - V_2),$$

then $s, i, j, t$ is an $(s, t)$-fap, contrary to (4.2.4).

By Max-Flow-Min-Cut Theorem and (4.7.2), $|M^*| \geq |M| = |W| \geq |W^*| \geq |M^*|$.

(4.7.3) Let $G(V, E)$ be a digraph and let $s, t \in V$. A collection $P_1, P_2, \ldots, P_m$ of $(s, t)$-dipaths are internally disjoint if the only two vertices in common of and $P_i$ and $P_j$ are $s$
and \( t \).

(4.7.4) Let \( G \) be a digraph and let \( s, t \in V(G) \) be two distinct vertices. Suppose that \( G \) has \( h \) internally disjoint \((s, t)\)-dipaths, and \( G - \{s, t\} \) has a set \( S \) of \( k \) vertices whose removal breaks all \((s, t)\)-paths. Then \( h \leq k \).

**Proof:** Each of the \( h \) internally disjoint \((s, t)\)-dipath must contain at least one vertex in \( S \).

(4.7.5) (Menger's Theorem).

Let \( G \) be a digraph and let \( s, t \in V(G) \) be two distinct vertices. The maximum number of internally disjoint \((s, t)\)-paths is equal to the minimum number of vertices of \( V - \{s, t\} \) whose removal breaks all \((s, t)\)-paths.

8. Total Unimodularity

(4.8.1) An \( m \times n \) real matrix is **totally unimodular** (TU) if every square submatrix of \( A \) has determinant 0, 1, or -1. This implies that each entry of a TU \( A \) must be 0, 1, or -1.

(4.8.2) Let \( A \) be an \( m \times n \) TU and let \( I_m \) be an \( m \times m \) identity matrix. Then each of the following holds.

(i) \([AI_m]\) is TU;
(ii) \([AA]\) is TU;
(iii) If \( A_1 \) is obtained from \( A \) be reversing the signs of some rows and/or columns, then \( A_1 \) is TU;
(iv) \([I_nA]\) is TU.

(4.8.3) (Hoffman-Kruskal) Consider the following LP:

maximize \( c^t x \)

\[Ax \leq b\]

\(x \geq 0\)

where \( A \) is a fixed integral \( m \times n \) matrix. Then the following are equivalent:

(4.8.3A) \( A \) is TU.
(4.8.3B) For any integral \( b \) and \( c \) such that an optimal solution exists, there exists an integral optimal solution.

**Proof:** (A) implies (B): Suppose that \( c \) and \( b \) are such that there exists an optimal solution. Let \( A_1 = [AI_m] \). Then there exists an optimal feasible basis \( B \) to the integer program

\[37\]
maximize $\mathbf{c}^\top \mathbf{x}$

\[
A_1 \mathbf{X} \leq \mathbf{b}
\]

$\mathbf{X} \geq 0$

where $\mathbf{X}' = (\mathbf{x}^\top, \mathbf{s}^\top)$ and where $\mathbf{s}$ is a vector of slack variables. The basic solution $\mathbf{x}^*$ is defined by $\mathbf{x}^*_N = \mathbf{0}$ and $\mathbf{x}^*_B = B^{-1}\mathbf{b}$, where $N$ is the set of non-basic variables.

(4.8.4) If $A$ is the vertex-arc incidence matrix of a digraph, then $A$ is TU.

9. Additional Exercises

(4.9.1) **Dining Problem** Several families go out to dinner together. To increase their social interaction, they would like to sit at tables so that no two members of the same family are at the same table.

(4.9.1A) Show how to formulate finding a seating arrangement that meets this objective as a maximum flow problem. (To start, assume that the dinner contingent has $p$ families and that the $i$th family has $a(i)$ members. Also assume that $q$ tables are available and that the $j$th table has a seating capacity $b(j)$.)

(4.9.1B) Randomly generate a set of numbers $p, q, a(i), b(j)$ (each being an integer at least 3) and use your solution to (4.9.1A) to find a solution of your problem. Find a feasible seating arrangement.

(4.9.2) **Statistical Security of Data** The U.S. Census Bureau produces a variety of tables from its census data. Suppose that it wishes to produce a $p \times q$ table $D = \{d_{ij}\}$ of nonnegative integers. Let $r(i)$ denote the sum of the matrix elements in the $i$th row and let $c(j)$ denote the sum of the matrix elements in the $j$th column. Assume that each $r(i) > 0$ and each $c(j) > 0$. The Census Bureau often wishes to disclose all the row and column sums along with some matrix elements (denoted by a set $Y$) and yet suppress the remaining elements to ensure the confidentiality of privileged information. Unless with care, by disclosing the elements in $Y$, the Census Bureau might permit someone to deduce one or more suppressed elements. It is possible to deduce a suppressed element $d_{ij}$ if only one value of $d_{ij}$ is consistent with the row and column sums and the disclosed elements in $Y$. We say that any such suppressed elements are **unprotected**.

(4.9.2A) Describe a polynomial-time algorithm for identifying all the unprotected elements of the matrix and their values.

(4.9.2B) Randomly generate a $p \times q$ table $D$ with both $p \geq 4$ and $q \geq 4$. Disclose the row and column sums together with a set $Y$ with at least half of the matrix elements in $D$. Use your algorithm to find the unprotected elements and their values.
A commander is located at one vertex of \( p \) in a communication network (undirected) \( G \) and his subordinates are located at vertices denoted by the set \( S \). Let \( u_{ij} \) be the effort required to eliminate an edge \((i, j)\) from the network. The problem is to determine the minimum effort required to block all communications between the commander and his subordinates.

(4.9.3A) How can you solve this problem in polynomial time?

(4.9.3B) Randomly generate a network with at least 10 vertices and select a set of 4 vertices as the set \( S \) and a vertex \( p \not\in S \). Apply your solution to (4.9.3A) to find a solution to (4.9.3B).

(4.9.5) **The Minimum Flow Problem** In the minimum flow problem, we assume that each arc has a nonnegative lower bound on arc flows, and wish to send the minimum amount of flow from the source to the sink, while satisfying the lower and upper bounds on arc flows.

(4.9.5A) Show how to solve the MFP by using two applications of any maximum flow algorithm that applies to problems with zero lower bounds on arc flows. (Hint: First construct a feasible flow and then convert it into a minimum flow.)

(4.9.5B) Prove the following **min-flow max-cut theorem**. Let the floor of an \((s, t)\)-cut \([S, \bar{S}]\) be defined as \( \sum_{(i,j)\in[S,\bar{S}]} l_{ij} - \sum_{(i,j)\in[\bar{S},S]} u_{ij} \). Show that the minimum value of all flows from \( s \) to \( t \) equals the maximum floor of all \((s, t)\)-cuts.

(4.9.6) Show how to transform a maximum-flow problem having several source vertices and several sink vertices to one with only one source vertex and one sink vertex.

(4.9.7) Suppose that you want to solve a maximum flow problem containing parallel arcs, but the maximum flow code you own cannot handle parallel arcs. How would you use your current code to solve your maximum flow problem?

(4.9.8) Show that if \( x_{ij} = u_{ij} \) for some arc \((i, j)\) in every maximum flow, this arc must be a forward arc in some minimum cut.

(4.9.9) An engineering department consisting \( p \) faculty members, \( F_1, \ldots, F_p \), will offer \( p \) courses \( C_1, \ldots, C_p \), in the coming semester and each faculty member will teach exactly one course. Each faculty ranks two courses he (or she) would teach, ranking them according to his (or her) preference.

(4.9.9A) We say that a course assignment is **feasible** if every faculty member teaches a course in his (or her) list of preference. How would you determine whether the department can find a feasible assignment?

(4.9.9B) A feasible assignment is \( k \)-**feasible** if it assigns at most \( k \) faculty members to
their second most preferred courses. For a given $k$, suggest an algorithm for determine a $k$-feasible assignment.

(4.9.9C) A feasible assignment if optimal if it maximizes the number of faculty members assigned to their preferred course. Suggest an algorithm for determining an optimal assignment and analyze its complexity.

(4.9.10) **Airline Schedule Problem** An airline has $p$ flight legs that it wishes to service by the fewest possible planes. To do so, it must determine the most efficient way to combine these legs into flight schedule. The starting time for flight $i$ is $a_i$ and the finishing time is $b_i$. The plane requires $r_{ij}$ hours to return to the point of destination of flight $i$ to the point of origin of flight $j$. Suggest a method solving this problem.

(4.9.11) A flow $x$ is even if for every arc $(i, j) \in E$, $x_{ij}$ is even; $x$ is odd if for every arc $(i, j) \in E$, $x_{ij}$ is odd. Prove or disprove the following:

(4.9.11A) If all arc capacities are even, the network has an even maximum flow.

(4.9.11B) If all arc capacities are odd, the network has an odd maximum flow.

(4.9.12) Which of the following claims are true and which are false? Justify your answers by giving a proof or by construct a counterexample.

(i) If $x$ is a maximum flow, then either $x_{ij} = 0$ or $x_{ji} = 0$ for every arc $(i, j) \in E$.

(ii) Any network always has a maximum flow $x$ for which, for every arc $(i, j) \in E$, either $x_{ij} = 0$ or $x_{ji} = 0$.

(iii) If all arcs in a network have different capacities, the network has a unique maximum flow.

(iv) If we replace every direct arc by an undirected edge, the maximum flow value remains unchanged.

(v) If we multiply each arc capacity by a positive constant $\lambda$, the minimum cut remains unchanged.

(vi) If we add a positive number $\lambda$ to each arc capacity, the minimum cut remains unchanged.
5 Minimum Cost Flows

1. The Problem and Terminology

(5.1.1) Let $G = (V, E)$ be a digraph with a non-negative capacity function $u : \rightarrow \mathbb{R}$, and a real valued cost function $c : E \rightarrow \mathbb{R}$. Both $u$ and $c$ can be viewed as vectors $u$ and $c$ indexed by $E$. Let $s, t \in V$, and let $x$ be an $(s, t)$-flow (also viewed as a vector $x$). The cost of the flow $x$ is the dot product

$$c \cdot x = \sum_{e \in E} c(e)x(e).$$

The Minimum Cost Flow Problem seeks a feasible $(s, t)$-flow $x$ whose cost is minimized.

(5.1.2) For a fixed $(s, t)$-flow $x$, an $x$-augmenting path (also called an $x$-incrementing path) $P = v_0e_1v_1e_2v_2 \cdots e_kv_k$ satisfies:

(5.1.2A) $v_i \in V$ and $e_j \in E$;

(5.1.2B) either $e_j = (v_{j-1}, v_j)$ and $x(e_j) < u(e_j)$ ($e_j$ is a forward arc), or $e_j = (v_j, v_{j-1})$ and $x(e_j) > 0$ ($e_j$ is a backward arc).

When $v_0 = v_k$, the above gives the definition for an $x$-augmenting circuit (also called an $x$-incrementing circuit).

(5.1.3) If $C$ is an $x$-augmenting circuit, then define

$$\epsilon(C) = \min\{\{u(e) - x(e)\mid e \text{ is forward}\}, \{x(e)\mid e \text{ is backward}\}\},$$

and define the cost of $C$:

$$c(C) = \sum_{e \text{ is forward in } C} c(e) - \sum_{e \text{ is backward in } C} c(e).$$

$C$ is a negative cost augmenting circuit if $c(C) < 0$.

(5.1.4) Given a network $(G, u, c)$ with capacity $u$ and cost $c$, and a feasible flow $x$, the residual network (also refereed as the auxiliary graph) $G(x)$ can be defined as follows. Edges of $G(x)$: We replace each arc $(i, j)$ by two arcs $(i, j)$ and $(j, i)$.

Residual Capacity: For each $(i, j) \in E(G)$,

$$u'(i, j) = u(i, j) - x(i, j) \text{ and } u'(j, i) = x(i, j)$$

Residual Cost: For each arc $(i, j) \in E(G)$,

$$c'(i, j) = c(i, j) \text{ and } c'(j, i) = -c(i, j).$$

41
Note that $G(x)$ consists only of arcs with positive residual capacity.

2. Flow Augmentation Algorithm

(5.2.1) If an $(s, t)$-flow $x$ with $v(x) = v$ is a minimum cost flow with value $v$, then $x$ has no negative cost augmenting circuits.

**Proof** Suppose that $x$ has a negative cost circuit $C$ (using notation in (5.1.2)). Let $\epsilon = \epsilon(C) > 0$. Define $x_1 : E \mapsto \mathbb{R}$ (also viewed as a vector $x_1$) as follows.

$$x_1(e) = \begin{cases} x(e) + \epsilon & \text{if } e \in E(C) \text{ is forward} \\ x(e) - \epsilon & \text{if } e \in E(C) \text{ is backward} \\ x(e) & \text{otherwise} \end{cases}$$

Then

$$c \cdot x_1 = c \cdot x + \epsilon(C)c(C).$$

Since $c(C) < 0$, $x$ is not minimum.

(5.2.2) If an $(s, t)$-flow $x$ with $v(x) = v$ has no negative cost augmenting circuits, then $x$ is a minimum cost flow with value $v$.

**Proof** Assume that $x$ is not minimum, and so there is a minimum cost $(s, t)$-flow $x_1$ with

$$c \cdot x_1 < c \cdot x \text{ and } v(x_1) = v(x) = v.$$

Define a new network $G_2$ (same underlying graph structure as $G$) with a new capacity $c_2$, and a new flow $x_2$ as follows:

- for $x_1(v_i, v_j) > x(v_i, v_j)$, set $x_2(v_i, v_j) = x_1(v_i, v_j) - x(v_i, v_j)$ and $c_2(v_i, v_j) = c(v_i, v_j)$;
- for $x_1(v_i, v_j) < x(v_i, v_j)$, set $x_2(v_j, v_i) = x(v_i, v_j) - x_1(v_i, v_j)$ and $c_2(v_j, v_i) = -c(v_i, v_j)$.

Note that $x_2 \geq 0$, and

$$c_2 \cdot x_2 = c \cdot x_1 - c \cdot x,$$

and that $x_2$ is a circulation in $G_2$, since $x$ and $x_1$ are flows of the same value. Therefore, $x_2$ is the sum of flow circuits:

$$x_2 = \sum_{i=1}^{k} f_i.$$

Let the dicircuit $C_i$ in $G_2$ be the support of $f_i$. For $(v_i, v_j) \in E(C_i)$, $x_2(v_i, v_j) > 0$ and so either $x_1(v_i, v_j) > x(v_i, v_j)$ in which case $x(v_i, v_j) < u(v_i, v_j)$, or $x_1(v_i, v_j) < x(v_i, v_j)$ in which case $x(v_j, v_i) > 0$. Let $D_i$ be the circuit in $G$ corresponding $C_i$. Then $D_i$ is an $x$-augmenting circuit.
Note that \( c_2(C_i) = c(D_i) \). Since \( c_2 \cdot x_2 < 0 \), there is a circuit flow \( f_j \) with \( c_2(C_j) < 0 \), and so \( x \) has a negative augmenting circuit.

(5.2.3) **An idea of an algorithm:** (5.2.1) and (5.2.2) together provide a criterion to determine if a given flow is minimum or not, with a fixed value, by checking the existence of negative augmenting circuits. To see if an arc \((v, v')\) with \( x(v, v') < u(v, v') \) is in a negative cost circuit (as a forward arc), it suffices to find a minimum cost augmenting \((v', v)\)-path \( P \). Define \( \epsilon(P) \) and \( c(P) \) as in (5.1.3).

(5.2.4) Let \( x \) be a minimum cost flow of value \( v \) and let \( P \) be a cheapest \((s, t)\)-flow augmenting path (one with minimum cost). Let \( \delta \leq \epsilon(P) \). then augmenting along \( P \) by \( \delta \) results in a min cost flow with value \( v + \delta \).

**Sketch of Proof.** Let \( x_1 \) denote the resulting flow of value \( v + \delta \). We need to show that it is min cost. If \( x_1 \) is not minimum cost, then by (5.2.2), \( x_1 \) has a negative augmenting circuit \( C \). By (5.2.1), \( C \) cannot be an augmenting circuit of \( x \) and so \( C \) must use arcs in \( P \) in the opposite direction. the remaining arcs of \( P \) and \( C \) decompose into an augmenting path \( P_1 \) and augmenting circuits of \( x \), with all circuits positive by (5.2.1). Thus \( c(P_1) < c(P) \).

(5.2.5) **Rough Outline of Busacher-Goven Augmentation Algorithm**

1. Form a feasible flow of value at most \( v \) (e.g. the zero flow).
2. Search for the negative cost augmenting circuits.
3. IF such an augmenting circuit \( C \) is found, THEN augment around it by \( \epsilon(C) \) and return to Step 2.
   ELSE GOTO Step 2.
4. IF the flow value is \( v \), STOP (current flow is minimum of value \( v \)).
5. ELSE search for a cheapest \((s, t)\)-flow augmenting path \( P \).
   IF none, STOP (current flow is a maximum flow).
   ELSE augment along the path \( P \) as much as possible subject to the condition that the value of the flow does not exceed \( v \). GOTO Step 4.

3. **Edmonds and Karp Variation**

(5.3.1) **Updating of the cost.** Let \( T \) be a cheapest \((s, t)\)-flow augmenting path tree, let \( d_v \) denote the cost of a cheapest augmenting path from \( s \) to a vertex \( v \). Let

\[
c_1(v_i, v_j) := d_{v_i} + c(v_i, v_j) - d_{v_j}.
\]

Thus the new cost \( c_1 \) on \((v_i, v_j)\) is updated by the right side expression, where the \( c(v_i, v_j) \) (on the right side) denotes the old cost.
(5.3.2) With this updating, each of the following holds:

(i) If \( x(v_i, v_j) < u(v_i, v_j) \), then \( c_1(v_i, v_j) \geq 0 \). If \( x(v_i, v_j) > 0 \), then \( c_1(v_i, v_j) \leq 0 \).

(ii) For any augmenting \((s, t)\)-path \( P \), \( c_1(P) = c(P) + d_s - d_t \).

(iii) For any augmenting circuit \( C \), \( c_1(C) = c(C) \).

**Proof.** (i) Note \( d_{v_j} \leq d_{v_i} + c(v_i, v_j) \) (if \((v_i, v_j)\) is a potential forward arc) and \( d_{v_i} \leq d_{v_j} - c(v_i, v_j) \) (if \((v_i, v_j)\) is a potential backward arc). In terms of the new cost, we have \( c_1(v_i, v_j) \geq 0 \) in the former case and \( c_1(v_i, v_j) \leq 0 \) in the latter case, by (5.3.1).

(ii) Let \( P = v_0 c_1 v_1 c_2 v_2 \cdots c_k v_k \) be an augmenting path. By (3.1), if \( e_j = (v_{j-1}, v_j) \) is forward, then the new cost \( c_1(e_j) = d_{v_{j-1}} + c(v_{j-1}, v_j) - d_{v_j} \); if \( e_j = (v_j, v_{j-1}) \) is backward, then the new cost \( c_1(e_j) = - (d_{v_j} - d_{v_{j-1}}) + c(v_{j-1}, v_j) \). Thus when computing \( c_1(P) \), the terms \( d_{v_j} \) cancel in pairs except \( d_s \) and \( d_t \).

(iii) Similar to that for (ii).

(5.3.3) **Edmonds-Karp Augmentation Algorithm**

Steps 1-4 are the same as in the Busacher-Gowen Algorithm.

5. Search for a cheapest augmenting path tree \( T \), assign vertex labels \( d_v \).

   IF \( t \in T \), STOP (current flow is maximum).

   ELSE augment along the \((s, t)\)-flow augmenting path in \( T \), as much as possible subject to the condition that the value of the flow does not exceed \( v \).

6. IF the flow has value \( v \), STOP (current flow is a min cost flow of value \( v \)).

7. Reset costs: \( c(v_i, v_j) := d_{v_i} + c(v_i, v_j) - d_{v_j} \) for \((v_i, v_j)\) \( \in E \).

   GOTO to Step 5.

(5.3.4) **Proof of (5.2.4)** Consider the new cost \( c_1(C) \) for any augmenting circuit \( C \) after augmentation along \( P \) by \( \delta \). For arcs unaffected by the augmentation (that is arcs in \( E - E(P) \)), \( c_1(e) \geq 0 \) for a forward arc \( e \), and \( c_1(e') \leq 0 \) for a backward arc \( e' \), by (5.3.2)(i).

For all \( e \in E(P) \), \( c_1(e) = 0 \), since \( P \) is in the cheapest path tree. Thus the new cost \( c_1(C) \geq 0 \), by (5.3.2)(iii). Thus the augmenting circuit cannot be negative and so the flow \( x_1 \) obtained by augmenting along \( P \) by \( \delta \) is a min cost flow of value \( v + \delta \), by (5.2.2).

4. Additional Exercises

(5.4.1) **Entrepreneur’s Problem** An entrepreneur faces the following problem. In each of \( T \) periods, he can buy, sell, or hold for later sale some commodity, subject to the following constraints. In each period \( i \), he can buy at most \( a_i \) units of the commodity, can holdover at most \( b_i \) units of the commodity for the next period, and must sell at least \( c_i \) units of
commodity (perhaps due to former agreements). The entrepreneur cannot sell the commodity in the same period he buys it. Assuming that \( p_i, w_i \) and \( s_i \) denote the purchase cost, inventory carrying cost, and selling price per unit in period \( i \), what buy-sell policy should the entrepreneur adopt to maximize total profit in \( T \) periods? Formulate this problem as a minimum cost flow problem for \( T = 4 \).

(5.4.2) **Allocation of Contractors to Public Works** A large publicly owned corporation has 12 divisions in the country. Each division faces a similar problem. Each year the division subcontracts work to private contractors. The work is of several different types and is done by teams, each of which is capable of doing all types of work. One of these divisions is divided into several districts: the \( j \)th district requires \( r_j \) teams. The contractors are of two types: experienced and inexperienced. Each contractor \( i \) quotes a price \( c_{ij} \) to have a team conduct the work in district \( j \). The objective is to allocate the work in the districts to the various contractors, satisfying the following conditions: (1) each district \( j \) has \( r_j \) assigned teams; (2) the division contracts with contractor \( i \) for no more than \( u_i \) teams, the maximum number of teams it can supply; and (3) each district has at least one experienced contractor assigned to it. Formulate this problem as a minimum cost flow problem for a division with three districts, and with two experienced and two inexperienced contractors. (Hint: Split each district vertex into two vertices, one of which requires an experienced contractor.)

(5.4.5) With respect to an optimal solution \( x^* \) of a minimum cost flow problem, suppose that we redefine arc capacities \( u' \) as follows:

\[
  u'_{ij} = \begin{cases} 
    u_{ij} & \text{if } x^*_{ij} = u_{ij} \\
    \infty & \text{if } x^*_{ij} < u_{ij}
  \end{cases}
\]

Show that \( x^* \) is also an optimal solution of the minimum cost flow problem with the arc capacities as \( u' \).

(5.4.6) With respect to an optimal solution \( x^* \) of a minimum cost flow problem, suppose that we redefine arc capacities \( u' = x^* \). Show that \( x^* \) is also an optimal solution of the minimum cost flow problem with arc capacities \( u' \).

(5.4.7) In the minimum cost flow problem, suppose that one specific arc \((p, q)\) has no lower and upper flow bounds. How would you transform this problem into the standard minimum cost flow problem?

(5.4.8) Let \((k, l)\) and \((p, q)\) denote a minimum cost arc and a maximum cost arc in a network. Is it possible that no minimum cost flow have a positive flow on arc \((k, l)\)? Is it possible that every minimum cost flow have a positive flow on arc \((p, q)\)? Justify your answer.
(5.4.9) Prove or disprove the following claims.

(i) Suppose that all supply/demands and arc capacities in a minimum cost flow problem are all even integers. Then for some optimal flow $x^*$, each arc flow $x^*_{ij}$ is an even integer.

(ii) Suppose that all supply/demands and arc capacities in a minimum cost circulation problem are all even integers. Then for some optimal flow $x^*$, each arc flow $x^*_{ij}$ is an even integer.

(5.4.10) Let $x^*$ be an optimal solution of the minimum cost flow problem. Define $G^o$ as a subgraph of the residual network $G(x^*)$ consisting of all arcs with zero reduced cost. Show that the minimum cost flow problem has an alternative solution (other than $x^*$) if and only if $G^o$ has a dicircuit.

(5.4.11) Suppose that you are given a non integral optimal solution to a minimum cost flow problem with integral data. Suggest a method for converting this solution into an integer optimal solution. Your method should maintain optimality of the solution at every step.
6 Optimal Matching

1. Alternating Paths

(6.1.1) Let $G = (V, E)$ be an undirected graph. A subset $M \subseteq E$ is a matching if no two edges of $M$ are adjacent in $G$. If one edge in $M$ is incident with a vertex $v \in V$, then $v$ is $M$-covered. Vertices that are not $M$-covered are $M$-exposed. A maximum matching is one with maximum cardinality. A maximal matching is a matching $M$ such that $G$ has a matching $M_1$ with $M \subseteq M_1$, then $M = M_1$.

Notation: $\nu(G) = \max\{|M| : M \text{ is a matching of } G\}$. Then $2\nu(G) \leq |V|$. 

$\text{def}(G) = |V| - 2\nu(G)$ is the deficiency of $G$.

Any matching $M$ with $2|M| = |V|$ is called a perfect matching.

(6.1.2) The following are equivalent:

(i) $G$ has a perfect matching.
(ii) $\text{def}(G) = 0$.
(iii) $G$ has a matching that covers all the vertices.

(6.1.3) The Problems: Let $G = (V, E)$ be a graph and let $c : E \mapsto \mathbb{R}$ be a function.

(i) Does $G$ have a perfect matching?
(ii) If $G$ has a perfect matching, how to find a minimum weighted perfect matching of $G$?
(iii) How to find a maximum matching of $G$?
(iv) How to find a minimum weighted maximum matching of $G$?

(6.1.4) (Alternating Path Theorem) Given a matching $M$ in $G$, a path $P = v_0e_1v_1e_2 \cdots e_kv_k$ is $M$-alternating if the edges of $P$ are alternatively in $M$ and not in $M$.

(i) If $P$ is $M$-alternating such that both $v_0$ and $v_k$ are $M$-exposed, then (the symmetric difference) $M_1 = M \Delta E(P) = (M - E(P)) \cup (E(P) - M)$ is also a matching of $G$ with $|M_1| = |M| + 1$, and so in this case, $P$ is an $M$-augmenting path.

(ii) $M$ is maximum if and only if $G$ has no $M$-augmenting paths.

Proof: (i) shows the only if part for (ii). For the if part of (ii), suppose that $M$ is not maximum, then $G$ has a matching $N$ with $|N| > |M|$. Then the subgraph induced by $J = M \Delta N$ is a disjoint union of circuits and/or paths whose edges are alternating in $M$ and in $N$. As $|N| > |M|$, there must be an $M$-augmenting path.

(6.1.5) Let $H$ be a graph. A component $H_1$ of $H$ is an odd component if $|V(H_1)|$ is an odd number. Let $\text{oc}(H)$ denote the number of odd components of $H$. 

47
(6.1.6) Let \( G = (V, E) \) be a graph. Each of the following must hold.

(i) If \( \text{def}(G) = 0 \), \( \forall A \subseteq V(G), \text{oc}(G - A) \leq |A| \).

(ii) In general, \( \forall A \subseteq V(G), \nu(G) \leq \frac{1}{2}(|V| - \text{oc}(G - A) + |A|) \).

**Proof:** Let \( H_1, H_2, \ldots, H_k \) be the odd components of \( G - A \).

(i) Each odd component \( H_i \) of \( G - A \) must have a vertex \( v_i \) covered by an edge in \( M \) joining to a vertex \( u_i \) in \( A \), and as \( M \) is a matching, \( v_i \mapsto u_i \) is an injection.

(ii) Let \( M \) be a matching. For each \( i \), either \( H_i \) has an \( M \)-exposed vertex \( z_i \), or \( H_i \) has a vertex \( v_i \) that is linked by an edge \( e_i \) in \( M \) to a vertex \( u_i \in A \). Thus there are at most \( |A| \) such \( e_i \)'s, and at least \( k - |A| \) of the \( z_i \)'s. Counting the number of vertices discussed, we have

\[
2|M| + (k - |A|) \leq |V|, \text{ and so } \nu(G) \leq \frac{1}{2}(|V| - \text{oc}(G - A) + |A|).
\]

(6.1.7) **Tutte’s Matching Theorem** A graph \( G = (V, E) \) has a perfect matching if and only if \( \forall A \subseteq V, \text{oc}(G - A) \leq |A| \).

(6.1.8) Let \( G \) be a graph and \( H \) be a connected subgraph of \( G \). The **contraction** \( G/H \) is the graph obtained from \( G \) by identifying the two ends of each edge in \( E(H) \). In \( G/H \), the subgraph \( H \) is shrunk into a vertex \( v_H \). We say that \( H \) is the **preimage** of \( v_H \).

(6.1.9) Let \( C \) be an odd circuit of \( G \), \( G' = G/C \), and let \( M' \) be a matching of \( G' \). Then

(i) \( G \) has a matching \( M \) such that \( M \subseteq M' \cup E(C) \) and the number of \( M \)-exposed vertices of \( G \) is the same as the number of \( M' \)-exposed vertices in \( G' \).

(ii) \( \nu(G) \geq \nu(G/C) + \frac{1}{2}(|V(C)| - 1) \). (An odd circuit \( C \) such that equality holds here is called a **tight odd circuit** of \( G \).)

**Proof:** (i) If \( v_H \) is \( M' \)-exposed in \( G' \), then pick any vertex in \( V(C) \) as \( v \), otherwise there is an edge \( e \in M' \) incident with a vertex \( v \) in \( C \). \( C - v \) has a perfect matching \( M'' \) and \( M = M' \cup M'' \) satisfies the conclusion.

(ii) Choose \( |M'| = \nu(G/C) \) and note that \( |M''| = \frac{1}{2}(|V(C)| - 1) \).

(6.1.10) Let \( G = (V, E) \) be a graph. A vertex \( v \in V \) is **inessential** if \( G \) has a maximum matching \( M \) such that \( v \) is \( M \)-exposed. Any vertex which is not inessential is a **essential** vertex of \( G \) (that is, a vertex is essential if and only if every maximum matching of \( G \) covers it).

(6.1.11) If \( A \subseteq V \) such that

\[
\nu(G) \leq \frac{1}{2}(|V| - \text{oc}(G - A) + |A|),
\]

48
then every vertex in \( A \) is essential.

**Proof:** Pick \( v \in A \), and let \( G' = G - v \) and \( A' = A - v \). Then \( \text{oc}(G - A) = \text{oc}(G' - A') \) but \( |A| < |A'| \). It follows that \( \nu(G') < \nu(G) \) and so every maximum matching \( M \) must cover \( v \).

(6.1.12) Let \( G = (V, E) \) be a graph, and \( e = v_1v_2 \in E(G) \). If \( v_1, v_2 \) are inessential, then \( G \) has a tight odd circuit \( C \) using \( e \). Moreover, \( v_C \) is inessential in \( G/C \).

**Proof:** For \( i = 1, 2 \), let \( M_i \) be a maximum matching of \( G \) not covering \( v_i \). Since \( M_i \) is maximum, \( M_i \) covers \( v_{3-i} \).

Let \( H \) be a spanning subgraph of \( G \) with edge set \( M_1 \Delta M_2 \). Then \( v_1 \) is in a component \( H_1 \) of \( H \). By the Alternating Path Theorem, \( H_1 \) is an \((v_1, u_1)\)-path such that \( u_1 \) is \( M_1 \)-covered. If \( u_1 \neq v_2 \), then \( H_1 + e \) is an \( M_2 \)-augmenting path, contrary to the assumption that \( M_2 \) is maximum. Therefore, \( u_1 = v_2 \). Thus \( C = H_1 + e \) is an odd circuit, and \( |M_1 \cup E(C)| = \frac{1}{2}(|V(C)| - 1) \), indicating that \( C \) is tight.

Moreover, \( M_1 - E(C) \) is a maximum matching of \( G/C \) not covering \( v_{C} \).

(6.1.13) (**Tutte-Berge Formula**) For a graph \( G = (V, E) \),

\[
\nu(G) = \min \left\{ \frac{1}{2}(|V| - \text{oc}(G - A) + |A|) : A \subseteq V \right\}.
\]

**Proof:** By (6.1.6)(ii), we only need to show that there exist a matching \( M \) and a set \( A \) such that

\[
|M| = \left\lfloor \frac{1}{2}(|V| - \text{oc}(G - A) + |A|) \right\rfloor,
\]

or, equivalently, the number of \( M \)-exposed vertices equals \( |A| - \text{oc}(G - A) \). We construct such a matching \( M \) and a set \( A \) by induction on \(|E|\). As (6.1.13) holds trivially for graphs with \(|E| = 0\), we assume that \(|E| > 0\).

Pick \( e = v_1v_2 \in E \). Suppose that \( v_1 \) (or \( v_2 \)) is essential. Then \( \nu(G - v_1) = \nu(G) - 1 \). By induction, \( G' = G - v_1 \) has a vertex subset \( A' \) and a matching \( M' \) such that \( 2|M'| + |A'| - \text{oc}(G' - A') = |V| - 1 \). Since \( \nu(G - v_1) = \nu(G) - 1 \), \( G \) has a matching \( M \) with \( |M| = |M'| + 1 \). Let \( A = A' \cup v \). Then \( \text{oc}(G' - A') = \text{oc}(G - A) \), and so

\[
2|M| + |A| - \text{oc}(G - A) = |V|.
\]

Now suppose that both \( v_1 \) and \( v_2 \) are inessential. By (6.1.12), there is a tight circuit \( C \).

By induction, \( G'' = G/C \) has a matching \( M'' \) and a set \( A'' \) such that \( 2|M''| + |A''| - \text{oc}(G'' - A'') = |V(G'')| = |V| - |V(C)| + 1 \). Thus the \( M'' \)-exposed vertices is \( |A''| - \text{oc}(G'' - A'') \). By (6.1.9)(i), \( M'' \) can be extended to a matching \( M \) in \( G \) such that the \( M \)-exposed vertices in \( G \) is \( |A''| - \text{oc}(G'' - A'') = |V| - 2|M| \). (Need only to set \( A = A'' \)).
**Proof of (6.1.7):**  By (6.1.13), \( G = (V, E) \) has a perfect matching if and only if
\[
\frac{1}{2}|V| = \nu(G) = \min \{ \frac{1}{2}(|V| - \text{oc}(G - A) + |A|) : A \subseteq V \},
\]
which is equivalent to
\[
\max \{|A| - \text{oc}(G - A) : A \subseteq V \} \geq 0 \iff \forall A \subseteq V, |A| \geq \text{oc}(G - A).
\]

2. Maximum Matching Algorithms

(6.2.1) **The M-alternating Tree Algorithm:**

**Input:** A graph \( G = (V, E) \), a matching \( M \) of \( G \), an \( M \)-exposed vertex \( r \) on \( G \).

**Initialization:** \( A = \emptyset, B = \{r\}, X = \emptyset, T = (A \cup B, X) \).

**Iteration:** While \( \exists e = vw \in E \) such that \( v \in B \), \( w \notin A \cup B \), and \( \exists wz \in M \),
\[
\text{set } A := A \cup \{w\} \text{ and } B := B \cup \{z\}, \\
X := X \cup \{vw, wz\} \text{ and } T := (A \cup B, X).
\]

We have these conclusions:

(i) This algorithm generates a (maximal) subgraph \( T \) and two sets \( A \) and \( B \); \( T \) is a tree such that every \( v \in V(T) - \{r\} \) is covered by an edge in \( M \cap E(T) \).

(ii) For every \( v \in A \), the unique \((r, v)\)-path in \( T \) is an odd-length \( M \)-alternating path in \( T \); for every \( v \in B \), the unique \((r, v)\)-path in \( T \) is an even-length \( M \)-alternating path in \( T \).

Therefore \(|B| = |A| + 1\).

The tree \( T \) is an **M-alternating tree** from \( r \) (or rooted at \( r \)) in \( G \), and the sets \( A \) and \( B \) are also denoted \( A(T) \) and \( B(T) \).

(6.2.2) An \( M \)-alternating tree \( T \) is **frustrated** if for every \( v \in B(T) \) and for every \( vw \in E \), \( w \in A(T) \).

(6.2.3) Let an \( M \)-alternating tree \( T \) rooted at \( r \) be generated by the Algorithm (6.2.1). Then \( T \) is not frustrated if and only if \( G \) has an \( M \)-augmenting path from \( r \).

**Proof:** \( T \) is not frustrated \( \iff \exists v \in B(T) \) and \( vw \in E \) such that \( w \notin A(T) \) (as \( T \) is generated by (6.2.1), if \( w \) is \( M \)-covered, then \( \exists wz \in M \) and so \( w \) would have been in \( A(T) \)) and \( w \) is not \( M \)-covered \( \iff \) The unique \((r, v)\)-path in \( T \) together with \( vw \) is an \( M \)-augmenting path.

(6.2.4) If \( G \) has a matching \( M \), and a frustrated \( M \)-alternating tree \( T \), then \( G \) has no
perfect matching.

**Proof:** Let \( A = A(T) \). Then \( oc(G - A) \geq |B(T)| > |A| \). Then apply Tutte’s Matching Theorem.

(6.2.5) **An idea to develop a Perfect Matching Algorithm**

**Input:** A graph \( G = (V, E) \).

**Initialization:** \( M = \emptyset \).

**Iteration:** While \( \exists r \in V \) such that \( r \) is \( M \)-exposed,
Apply (6.2.1) to find an \( M \)-alternating tree from \( r \).
If \( T \) is not frustrated, then find an \( M \)-augmenting path \( P \),
and augment \( M := M \odot E(P) \).
If \( G \) has no \( M \)-exposed vertices,
then STOP, \( M \) is a perfect matching of \( G \).
ELSE (\( T \) is frustrated) STOP
and \( M \) does not have a perfect matching.

Why does it work? The Algorithm stops when either (1) \( G \) has no more \( M \)-exposed vertices, whence \( M \) is a perfect matching; or (2) \( T \) is frustrated, whence by (6.2.4), \( G \) does not have a perfect matching.

(6.2.6) Suppose that \( G = (V, E) \) is bipartite with a matching \( M \) and an \( M \)-alternating tree \( T \). If \( G \) has no edge joining a vertex in \( B(T) \) to a vertex in \( V(G) - V(T) \), then \( T \) must be frustrated.

**Proof:** Denote \( A = A(T) \) and \( B = B(T) \). Assume that \( T \) is from \( r \). Let \( v \in B \) and \( vw \in E \). If \( w \not\in A \), then the assumption says \( w \in V(T) \) and so we must have \( w \in B \). Then the even \((r,v)\)-path and \((r,w)\)-path together with \( vw \) shows that \( G \) has an odd cycle.

(6.2.7) **Perfect Matching Algorithm for Bipartite Graphs**

**Input:** A graph \( G = (V, E) \).

**Initialization:** \( M = \emptyset \).

**Iteration:** While \( \exists r \in V \) such that \( r \) is \( M \)-exposed,
Set \( T := (\{r\}, \emptyset) \).
While \( \exists vw \in E \) with \( v \in B(T) \), \( w \in V(T) \)
IF \( w \) is \( M \)-exposed, THEN use \( vw \) to augment \( M : M \cup \{vw\} \);
If \( G \) has no \( M \)-exposed vertices,
then STOP, \( M \) is a perfect matching of \( G \).
STOP: \( M \) does not have a perfect matching.

Why does it work? The Algorithm stops when either (1) \( G \) has no more \( M \)-exposed vertices, whence \( M \) is a perfect matching; or (2) for some \( M \)-augmenting tree \( T \), \( G \) has no
edges joining a vertex in $B(T)$ to a vertex in $V(G) - V(T)$, which implies, by (6.2.6), $T$ is frustrated, whence by (6.2.4), $G$ does not have a perfect matching.

(6.2.8) Let $G$ be a graph and let $C$ be an odd circuit of $G$. Then $G/C$ is an odd circuit contraction or an odd circuit shrinking of $G$. A graph $G'$ is a derived graph of $G$ if $G'$ is obtained from $G$ by a sequence of odd circuit contractions. If a vertex $v$ of $G'$ is the contraction image of an odd circuit, then $v$ is a pseudo-vertex of $G'$; otherwise $v$ is an original vertex of $G'$.

For each $v \in V(G')$, if $v$ is an original vertex, then let $S(v) = \{v\}$; if $v$ is a pseudo-vertex, then let $S(v)$ denote the set of all the vertices in $G$ that are eventually contracted into $v$.  

52