

Notes for Operations Research & More

Lan Peng, PhD Student

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

November 6, 2019

November 6, 2019

Contents

I	Preliminary Topics	7
1	Introduction to Optimization	9
1.1	Optimization Model	9
2	Review of Linear Algebra	11
2.1	Field	11
2.2	Real Vector Spaces	11
2.3	Linear, Conic, Affine, and Convex Combinations	11
2.4	Determinants	11
2.5	Inner Products	11
2.6	Norms	12
2.7	Eigenvectors and Eigenvalues	12
2.8	Decompositions	12
3	Review of Real Analysis	13
3.1	Sequences and Series	13
4	Review of Topology	15
4.1	Open Sets and Closed Sets	15
II	Linear Programming	17
5	Formulation	19
5.1	Linear Programming Problem Manipulation	19
5.1.1	Inequalities and Equalities	19
5.1.2	Minimization and Maximization	19
5.1.3	Standard Form and Canonical Form	19
5.2	Typical Problems	19
5.3	Formulation Skills	19
5.3.1	Absolute Value	19
5.3.2	A Minimax Objective	20
5.3.3	A fractional Objective	20
5.3.4	A range Constraint	20
6	Simplex Method	21
6.1	Basic Feasible Solutions and Extreme Points	21
6.2	Simplex Method	21
6.2.1	Key to Simplex Method	21
6.2.2	Simplex Method Algorithm	22
6.3	Find Elements in Tableau	22
6.4	Artificial Variable	22
6.4.1	Two-Phase Method	22
6.4.2	Big M Method	23
6.4.3	Single Artificial Variable	23
6.5	Revised Simplex Method	23

6.5.1	Key to Revised Simplex Method	23
6.5.2	Comparison between Simplex and Revised Simplex	23
6.5.3	Decomposition of B inverse	23
6.6	Simplex for Bounded Variables	24
6.6.1	Bounded Variable Formulation	24
6.6.2	Basic Feasible Solution	24
6.6.3	Improving Basic Feasible Solution	25
6.7	Degeneracy and Cycling	25
6.7.1	Degeneracy	25
6.7.2	Cycling	25
6.7.3	Cycling Prevent	25
6.8	Dual Simplex Method	25
6.9	As a Search Algorithm	25
6.9.1	Improving Search Algorithm	25
6.9.2	Optimality Test	26
6.9.3	Find Direction	26
6.9.4	Find the Step Length	26
6.9.5	Simplex Method Algorithm	26
6.9.6	Simplex Method Tableau	26
6.9.7	Simplex Method as a Search Algorithm	26
7	Duality, Sensitivity and Relaxation	27
7.1	Duality	27
7.1.1	Dual Formulation	27
7.1.2	Mixed Forms of Duality	27
7.1.3	Dual of the Dual is the Primal	27
7.1.4	Primal-Dual Relationships	28
7.1.5	Shadow Price	28
7.2	Sensitivity	28
7.2.1	Change in the Cost Vector	28
7.2.2	Change in the Right-Hand-Side	29
7.2.3	Change in the Matrix	29
7.3	Relaxation	29
7.3.1	Why Rounding Can be Bad - IP Example	29
7.3.2	Why Rounding Can be Bad - QAP example	29
7.3.3	IP and Convex Hull	30
7.3.4	Local Optimal and Global Optimal	30
7.3.5	LP Relaxation	30
8	Decomposition Principle	31
9	Ellipsoid Algorithm	33
10	Projective Algorithm	35
11	Interior-Point Algorithm	37
III	Graph and Network Theory	39
12	Graphs and Subgraphs	41
12.1	Graph	41
12.2	Graph Isomorphism	41
12.3	The Adjacency and Incidence Matrices	41
12.4	Subgraph	41
12.5	Degree	41
12.6	Special Graphs	42
12.7	Directed Graph	43

12.8 Sperner's Lemma	43
13 Paths, Trees, and Cycles	45
13.1 Walk	45
13.2 Path and Cycle	45
13.3 Tree and forest	46
13.4 Spanning tree	46
13.5 Cayley's Formula	48
13.6 Connectivity	48
13.7 Blocks	48
14 Euler Tours and Hamilton Cycles	49
14.1 Euler Tours	49
14.2 Hamilton Cycles	49
15 Planarity	51
15.1 Plane and Planar Graphs	51
15.2 Dual Graphs	51
15.3 Euler's Formula	51
15.4 Bridges	51
15.5 Kuratowski's Theorem	51
15.6 Four-Color Theorem	51
15.7 Graphs on other surfaces	51
16 Minimum Spanning Tree Problem	53
16.1 Basic Concepts	53
16.2 Kroskal's Algorithm	53
16.3 Prim's Algorithm	53
16.4 Comparison between Kroskal's and Prim's Algorithm	53
16.5 Extensible MST	53
16.6 Solve MST in LP	54
17 Shortest-Path Problem	55
17.1 Basic Concepts	55
17.2 Breadth-First Search Algorithm	55
17.3 Ford's Method	55
17.4 Ford-Bellman Algorithm	55
17.5 SPFA Algorithm	56
17.6 Dijkstra Algorithm	56
17.7 A* Algorithm	56
17.8 Floyd-Warshall Algorithm	56
17.9 Johnson's Algorithm	57
18 Maximum Flow Problem	59
18.1 Basic Concept	59
18.2 Solving Maximum Flow Problem in LP	59
18.3 Prime and Dual of Maximum Network Flow Problem	60
18.4 Maximum Flow Minimum Cut Theorem	60
18.5 Ford-Fulkerson Method	61
18.6 Polynomial Algorithm for max flow	61
18.7 Dinic Algorithm	62
19 Minimum Weight Flow Problem	63
19.1 Transshipment Problem	63
19.2 Network Simplex Method	63

20 Matchings	65
20.1 Hall's "Marriage" Theorem	65
20.2 Transversal Theory	65
20.3 Menger's Theorem	65
20.4 The Hungarian Algorithm	65
21 Colorings	67
21.1 Edge Chromatic Number	67
21.2 Vizing's Theorem	67
21.3 The Timetabling Problem	67
21.4 Vertex Chromatic Number	67
21.5 Brooks' Theorem	67
21.6 Hajós' Theorem	67
21.7 Chromatic Polynomials	67
21.8 Girth and Chromatic Number	67
22 Independent Sets and Cliques	69
22.1 Independent Sets	69
22.2 Ramsey's Theorem	69
22.3 Turán's Theorem	69
22.4 Schur's Theorem	69
23 Matroids	71

Part I

Preliminary Topics

Chapter 1

Introduction to Optimization

1.1 Optimization Model

The following is the basic forms of terminology:

$$(P) \quad \min \quad f(x) \quad (1.1)$$

$$\text{s.t.} \quad g_i(x) \leq 0, \quad i = 1, 2, \dots, m \quad (1.2)$$

$$h_j(x) = 0, \quad j = 1, 2, \dots, l \quad (1.3)$$

$$x \in X \quad (1.4)$$

We have

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

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

Types of Optimization Problem

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

Chapter 2

Review of Linear Algebra

2.1 Field

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

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

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

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

Example. The set of integers is not a field.

Example. The set of rational numbers is a field.

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

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

2.2 Real Vector Spaces

2.3 Linear, Conic, Affine, and Convex Combinations

2.4 Determinants

2.5 Inner Products

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

- $\langle \alpha + \beta | \gamma \rangle = \langle \alpha | \gamma \rangle + \langle \beta | \gamma \rangle$
- $\langle c\alpha | \beta \rangle = c \langle \alpha | \beta \rangle$
- $\langle \alpha | \beta \rangle = \overline{\langle \beta | \alpha \rangle}$
- $\langle \alpha | \alpha \rangle \geq 0, \langle \alpha | \alpha \rangle = 0$ iff $\alpha = 0$

Furthermore, the above properties imply that

- $\langle \alpha | c\beta + \gamma \rangle = \bar{c} \langle \alpha | \beta \rangle + \langle \alpha | \gamma \rangle$

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

$$\langle \alpha | \beta \rangle = \sum_j x_j \bar{y}_j \quad (2.1)$$

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

$$\langle \alpha | \beta \rangle = \sum_j x_j y_j \quad (2.2)$$

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

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

$$\langle \alpha | \beta \rangle = x_1 y_1 - x_2 y_1 - x_1 y_2 + 4x_2 y_2 \quad (2.3)$$

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

$$\langle \mathbf{A} | \mathbf{B} \rangle = \text{trace}(\mathbf{B}^* \mathbf{A}) \quad (2.4)$$

is an inner product, where

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

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

$$\langle \mathbf{A} | \mathbf{B} \rangle = \text{trace}(\mathbf{B}^T \mathbf{A}) = \sum_j (AB^T)_{jj} = \sum_j \sum_k A_{jk} B_{jk} \quad (2.6)$$

2.6 Norms

2.7 Eigenvectors and Eigenvalues

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

$$\mathbf{A}\mathbf{x} = \lambda \mathbf{x} \quad (2.7)$$

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

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

$$\det(\lambda I - A) = 0 \quad (2.8)$$

Corollary 2.1.1.

$$\sum \lambda_A = \text{tr}(\mathbf{A}) \quad (2.9)$$

Corollary 2.1.2.

$$\prod \lambda_A = \det(\mathbf{A}) \quad (2.10)$$

Notice: Gaussian elimination changes the eigenvalues.

2.8 Decompositions

Chapter 3

Review of Real Analysis

3.1 Sequences and Series

Chapter 4

Review of Topology

4.1 Open Sets and Closed Sets

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Then

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

and S is a closed set.

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

Notice: z is not necessarily in A .

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

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

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

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

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

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

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

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

Part II

Linear Programming

Chapter 5

Formulation

5.1 Linear Programming Problem Manipulation

5.1.1 Inequalities and Equalities

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

$$\sum_{j=1}^n a_{ij}x_j \geq b_i \Rightarrow \sum_{j=1}^n a_{ij}x_j - x_{n+1} = b_i \quad (5.1)$$

or

$$\sum_{j=1}^n a_{ij}x_j \leq b_i \Rightarrow \sum_{j=1}^n a_{ij}x_j + x_{n+1} = b_i \quad (5.2)$$

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

$$\sum_{j=1}^n a_{ij}x_j = b_i \Rightarrow \begin{cases} \sum_{j=1}^n a_{ij}x_j \leq b_i \\ \sum_{j=1}^n a_{ij}x_j \geq b_i \end{cases} \quad (5.3)$$

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

$$\sum_{j=1}^n a_{ij}x_j > b_i \Rightarrow \sum_{j=1}^n a_{ij}x_j \geq b_i + \epsilon \quad (5.4)$$

where ϵ is a small number.

5.1.2 Minimization and Maximization

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

$$\min \sum_{j=1}^n c_j x_j = -\max \sum_{j=1}^n c_j x_j \quad (5.5)$$

5.1.3 Standard Form and Canonical Form

Standard Form

$$\min \sum_{j=1}^n c_j x_j \quad (5.6)$$

$$\text{s.t. } Ax = b \quad (5.7)$$

$$x \geq 0 \quad (5.8)$$

Canonical Form

$$\min \sum_{j=1}^n c_j x_j \quad (5.9)$$

$$\text{s.t. } Ax \leq b \quad (5.10)$$

$$x \geq 0 \quad (5.11)$$

5.2 Typical Problems

5.3 Formulation Skills

5.3.1 Absolute Value

Consider the following model statement:

$$\min \sum_{j \in J} c_j |x_j|, \quad c_j > 0 \quad (5.12)$$

$$\text{s.t. } \sum_{j \in J} a_{ij}x_j \gtrless b_i, \quad \forall i \in I \quad (5.13)$$

$$x_j \text{ unrestricted}, \quad \forall j \in J \quad (5.14)$$

Modeling:

$$\min \sum_{j \in J} c_j (x_j^+ + x_j^-), \quad c_j > 0 \quad (5.15)$$

$$\text{s.t. } \sum_{j \in J} a_{ij}(x_j^+ - x_j^-) \gtrless b_i, \quad \forall i \in I \quad (5.16)$$

$$x_j^+, x_j^- \geq 0, \quad \forall j \in J \quad (5.17)$$

5.3.2 A Minimax Objective

Consider the following model statement:

$$\min \max_{k \in K} \sum_{j \in J} c_{kj} x_j \quad (5.18)$$

$$\text{s.t. } \sum_{j \in J} a_{ij} x_j \geq b_i, \quad \forall i \in I \quad (5.19)$$

$$x_j \geq 0, \quad \forall j \in J \quad (5.20)$$

Modeling:

$$\min z \quad (5.21)$$

$$\text{s.t. } \sum_{j \in J} a_{ij} x_j \geq b_i, \quad \forall i \in I \quad (5.22)$$

$$\sum_{j \in J} c_{kj} x_j \leq z, \quad \forall k \in K \quad (5.23)$$

$$x_j \geq 0, \quad \forall j \in J \quad (5.24)$$

Modeling:

$$\min \sum_{j \in J} c_j x_j, \quad c_j > 0 \quad (5.37)$$

$$\text{s.t. } u_i + \sum_{j \in J} a_{ij} x_j = e_i, \quad \forall i \in I \quad (5.38)$$

$$x_j \geq 0, \quad \forall j \in J \quad (5.39)$$

$$0 \leq u_i \leq e_i - d_i, \quad \forall i \in I \quad (5.40)$$

5.3.3 A fractional Objective

Consider the following model statement:

$$\min \frac{\sum_{j \in J} c_j x_j + \alpha}{\sum_{j \in J} d_j x_j + \beta} \quad (5.25)$$

$$\text{s.t. } \sum_{j \in J} a_{ij} x_j \geq b_i, \quad \forall i \in I \quad (5.26)$$

$$x_j \geq 0, \quad \forall j \in J \quad (5.27)$$

Modeling:

$$\min \sum_{j \in J} c_j x_j t + \alpha t \quad (5.28)$$

$$\text{s.t. } \sum_{j \in J} a_{ij} x_j \geq b_i, \quad \forall i \in I \quad (5.29)$$

$$\sum_{j \in J} d_j x_j t + \beta t = 1 \quad (5.30)$$

$$t > 0 \quad (5.31)$$

$$x_j \geq 0, \quad \forall j \in J \quad (5.32)$$

$$(t = \frac{1}{\sum_{j \in J} d_j x_j + \beta}) \quad (5.33)$$

5.3.4 A range Constraint

Consider the following model statement:

$$\min \sum_{j \in J} c_j x_j \quad (5.34)$$

$$\text{s.t. } d_i \leq \sum_{j \in J} a_{ij} x_j \leq e_i, \quad \forall i \in I \quad (5.35)$$

$$x_j \geq 0, \quad \forall j \in J \quad (5.36)$$

Chapter 6

Simplex Method

6.1 Basic Feasible Solutions and Extreme Points

Definition 6.1.1. Consider the system $\{A_{m \times n}x = b_m, x \geq 0\}$, suppose $\text{rank}(A, b) = \text{rank}(A) = m$, we can arrange A and have a partition of A . Let $A = [B, N]$ where B is $m \times m$ invertible matrix, and N is a $m \times (n-m)$ matrix. The solution $x = \begin{bmatrix} x_B \\ x_N \end{bmatrix}$ to the equation $Ax = b$, where

$$x_B = B^{-1}b \quad (6.1)$$

and

$$x_N = 0 \quad (6.2)$$

is called **basic solution** of system. If $x_B \geq 0$, it is called **basic feasible solution**. If $x_B > 0$ it is called **non-degenerate basic feasible solution**. For $x_B \geq 0$, if some $x_j = 0$, those components are called **degenerated basic feasible solution**.

B is called the **basic matrix**, N is called **nonbasic matrix**.

Theorem 6.1. x is an E.P. $\iff x$ is a B.F.S

Proof. First, Let x be a B.F.S., Suppose $x = \lambda u + (1 - \lambda)v$, for $u, v \in S, \lambda \in (0, 1)$

Let $I = \{i : x_i > 0\}$ Then

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

Second, suppose x is not B.F.S., i.e. $\{a_i : i \in I\}$ are linearly dependent.

Then there $\exists u \neq 0, u_i = 0, i \notin I$ such that $Au = 0$.

Hence, for a small ϵ , $x = \frac{1}{2}(x + \epsilon u) + \frac{1}{2}(x - \epsilon u)$, x is not E.P. \square

6.2 Simplex Method

6.2.1 Key to Simplex Method

Cost Coefficient

The cost coefficient can be derived from the following

$$z = cx \quad (6.3)$$

$$= c_B x_B + c_N x_N \quad (6.4)$$

$$= c_B (B^{-1}b - B^{-1}N x_N) + c_N x_N \quad (6.5)$$

$$= c_B B^{-1}b - \sum_{j \in N} (c_B B^{-1}a_j - c_j)x_j \quad (6.6)$$

$$= c_B B^{-1}b - \sum_{j \in N} (z_j - c_j)x_j \quad (6.7)$$

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

The formulation can be transformed into

$$\min \quad z = z_0 - \sum_{j \in N} (z_j - c_j)x_j \quad (6.8)$$

$$\text{s.t.} \quad \sum_{j \in N} y_j x_j + x_B = \bar{b} \quad (6.9)$$

$$x_j \geq 0, j \in N \quad (6.10)$$

$$x_B \geq 0 \quad (6.11)$$

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

Pivot

After finding the most violated $z_j - c_j$, we find a variable, say x_k , where $z_k - c_k = \min\{z_j - c_j\}$ to be the variable leaving the basis.

If there are degenerated variables, we can perform different method to choose variable to enter basis.

Minimum Ratio

$$x_{B_i} = \bar{b}_i - y_{ik}x_k \geq 0 \quad (6.12)$$

Therefore we have the minimum ratio rule

$$x_k = \min_{i \in B} \left\{ \frac{\bar{b}_i}{y_{ik}}, y_{ik} > 0 \right\} \quad (6.13)$$

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

6.2.2 Simplex Method Algorithm

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

Algorithm 1 Simplex Method

Require: Given a basic feasible solution with basis B

Ensure: Optimal objective value $\min z = cx$

```

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

```

6.3 Find Elements in Tableau

Initial tableau:

	z	x_1	x_2	x_3	x_4	x_5	RHS
z	1	1	3	0	0	0	0
x_3	0	1	-2	1	0	0	0
x_4	0	-2	1	0	1	0	4
x_5	0	5	3	0	0	1	15

(6.14)

Last tableau:

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

(6.15)

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

$$B = \begin{pmatrix} \frac{13}{11} & \frac{3}{11} & \frac{97}{11} \\ \frac{5}{11} & \frac{2}{11} & \frac{50}{11} \\ -\frac{3}{11} & \frac{1}{11} & \frac{3}{11} \end{pmatrix} \quad (6.16)$$

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

$$B = \begin{pmatrix} 1 & -2 & 1 \\ 0 & 1 & -2 \\ 0 & 3 & 5 \end{pmatrix} \quad (6.17)$$

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

$$c_B = \begin{pmatrix} 0 \\ 3 \\ 1 \end{pmatrix} \quad (6.18)$$

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

$$w = c_B B^{-1} = \begin{pmatrix} 0 \\ -\frac{12}{11} \\ -\frac{7}{11} \end{pmatrix} \quad (6.19)$$

6.4 Artificial Variable

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

$$\sum_{i=1}^n a_i x_i \geq (\text{or } =) 0 \Rightarrow \sum_{j=1}^n a_j x_j + x_a = 0 \quad (6.20)$$

Notice that in an optimal solution, $x_a = 0$, otherwise it is not valid.

Artificial variables are only a tool to get the simplex method started.

6.4.1 Two-Phase Method

Two-Phase Method

For **Phase I:**

Solve the following program start with a basic feasible

solution $x = 0, x_a = b$, i.e., the artificial variable forms the basis.

$$\min \quad 1x_a \quad (6.21)$$

$$\text{s.t.} \quad Ax + x_a = b \quad (6.22)$$

$$x \geq 0 \quad (6.23)$$

$$x_a \geq 0 \quad (6.24)$$

If the optimal $1x_a \neq 0$, infeasible, stop. Otherwise proceed Phase II. For **Phase II**:

Remove the columns of artificial variables, replace the objective function with the original objective function, proceed to solve simplex method.

Discussion

Case A: $x_a \neq 0$

Infeasible.

Case B.1: $x_a = 0$ and all artificial variables are out of the basis

At the end of Phase I, we derive

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

(6.25)

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

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

(6.26)

Case B.2: Some artificial variables are in the basis at zero values

This is because of degeneracy. We pivot on those artificial variables, once they leave the basis, eliminate them.

6.4.2 Big M Method

6.4.3 Single Artificial Variable

6.5 Revised Simplex Method

6.5.1 Key to Revised Simplex Method

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

The original matrix:

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

(6.27)

The revised matrix:

Basic Inverse	RHS
$w = c_B B^{-1}$	$c_B b = c_B B^{-1}b$
B^{-1}	$b = B^{-1}b$

(6.28)

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

B^{-1}	RHS	x_k
w	$c_B b$	$z_k - c_k$
B^{-1}	b	y_k

(6.29)

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

B^{-1}	RHS	x_k
w	$c_B b$	$z_k - c_k$
B^{-1}	b_1	y_{1k}
	b_2	y_{2k}
	\dots	\dots
	b_r	y_{rk} (pivot at here)
	\dots	\dots
	b_m	y_{mk}

(6.30)

6.5.2 Comparison between Simplex and Revised Simplex

Advantage of Revised Simplex

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

Disadvantage of Revised Simplex

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

Computation Complexity

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

(6.31)

When to use?

- When $m \gg n$, do revised simplex method on the dual problem
- When $m \simeq n$, revised simplex method is not as good as simplex method
- When $m \ll n$ perfect for revised simplex method.

6.5.3 Decomposition of B inverse

Let $B = \{a_{B_1}, a_{B_2}, \dots, a_{B_r}, \dots, a_{B_m}\}$ and B^{-1} is known. If a_{B_r} is replaced by a_{B_k} , then B becomes \bar{B} . Which means a_{B_r} enters the basis and a_{B_k} leaves the basis.

Then \bar{B}^{-1} can be represent by B^{-1} . Noting that $a_k =$ and By_k and $a_{B_i} = Be_i$, then

$$\bar{B} = (a_{B_1}, a_{B_2}, \dots, a_{B_{r-1}}, a_k, a_{B_{r+1}}, a_m) \quad (6.32)$$

$$= (Be_1, Be_2, \dots, Be_{r-1}, By_k, Be_{r+1}, \dots, Be_m) \quad (6.33)$$

$$= BT \quad (6.34)$$

where T is

$$T = \begin{bmatrix} 1 & 0 & \dots & 0 & y_{1k} & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & y_{2k} & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 1 & y_{r-1,k} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & y_{rk} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & y_{r+1,k} & 1 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 & y_{mk} & 0 & \dots & 1 \end{bmatrix} \quad (6.35)$$

and

$$E = T^{-1} = \begin{bmatrix} 1 & 0 & \dots & 0 & \frac{-y_{1k}}{y_{rk}} & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & \frac{-y_{2k}}{y_{rk}} & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 1 & \frac{-y_{r-1,k}}{y_{rk}} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \frac{1}{y_{rk}} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \frac{-y_{r+1,k}}{y_{rk}} & 1 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 & \frac{-y_{mk}}{y_{rk}} & 0 & \dots & 1 \end{bmatrix} \quad (6.36)$$

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

$$B_t^{-1} = E_{t-1}E_{t-2} \cdots E_2E_1I \quad (6.37)$$

Using E in calculation can simplify the product of matrix where

$$cE = c_1, c_2, \dots, c_m \begin{bmatrix} 1 & 0 & \dots & g_1 & \dots & 0 \\ 0 & 1 & \dots & g_2 & \dots & 0 \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & \dots & g_m & \dots & 1 \end{bmatrix} \quad (6.38)$$

$$= (c_1, c_2, \dots, c_{r-1}, cg, c_{r+1}, \dots, c_m) \quad (6.39)$$

$$Ea = \begin{bmatrix} 1 & 0 & \dots & g_1 & \dots & 0 \\ 0 & 1 & \dots & g_2 & \dots & 0 \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & \dots & g_m & \dots & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} \quad (6.40)$$

$$= \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{r-1} \\ 0 \\ a_{r+1} \\ \vdots \\ a_m \end{bmatrix} + a_r \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_{r-1} \\ g_r \\ g_{r+1} \\ \vdots \\ g_m \end{bmatrix} \quad (6.41)$$

$$= \bar{a} + a_r g \quad (6.42)$$

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

$$w = c_B B^{-1} = c_B E_{t-1} E_{t-2} \dots E_2 E_1 \quad (6.43)$$

$$y_k = B^{-1} a_k = E_{t-1} E_{t-2} \dots E_2 E_1 a_k \quad (6.44)$$

$$\bar{b} = B_{t+1}^{-1} b = E_t E_{t-1} E_{t-2} \dots E_2 E_1 b \quad (6.45)$$

6.6 Simplex for Bounded Variables

6.6.1 Bounded Variable Formulation

$$\min \quad cx \quad (6.46)$$

$$\text{s.t.} \quad Ax = b \quad (6.47)$$

$$l \leq x \leq b \quad (6.48)$$

Reason why we don't the following formulation

$$\min \quad cx \quad (6.49)$$

$$\text{s.t.} \quad Ax = b \quad (6.50)$$

$$x - Ix_l = l \quad (6.51)$$

$$x + Ix_u = u \quad (6.52)$$

$$x \geq 0 \quad (6.53)$$

$$x_l \geq 0 \quad (6.54)$$

$$x_u \geq 0 \quad (6.55)$$

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

6.6.2 Basic Feasible Solution

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

$$\bar{x}_B = B^{-1}b - B^{-1}N_l x_{N_l} - B^{-1}N_u x_{N_u} \quad (6.56)$$

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

6.6.3 Improving Basic Feasible Solution

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

$$x_B = B^{-1}b - B^{-1}N_l x_{N_l} - B^{-1}N_u x_{N_u} \quad (6.57)$$

$$z = c_B x_B + c_{N_l} x_{N_l} + c_{N_u} x_{N_u} \quad (6.58)$$

$$= c_B(B^{-1}b - B^{-1}N_l x_{N_l} - B^{-1}N_u x_{N_u}) \quad (6.59)$$

$$+ c_B x_B + c_{N_l} x_{N_l} + c_{N_u} x_{N_u} \quad (6.60)$$

$$= c_B B^{-1}b + (c_{N_l} - c_B B^{-1}N_l)x_{N_l} \quad (6.61)$$

$$+ (c_{N_u} - c_B B^{-1}N_u)x_{N_u} \quad (6.62)$$

$$= c_B B^{-1}b - \sum_{j \in J_1} (z_j - c_j)x_j - \sum_{j \in J_2} (z_j - c_j)x_j \quad (6.63)$$

J_1 is the set of variables at lower bound, J_2 is the set of the variables at upper bound.

Notice that the right-hand-side no longer provide $c_B B^{-1}b$ and $B^{-1}b$. For the variable entering the basis, find the variable with

$$\max\{\max_{j \in J_1}\{z_j - c_j\}, \max_{j \in J_2}\{c_j - z_j\}\} \quad (6.64)$$

to enter the basis

Tip: "Most violated rule"

The minimum ratio rule is revised for bounded simplex

$$\Delta = \min\{\gamma_1, \gamma_2, u_k - l_k\} \quad (6.65)$$

$$\gamma_1 = \begin{cases} \min_{r \in J_1} \left\{ \frac{\bar{b}_r - l_{B_r}}{y_{rk}} : y_{rk} > 0 \right\} \\ \min_{r \in J_2} \left\{ \frac{\bar{b}_r - l_{B_r}}{-y_{rk}} : y_{rk} < 0 \right\} \\ \infty \end{cases} \quad (6.66)$$

$$\gamma_2 = \begin{cases} \min_{r \in J_1} \left\{ \frac{u_{B_r} - \bar{b}_r}{-y_{rk}} : y_{rk} < 0 \right\} \\ \min_{r \in J_2} \left\{ \frac{u_{B_r} - \bar{b}_r}{y_{rk}} : y_{rk} > 0 \right\} \\ \infty \end{cases} \quad (6.67)$$

Tip:

Use $l \leq x + \Delta \leq u$ to test the range of δ , if it hits lower bound, it is called γ_1 , if it hits upper bound, it is called γ_2 .

6.7 Degeneracy and Cycling

6.7.1 Degeneracy

Degeneracy in Simplex Method

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

Degeneracy for Bounded Variables

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

6.7.2 Cycling

In the degenerate case, pivoting by the simplex rule does not always give a strict decrease in the objective function value, because it may have $b_r = 0$. It is possible that the tableau may repeat if we use the simplex rule.

Geometrically speaking, it means that at the same point - extreme point - it corresponds to more than one feasible solutions, so when we are pivoting, we stays at the same place.

In computer algorithm, we rarely care about cycling because the data in computer is not precise, it is very hard to get into cycling.

6.7.3 Cycling Prevent

Lexicographic Rule

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

Bland's Rule

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

Successive Ratio Rule

- Select the pivot column as any column k where $z_k - c_k \leq 0$
 - Given k , select the pivot row r as the minimum successive ratio row associated with column k .
- In other words, for pivot columns where there is no tie in the usual minimum ratio, the successive ratio rule reduces to the simplex rule

6.8 Dual Simplex Method

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

6.9 As a Search Algorithm

6.9.1 Improving Search Algorithm

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

represented as:

$$x^t = x^{t-1} + \lambda_{t-1} d^{t-1} \quad (6.68)$$

Where

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

6.9.2 Optimality Test

$$z = cx \quad (6.69)$$

$$= [c_B \quad c_N] \begin{bmatrix} x_B \\ x_N \end{bmatrix} \quad (6.70)$$

$$= c_B x_B + c_N x_N \quad (6.71)$$

$$\text{and } Ax = b \quad (6.72)$$

$$\therefore Bx_B + Nx_N = b, x_B \geq 0, x_N \geq 0 \quad (6.73)$$

$$\therefore x_B = B^{-1}b - B^{-1}Nx_N \quad (6.74)$$

$$z = c_B B^{-1}b - c_B B^{-1}Nx_N + c_N x_N \quad (6.75)$$

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

$$z - \hat{z} = [0 \quad c_N - c_B B^{-1}N] \begin{bmatrix} x_B \\ x_N \end{bmatrix} \quad (6.76)$$

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

6.9.3 Find Direction

Suppose we choose x_k as a candidate to pivot into Basis

$$x = \begin{bmatrix} B^{-1}b - B^{-1}a_k x_k \\ 0 + e_k x_k \end{bmatrix} = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix} + \begin{bmatrix} -B^{-1}a_k \\ e_k \end{bmatrix} x_k \quad (6.77)$$

In this form, we can see: x is the result after t th iteration, $\begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$ is the result after $(t-1)$ th iteration. $\begin{bmatrix} -B^{-1}a_k \\ e_k \end{bmatrix}$ is the iteration direction, x_k is the step length.

The only requirement of x_k is $r_k < 0$ where $r_k = c_k - z_k$ is reduce cost, which is the k th entry of $c_N - c_B B^{-1}N$. Generally speaking, we usually take $r_k = \min\{c_j - z_j\}$ (which in fact can not guarantee the efficient of the algorithm.)

6.9.4 Find the Step Length

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

$$B^{-1}b - B^{-1}a_k x_k \geq 0 \quad (6.78)$$

Denote $B^{-1}b$ as \bar{b} , denote $B^{-1}a_k$ as y_k

If $y_k \leq 0$, we can have x_k as large as infinite, which means unboundedness.

If $y_k > 0$ now we can use the minimum ratio to guarantee non-negativity.

Remember hit the bound, basic variable leave the basis and become non-basic variable.

6.9.5 Simplex Method Algorithm

6.9.6 Simplex Method Tableau

6.9.7 Simplex Method as a Search Algorithm

Chapter 7

Duality, Sensitivity and Relaxation

7.1 Duality

7.1.1 Dual Formulation

For any prime problem

$$\min \quad cx \quad (7.1)$$

$$\text{s.t.} \quad Ax \geq b \quad (7.2)$$

$$x \geq 0 \quad (7.3)$$

we can have a dual problem

$$\max \quad wb \quad (7.4)$$

$$\text{s.t.} \quad wA \leq c \quad (7.5)$$

$$w \geq 0 \quad (7.6)$$

7.1.2 Mixed Forms of Duality

For the following prime problem

$$\text{P(or D)} \quad \min \quad c_1x_1 + c_2x_2 + c_3x_3 \quad (7.7)$$

$$\text{s.t.} \quad A_{11}x_1 + A_{12}x_2 + A_{13}x_3 \geq b_1 \quad (7.8)$$

$$A_{21}x_1 + A_{22}x_2 + A_{23}x_3 \leq b_2 \quad (7.9)$$

$$A_{31}x_1 + A_{32}x_2 + A_{33}x_3 = b_3 \quad (7.10)$$

$$x_1 \geq 0 \quad (7.11)$$

$$x_2 \leq 0 \quad (7.12)$$

$$x_3 \text{ unrestricted} \quad (7.13)$$

The dual of the problem

$$\text{D(or P)} \quad \max \quad w_1b_1 + w_2b_2 + w_3b_3 \quad (7.14)$$

$$\text{s.t.} \quad w_1A_{11} + w_2A_{21} + w_3A_{31} \leq c_1 \quad (7.15)$$

$$w_1A_{12} + w_2A_{22} + w_3A_{32} \geq c_2 \quad (7.16)$$

$$w_1A_{13} + w_2A_{23} + w_3A_{33} = c_3 \quad (7.17)$$

$$w_1 \geq 0 \quad (7.18)$$

$$w_2 \leq 0 \quad (7.19)$$

$$w_3 \text{ unrestricted} \quad (7.20)$$

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

	Minimization		Maximization	
Var	≥ 0	\longleftrightarrow	≤ 0	Cons
	≤ 0	\longleftrightarrow	≥ 0	
	Unrestricted	\longleftrightarrow	$=$	
Cons	≥ 0	\longleftrightarrow	≥ 0	Var
	≤ 0	\longleftrightarrow	≤ 0	
	$=$	\longleftrightarrow	Unrestricted	

7.1.3 Dual of the Dual is the Primal

For a primal problem (P)

$$(P) \quad \min \quad cx \quad (7.21)$$

$$\text{s.t.} \quad Ax \geq b \quad (7.22)$$

$$x \geq 0 \quad (7.23)$$

The dual problem (D) is

$$(D) \quad \max \quad wb \quad (7.24)$$

$$\text{s.t.} \quad wA \leq c \quad (7.25)$$

$$w \geq 0 \quad (7.26)$$

Rewrite the dual

$$\min \quad -b^T w^T \quad (7.27)$$

$$\text{s.t.} \quad -A^T w^T \geq -c^T \quad (7.28)$$

$$w^T \geq 0 \quad (7.29)$$

Find the dual of this problem

$$\max \quad x^T(-c^T) \quad (7.30)$$

$$\text{s.t.} \quad x^T(-A^T) \leq (-b^T) \quad (7.31)$$

$$x^T \geq 0 \quad (7.32)$$

$$(7.33)$$

Rewrite the dual of the dual

$$(P) \quad \min \quad cx \quad (7.34)$$

$$\text{s.t.} \quad Ax \geq b \quad (7.35)$$

$$x \geq 0 \quad (7.36)$$

7.1.4 Primal-Dual Relationships

Weak Duality Property

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

$$Ax_0 \geq b, \quad x_0 \geq 0 \quad (7.37)$$

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

$$w_0 A \leq c, \quad w_0 \geq 0 \quad (7.38)$$

Therefore, we have

$$cx_0 \geq w_0 Ax_0 \geq w_0 b \quad (7.39)$$

which is called the weak duality property. This property is for any feasible solution in the primal and dual problem.

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

Fundamental Theorem of Duality

With regard to the primal and dual LP problems, one and only one of the following can be true.

- Both primal and dual has optimal solution x^* and w^* , where $cx^* = w^*b$
- One problem has an unbounded optimal objective value, the other problem must be infeasible
- Both problems are infeasible.

Strong Duality Property

From KKT condition, we know that in order to make x^* the optimal solution, the following condition should be met.

- Primal Optimal: $Ax^* \geq b, x^* \geq 0$
- Dual Optimal: $w^*A \leq c, w^* \geq 0$
- Complementary Slackness:

$$\begin{cases} w^*(Ax^* - b) = 0 \\ (c - w^*A)x^* = 0 \end{cases} \quad (7.40)$$

The first condition means the primal has an optimal solution, the second condition means the dual has an optimal solution. The third condition means $cx^* = w^*b$, which is also called **strong duality property**

Tip: w in the dual problem is the same as the $w = c_B B^{-1}$ in primal problem.

Complementary Slackness Theorem

Let x^* and w^* be any feasible solutions, they are optimal iff

$$(c_j - w^*a_j)x_j^* = 0, \quad j = 1, \dots, n \quad (7.41)$$

$$w_i^*(a^i x^* - b_i) = 0, \quad i = 1, \dots, m \quad (7.42)$$

In particular

$$x_j^* > 0 \Rightarrow w^*a_j = c_j \quad (7.43)$$

$$w^*a_j < c_j \Rightarrow x_j^* = 0 \quad (7.44)$$

$$w_i^* > 0 \Rightarrow a^i x^* = b_i \quad (7.45)$$

$$a^i x^* > b_i \Rightarrow w_i^* = 0 \quad (7.46)$$

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

Use Dual to Solve the Primal

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

7.1.5 Shadow Price

Shadow Price under Non-degeneracy

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

$$z = c_B B^{-1}b - \sum_{j \in N} (z_j - c_j)x_j = w^*b - \sum_{j \in N} (z_j - c_j)x_j \quad (7.47)$$

therefore

$$\frac{\partial z^*}{\partial b_i} = c_B B_i^{-1} = w_i^* \quad (7.48)$$

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

Shadow Price under Degeneracy

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

7.2 Sensitivity

7.2.1 Change in the Cost Vector

Case 1: Nonbasic Variable

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

Case 2: Basic Variable

If c_{B_t} is replaced by c'_{B_t} , then $z'_j - c_j$ is

$$z'_j - c_j = c'_B B^{-1} a_j - c_j = (z_j - c_j) - (c'_{B_t} - c_{B_t}) B^{-1} a_{B_t} \quad (7.49)$$

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

7.2.2 Change in the Right-Hand-Side

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

7.2.3 Change in the Matrix**Case 1: Changes in Activity(Variable) Vectors for Nonbasic Columns**

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

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

If a basic columns changed, it means B and B^{-1} changed, and every column changed. We can do this in two steps:

- step I: add a new column with a'_j
- step II: remove the original column a_j

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

Add a New Activity(Variable)

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

Add a New Constraint

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

7.3 Relaxation**7.3.1 Why Rounding Can be Bad - IP Example**

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

$$\max \quad z = x_1 + 0.64x_2 \quad (7.50)$$

$$\text{s.t.} \quad 50x_1 + 31x_2 \leq 250 \quad (7.51)$$

$$3x_1 - 2x_2 \geq -4 \quad (7.52)$$

$$x_1, x_2 \geq 0 \quad (\text{for LP}) \quad (7.53)$$

$$x_1, x_2 \in \mathbb{Z}^+ \quad (\text{for IP}) \quad (7.54)$$

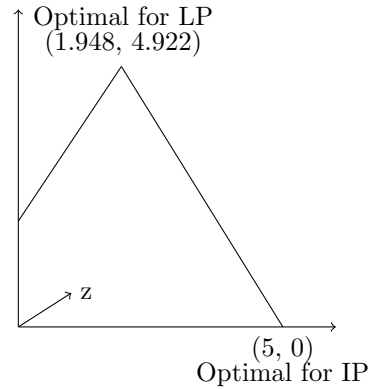


Figure 7.1: Optimal solution for LP / IP

7.3.2 Why Rounding Can be Bad - QAP example

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

$$\min \quad z = \sum_{i \in D} \sum_{s \in O} c_{is} x_{is} + \sum_{i \in D} \sum_{j \in D} \sum_{s \in O} \sum_{t \in O} w_{ij}^{st} y_{ij}^{st} \quad (7.55)$$

$$\text{s.t.} \quad \sum_{i \in D} x_{is} = 1, \quad s \in D \quad (7.56)$$

$$\sum_{s \in O} x_{is} = 1, \quad i \in D \quad (7.57)$$

$$x_{is} \in \{0, 1\}, \quad i \in D, s \in O \quad (7.58)$$

$$y_{ij}^{st} \geq x_{is} + x_{jt} - 1, \quad i \in D, j \in D, s \in O, t \in O \quad (7.59)$$

$$y_{ij}^{st} \geq 0, \quad i \in D, j \in D, s \in O, t \in O \quad (7.60)$$

$$y_{ij}^{st} \leq x_{is}, \quad i \in D, j \in D, s \in O, t \in O \quad (7.61)$$

$$y_{ij}^{st} \leq x_{jt}, \quad i \in D, j \in D, s \in O, t \in O \quad (7.62)$$

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

$$x_{is} = \frac{1}{|D|}, \quad i \in D, s \in O \quad (7.63)$$

$$y_{ij}^{st} = 0, \quad i \in D, j \in D, s \in O, t \in O \quad (7.64)$$

7.3.3 IP and Convex Hull

For IP problem

$$Z_{IP} \quad \max \quad z = cx \quad (7.65)$$

$$\text{s.t. } Ax \leq b \quad (7.66)$$

$$x \in Z^n \quad (7.67)$$

In feasible region $S = \{x \in Z^n, Ax \leq b\}$, the optimal solution $Z_{IP} = \max\{cx : x \in S\}$.

Denote $\text{conv}(S)$ as the convex hull of S then

$$Z_{IP}(S) = Z_{IP}(\text{conv}(S)) \quad (7.68)$$

7.3.4 Local Optimal and Global Optimal

Let

$$Z_s = \min\{f(x) : x \in S\} \quad (7.69)$$

$$Z_t = \min\{f(x) : x \in T\} \quad (7.70)$$

$$S \subset T \quad (7.71)$$

then

$$Z_t \leq Z_s \quad (7.72)$$

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

$$\begin{cases} x_T^* \in \arg \min\{f(x) : x \in T\} \\ x_T^* \in S \end{cases} \quad (7.73)$$

$$\Rightarrow x_T^* \in \arg \min\{f(x) : x \in S\} \quad (7.74)$$

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

7.3.5 LP Relaxation

To perform the LP relaxation, we expand the feasible region

$$x \in \{0, 1\} \rightarrow x \in [0, 1] \quad (7.75)$$

$$y \in Z^+ \rightarrow y \geq 0 \quad (7.76)$$

If we have $Z_{LP}(s) = \text{conv}(s)$ then

$$LP(s) : x \in R_+^n : Ax \leq b \quad (7.77)$$

The closer $LP(s)$ is to $\text{conv}(s)$ the better. Interestingly, we only need to know the convex in the direction of c .

There are several formulation problem have the property of $Z_{LP}(s) = \text{conv}(s)$, such as:

- Assignment Problem
- Spanning Tree Problem
- Max Flow Problem
- Matching Problem

Chapter 8

Decomposition Principle

Chapter 9

Ellipsoid Algorithm

Chapter 10

Projective Algorithm

Chapter 11

Interior-Point Algorithm

Part III

Graph and Network Theory

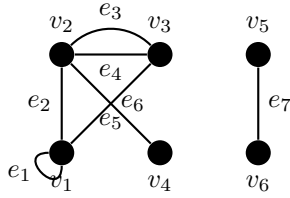
Chapter 12

Graphs and Subgraphs

12.1 Graph

Definition 12.1.1 (Graph). A **graph** G consists of a finite set $V(G)$ on vertices, a finite set $E(G)$ on edges and an **incident relation** than associates with any edge $e \in E(G)$ an unordered pair of vertices not necessarily distinct called **ends**.

Example. The following graph



can be represented as

$$V = V(G) = \{v_1, v_2, v_3, v_4, v_5, v_6\} \quad (12.1)$$

$$E = E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\} \quad (12.2)$$

$$e_1 = v_1v_1, \quad e_2 = v_2v_4, \quad \dots \quad (12.3)$$

Definition 12.1.2 (Loop, Parallel, Simple Graph). An edge with identical ends is called a **loop**. Two edges having the same ends are said to be **parallel**. A graph without loops or parallel edges is called **simple graph**.

Definition 12.1.3 (Adjacent). Two edges of a graph are **adjacent** if they have a common end, two vertices are **adjacent** if they are joined by an edge.

12.2 Graph Isomorphism

12.3 The Adjacency and Incidence Matrices

12.4 Subgraph

Definition 12.4.1 (Subgraph). Given two graphs G and H , H is a **subgraph** of G if $V(H) \subseteq V(G)$, $E(H) \subseteq$

$E(G)$ and an edge has the same ends in H as it does in G . Furthermore, if $E(H) \neq E(G)$ then H is a proper subgraph.

Definition 12.4.2 (Spanning). A subgraph H on G is **spanning** if $V(H) = V(G)$.

Definition 12.4.3 (Vertex-induced, Edge-induced). For a subset $V' \subseteq V(G)$ we define an **vertex-induced** subgraph $G[V']$ to be the subgraph with vertices V' and those edges of G having both ends in V' . The **edge-induced** subgraph $G[E']$ has edges E' and those vertices of G that are ends to edges in E' .

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

12.5 Degree

Definition 12.5.1 (Degree). Let $v \in V(G)$, then the **degree** of $v \in V(G)$ denote by $d_G(v)$ is defines to be the number of edges incident of v . Loops counted twice.

Theorem 12.1. For any graph $G = (V, E)$

$$\sum_{v \in V} d(v) = 2|E| \quad (12.4)$$

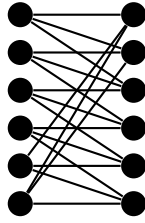
Proof. \forall edge $e = uv$ with $u \neq v$, e is counted once for u and once for v , a total of two altogether. If $e = uu$, a loop, then it is counted twice for u . \square

Problem 12.1. Explain clearly, what is the largest possible number of vertices in a graph with 19 edges and all vertices of degree at least 3. Explain why this is the maximum value.

Solution. The maximum number is 12.

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

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



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

$$2|E| = \sum_{v \in V} d(v) \geq 3 \times |N| = 3 \times 13 \Rightarrow |E| \geq 19.5 > 19 \quad (12.5)$$

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

□

Corollary 12.1.1. *Every graph has an even number of odd degree vertices.*

Proof.

$$V = V_E \cup V_O \Rightarrow \sum_{v \in V} d(v) = \sum_{v \in V_E} d(v) + \sum_{v \in V_O} d(v) = 2|E| \quad (12.6)$$

□

12.6 Special Graphs

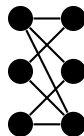
Definition 12.6.1 (Complete Graph). A **complete** graph $K_n (n \geq 1)$ is a simple graph with n vertices and with exactly one edge between each pair of distinct vertices.

Definition 12.6.2 (Cycle). A **cycle** graph $C_n (n \geq 3)$ consists of n vertices v_1, \dots, v_n and n edges $\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{n-1}, v_n\}$

Definition 12.6.3 (Wheel). A **wheel** graph $W_n (n \geq 3)$ is a simple graph obtains by adding one vertex to the cycle graph C_n , and connecting this new vertex to all vertices of C_n

Definition 12.6.4 (Bipartite Graph). A simple graph is said to be **bipartite** if the vertex set can be expressed as the union of two disjoint non-empty subsets V_1 and V_2 such that every edges has one end in V_1 and another end in V_2

Example. Here is an example for bipartite graph



Definition 12.6.5 (Complete Bipartite Graph). The **complete bipartite graph** K_{mn} is the bipartite graph V_1 containing m vertices and V_2 containing n vertices such that each vertex in V_1 is adjacent to every vertex in V_2

Example. Here is an example for K_{53}



Theorem 12.2. (König Theorem) *A graph G is bipartite iff every cycle is even.*

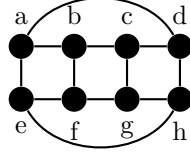
Proof. Hereby we prove the \Rightarrow and \Leftarrow

- (\Rightarrow) If the graph G is bipartite, by definition, the vertices of graph can be partition into two groups, that within the group there is no connection between vertices. Therefore, for each cycle, the odd index of vertices and even index of vertices has to be choose alternatively from each groups. Therefore the cycle has to be even.
- (\Leftarrow) Prove by contradiction. A graph can be connected or not connected.
 - If G is connected and has at least two vertices, for an arbitrary vertex $v \in V(G)$, we can calculate the minimum number of edges between the other vertices v' and v (i.e., length, denoted by $l(v', v)$), for all the vertices that has odd length to v , assign them to set V_1 , for the rest of vertices (and v), assign to set V_2 . Assume that G is not bipartite, which means there are at least one edge between distinct vertices in set V_1 or set V_2 , without lost of generality, assume that edge is uw , $u, w \in V_1$. For all vertices in V_1 there is an odd length of path between the vertex and v , therefore, there exists an odd $l(u, v)$, and an odd $l(w, v)$. The length of cycle $l(u, w, v) = 1 + l(u, v) + l(w, v)$, which is an odd number, it contradict with the prerequisite that all cycles are even, which means the assumption that G is not bipartite is incorrect, G should be bipartite.
 - If G is not connected. Then G can be partition into a set of disjointed subgraphs which are connected with at least two vertices or contains only one vertex. For the subgraph that has more that one vertices, we already proved that it has to be bipartite. For the subgraph

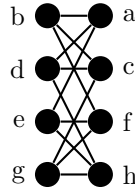
$G_i \subset G, i = 1, 2, \dots, n$, the vertices can be partition into $V_{i1} \in V(G_i)$ and $V_{i2} \in V(G_i)$, where $V_{i1} \cap V_{i2} = \emptyset$, the union of those subgraphs are bipartite too because $V_1 = \cup_{i=1}^n V_{i1} \in V(G)$ and $V_2 = \cup_{i=1}^n V_{i2} \in V(G)$ satisfied the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

□ **Definition 12.7.2.** We call directed graphs **digraphs**, we call edges in a digraph are called **arcs**, and vertices in a digraph **nodes**

Example. The following graph is bipartite, it only contains even cycles.

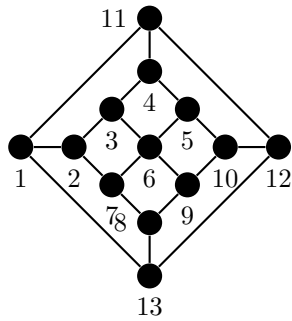


We can rearrange the graph to be more clear as following



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

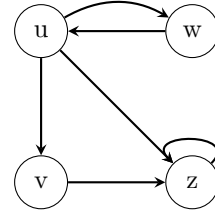
Example. The following graph is not bipartite



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

12.7 Directed Graph

Definition 12.7.1. A graph $G = (V, E)$ is called directed if for each edge $e \in E$, there is a **head** $h(e) \in V$ and a **tail** $t(e) \in V$ and the edges of e are precisely $h(e)$ and $t(e)$, denoted $e = (t(e), h(e))$



Definition 12.7.3. Similar as in the undirected case we have walks, traces, paths and cycles in digraphs.

Definition 12.7.4. A vertex $v \in V$ is **reachable** from a vertex $u \in V$ if there is a (u, v) -dipath. If at the same time u is reachable from v , they are **strongly connected**

Definition 12.7.5. A digraph is strongly connected if every pair of vertices are strongly connected.

Definition 12.7.6. A digraph is **strict** if it has no loops and whenever e and f are parallel, $h(e) = t(f)$

Definition 12.7.7. For a vertex v in a digraph D , the **indegree** of v in D , denoted by $d^+(v)$ is the number of arcs of D having head V . The **outdegree** of v is denoted by $d^-(v)$ is the number of arcs of D having tail v .

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

$$n_{ij} = \begin{cases} 1, & \text{if } v_i = h(e_j) \\ -1, & \text{if } v_i = t(e_j) \\ 0, & \text{otherwise} \end{cases} \quad (12.7)$$

$$\begin{bmatrix} -1 & 0 & -1 & -1 & 1 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \quad (12.8)$$

12.8 Sperner's Lemma

Chapter 13

Paths, Trees, and Cycles

13.1 Walk

Definition 13.1.1 (walk). A **walk** in a graph G is a finite sequence $w = v_0e_1v_1e_2...e_kv_k$, where for each $e_i = v_{i-1}v_i$ the edge and its ends exists in G . We say that walk v_0 to v_k on (v_0, v_k) -walk.

Example.

$$w = v_2e_4v_3e_4v_2e_5v_3 \quad (13.1)$$

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

Definition 13.1.2 (origin, terminal, internal, length). For (v_0, v_k) -walk, The vertices v_0 and v_k are called the **origin** and the **terminal** of the walk w , $v_1..v_{k-1}$ are called **internal** vertices. The integer k is the **length** of the walk. Length of w equals to the number of edges.

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

$$w^{-1} = v_ke_kv_{k-1}e_{k-1}...e_2v_1 \quad (13.2)$$

(The reverse walk is guaranteed to exist because it is an undirected graph)

Given two walks w and w' we can create a third walk denoted by ww' by concating w and w' . The new walk's origin is the same as terminal.

13.2 Path and Cycle

Definition 13.2.1 (trail). A **trail** is a walk with no repeating edges. e.g., $v_3e_4v_2e_5v_3$

Definition 13.2.2 (path). A **path** is a trail with no repeating vertices. e.g., $v_3e_4v_2$

Notice: Paths \subseteq Trails \subseteq Walks

Definition 13.2.3 (closed, cycle). A path is **closed** if it has positive length and its origin and terminal are the same. e.g., $v_1e_2v_2e_4v_3e_3v_1$. A closed trail where origin and internal vertices are distinct is called a **cycle** (The only time a vertex is repeated is the origin and terminal)

Definition 13.2.4 (even/odd cycle). A cycle is **even** if it has a even number of edges otherwise it is **odd**.

Problem 13.1. Prove that if C_1 and C_2 are cycles of a graph, then there exists cycles $K_1, K_2, ..., K_m$ such that $E(C_1)\Delta E(C_2) = E(K_1) \cup E(K_2) \cup ... \cup E(K_m)$ and $E(K_i) \cap E(K_j) = \emptyset, \forall i \neq j$. (For set X and Y , $X\Delta Y = (X - Y) \cup (Y - X)$, and is called the symmetric difference of X and Y)

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

$$C_1 = v_{11}e_{11}v_{12}e_{12}v_{13}e_{13}...v_{1n}e_{1n}v_{11} \quad (13.3)$$

$$C_2 = v_{21}e_{21}v_{22}e_{22}v_{23}e_{23}...v_{2k}e_{2k}v_{21} \quad (13.4)$$

Assume both cycle start at the same vertice, $v_{11} = v_{12}$. (If there is no intersected vertex for C_1 and C_2 , just simply set $K_1 = C_1$ and $K_2 = C_2$)

The following algorithm can give us all $K_j, j = 1, 2, ..., m$ by constructing $E(C_1)\Delta E(C_2)$. Also, the complexity is $O(mn)$, which makes the proof doable.

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

Require: Graph G , cycle C_1 and C_2

Ensure: $K_1, K_2, ...K_m$

```

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

Now we prove that $K_i \cap K_j = \emptyset, \forall i \neq j$. For each K_j , it is defined by two (v_o, v_t) -paths in the algorithm. From the algorithm we know that all the edges in (v_o, v_t) -path

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

Definition 13.2.5 (connected vertices). Two vertices u and v in a graph are said to be **connected** if there is a path between u and v .

Definition 13.2.6 (component). Connectivity between vertices is an equivalence relation on $V(G)$, if V_1, \dots, V_k are the corresponding equivalent classes then $G[V_1] \dots G[V_k]$ are **components** of G . If graph has only one component, then we say the graph is connected. A graph is connected iff every pair of vertices in G are connected, i.e., there exists a path between every pair of vertices.

Problem 13.2. If G is a simple graph with at least two vertices, prove that G has two vertices with the same degree.

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

- If G is connected, i.e., for all vertices, the degree is greater than 0. Also the graph is simple, for a graph with $|N|$ vertices, the degree of each vertex is less or equal to $|N| - 1$ (cannot have loop or parallel edge). For $|N|$ vertices, to make sure there is no two vertices that has same degree, it will need $|N|$ options for degrees, however, we only have $|N| - 1$ option. According to pigeon in holes principle, there has to be at least two vertices with the same degree.
- If G is not connected, i.e., the graph has more than one component. One of the following situation will happen:
 - For all components, each component contains only one vertex. Since we have at least two vertices, which means there are at least two component that has only one vertex. For those vertices, at least two vertices has the same degree as 0.
 - At least one component has more than one vertices. In this situation, we can find a component that has more than one vertices as a subgraph G' of the graph G . That G' is a connected simple graph by definition. We have already proved that a connected simple graph has two vertices with the same degree, which means G has two vertices with the same degree.

13.3 Tree and forest

Definition 13.3.1 (acyclic graph). A graph is called **acyclic** if it has no cycles

Definition 13.3.2 (forest, tree). A acyclic graph is called a **forest**. A connected forest is called a **tree**.

Theorem 13.1. Prove that T is a tree, if T has exactly one more vertex than it has edges.

Proof. 1. First we prove for any tree T that has at least two vertices, there has to be at least one leaf, i.e., now we prove that we can find u with degree of 1. Proof by constructing algorithm. (In fact we can prove that there are at least two leaves.)

Algorithm 3 Find one leaf in a tree

Require: $d(u) = 1$

Ensure: A tree T has at least one vertex

```

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

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

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

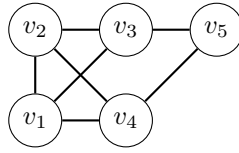
\square

13.4 Spanning tree

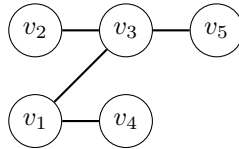
Definition 13.4.1 (spanning tree). A subgraph T of G is a **spanning tree** if it is spanning ($V(T) = V(G)$) and it is a tree.

\square

Example. In the following graph



This is a spanning tree



Problem 13.3. Prove that if T_1 and T_2 are spanning trees of G and $e \in E(T_1)$, then there exists a $f \in E(T_2)$, such that $T_1 - e + f$ and $T_2 + e - f$ are both spanning trees of G .

Proof. One of the following situation has to happen:

1. If for given $e \in E(T_1)$, $\exists f = e \in E(T_2)$, then $T_1 - e + f = T_1$, $T_2 + e - f = T_2$ are both spanning trees of G
2. If for given $e \in E(T_1)$, $e \notin E(T_2)$, the following will find an edge f that $T_1 - e + f$ and $T_2 + e - f$ are both spanning trees of G .
 - (a) T_1 is a spanning tree, removal of $e \in E(T_1)$ will disconnect the spanning tree into two components (by definition of spanning tree), denoted by $G_1 \subset G$ and $G_2 \subset G$, by definition, $V(G_1)$ and $V(G_2)$ is a partition of $V(G)$.
 - (b) Add e into T_2 . We can proof that by adding an edge into a tree will create exactly one cycle, denoted by C , $e \in E(C)$.
 - (c) For C , since it is a cycle and one end of e is in $V(G_1)$, the other end of e is in $V(G_2)$, there has to be at least two edges (can be more) that has one end in $V(G_1)$ and the other end in $V(G_2)$, denote the set of those edges as $E \subset E(C)$, one of those edges is $e \in E$
 - (d) Choose any $f \in E$ and $f \neq e$, for that f , $T_1 - e + f$ and $T_2 + e - f$ are both spanning trees of G .
 - (e) Prove that $T_1 - e + f$ is a spanning tree
 - i. $T_1 - e + f$ have the same set of vertices as T_1 , therefore it is spanning.
 - ii. It is connected both within G_1 and G_2 , for f , one end is in $V(G_1)$, the other end is in $V(G_2)$ therefore $T_1 - e + f$ is connected.

- iii. $T_1 - e + f$ have the same number of edges as T_1 , which is $|T_1| - 1$, therefore $T_1 - e + f$ is a tree. (We have proven the connectivity in the previous step.)
- iv. $T_1 - e + f$ is spanning, connected, a tree, therefore it is a spanning tree.

(f) Prove that $T_2 + e - f$ is a spanning tree

- i. $T_2 + e - f$ have the same set of vertices as T_2 , therefore it is spanning.
- ii. T_2 is connected, adding an edge will not break connectivity, therefore $T_2 + e$ is connected, removing an edge in a cycle will not break connectivity, therefore $T_2 + e - f$ is connected.
- iii. $T_2 - e + f$ have the same number of edges as T_2 , which is $|T_2| - 1$, therefore $T_2 + e - f$ is a tree. (We have proven the connectivity in the previous step.)
- iv. $T_2 - e + f$ is spanning, connected, a tree, therefore it is a spanning tree.

□

Theorem 13.2. Every connected graph has a spanning tree.

Proof. Prove by constructing algorithm: □

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

Require: a connected graph G and an enumeration e_1, \dots, e_m of the edges of G

Ensure: a spanning tree T of G

- 1: Let T be the spanning subgraph of G with $V(T) = V(G)$ and $E(T) = \emptyset$
- 2: $i \leftarrow 1$
- 3: **while** $i \leq |E|$ **do**
- 4: **if** $T + e_i$ is acyclic **then**
- 5: $T \leftarrow T + e_i$
- 6: $i \leftarrow i + 1$
- 7: **end if**
- 8: **end while**

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

For the complexity of spanning tree algorithm:

1. Space complexity, $2|E|$, which is $O(|E|)$
2. Time complexity
 - (a) How to check for acyclic?
 - i. At every stage T has certain components V_1, \dots, V_t , (every time we add an edge, the number of components minus 1)

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

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

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

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

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

13.5 Cayley's Formula

13.6 Connectivity

13.7 Blocks

Chapter 14

Euler Tours and Hamilton Cycles

14.1 Euler Tours

14.2 Hamilton Cycles

Chapter 15

Planarity

15.1 Plane and Planar Graphs

15.2 Dual Graphs

15.3 Euler's Formula

15.4 Bridges

15.5 Kuratowski's Theorem

15.6 Four-Color Theorem

15.7 Graphs on other surfaces

Chapter 16

Minimum Spanning Tree Problem

16.1 Basic Concepts

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

Definition 16.1.1 (Cut vertex). A vertex v of a connected graph G is a **cut vertex** if $G \setminus v$ is not connected.

Definition 16.1.2 (Connection problem). Given a connected graph G and a positive cost C_e for each $e \in E$, find a minimum-cost spanning connected subgraph of G . (Cycles all allowed)

Lemma 16.1. An edge $e = uv \in G$ is an edge of a cycle of G iff there is a path $G \setminus e$ from u to v .

Definition 16.1.3 (Minimum spanning tree problem). Given a connected graph G , and a cost $C_e, \forall e \in E$, find a minimum cost spanning tree of G

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

16.2 Kroskal's Algorithm

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

Require: A connected graph

Ensure: A MST

Keep a spanning forest $H = (V, F)$ of G , with $F = \emptyset$
while $|F| < |V| - 1$ **do**
 add to F a least-cost edge $e \notin F$ such that H remains a forest.
end while

16.3 Prim's Algorithm

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

Require: A connected graph

Ensure: A MST

Keep $H = (V(H), T)$ with $V(H) = \{v\}$, where $r \in V(G)$ and $T = \emptyset$

while $|V(T)| < |V|$ **do**

 Add to T a least-cost edge $e \notin T$ such that H remains a tree.

end while

16.4 Comparison between Kroskal's and Prim's Algorithm

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

16.5 Extensible MST

Definition 16.5.1 (cut). For a graph $G = (V, E)$ and $A \subseteq V$ we denote $\delta(A) = \{e \in E : e \text{ has an end in } A \text{ and an end in } V \setminus A\}$. A set of the form $\delta(A)$ for some A is called a **cut** of G .

Definition 16.5.2. We also define $\gamma(A) = \{e \in E : \text{both ends of } e \text{ are in } A\}$

Theorem 16.2. A graph $G = (V, E)$ is connected iff there is no $A \subseteq V$ such that $\emptyset \neq A \neq V$ with $\delta(A) = \emptyset$

Definition 16.5.3. Let us call a subset $A \in E$ **extensible** to a minimum spanning tree problem if A is contained in the edge set of some MST of G

Theorem 16.3. Suppose $B \subseteq E$, that B is extensible to an MST and that e is a minimum cost edge of some cut D satisfying $D \cap B = \emptyset$, then $B \cup \{e\}$ is extensible to an MST.

16.6 Solve MST in LP

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

$$X_e = \begin{cases} 1, & \text{if edge } e \text{ is in the optimal solution} \\ 0, & \text{otherwise} \end{cases} \quad (16.1)$$

The formulation is as following

$$\min \sum_{e \in E} c_e x_e \quad (16.2)$$

$$\text{s.t.} \quad \sum_{e \in E} x_e = |V| - 1 \quad (16.3)$$

$$x_e \geq 0 \quad (16.4)$$

$$e \in E \quad (16.5)$$

$$\sum_{e \in E(S)} x_e = |S| - 1, \forall S \subseteq V, S \neq \emptyset \quad (16.6)$$

$$(16.7)$$

Chapter 17

Shortest-Path Problem

17.1 Basic Concepts

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

Definition 17.1.1 (feasible potential). We call $y = (y_v : v \in V)$ a **feasible potential** if it satisfies

$$y_v + c_{vw} \geq y_w \quad \forall (v, w) \in E \quad (17.1)$$

and $y_r = 0$

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

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

$$C(P) = \sum_{i=1}^k C_{e_i} \geq \sum_{i=1}^k (y_{v_i} - y_{v_{i-1}}) = y_{v_k} - y_{v_0} \quad (17.2)$$

□

17.2 Breadth-First Search Algorithm

17.3 Ford's Method

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

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

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

Algorithm 7 Ford's Method

Ensure: Shortest Paths from r to all other nodes in V

Require: A digraph with arc costs, starting node r

Initialize, $y_r = 0$ and $y_v = \infty, v \in V \setminus r$

Initialize, $P(r) = 0, P(v) = -1, \forall v \in V \setminus r$

while y is not a feasible potential **do**

 Let $e = (v, w) \in E$ (this could be problematic)

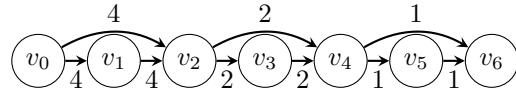
if $y_v + c_{vw} < y_w$ (incorrect) **then**

$y_w \leftarrow y_v + c_{vw}$ (correct it)

$P(w) = v$ (set v as predecessor)

end if

end while



17.4 Ford-Bellman Algorithm

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

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

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

For Bellman, will either terminate with shortest path from r to all $v \in V \setminus r$ or it will terminate stating that there is a negative cycle. In $O(mn)$

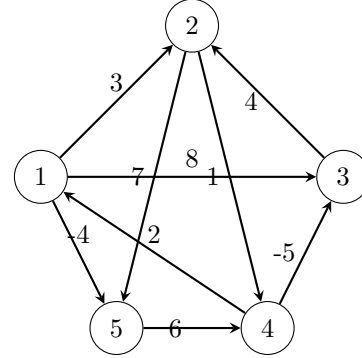
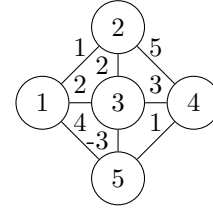
In the algorithm if $i = n$ and there exists a feasible potential, the problem has a negative cycle.

Suppose that the nodes of G can be ordered from left to right so that all arcs go from left to right. That is suppose there is an ordering $v_1, v_2, \dots, v_n \in V$ so that $(v_i, v_j) \in E$ implies $i < j$. We call such an ordering **topological sort**.

If we order E in the sequence that $v_i v_j$ precedes $v_k v_l$ if $i < k$ based on topological order then ford algorithm will terminate in one pass.

Algorithm 8 Ford-Bellman Algorithm**Ensure:** Shortest Paths from r to all other nodes in V **Require:** A digraph with arc costs, starting node r Initialize y and p **for** $i = 0; i < N; i++$ **do** **for** $\forall e = (v, w) \in E$ **do** **if** $y_v + c_{vw} < y_w$ (incorrect) **then** $y_w \leftarrow y_v + c_{vw}$ (correct it) $P(w) = v$ (set v as predecessor) **end if** **end for****end for****for** $\forall e = (v, w) \in E$ **do** **if** $y_v + c_{vw} < y_w$ (incorrect) **then**

Return error, negative cycle

end if**end for****17.5 SPFA Algorithm****17.6 Dijkstra Algorithm****Algorithm 9** Dijkstra Algorithm**Ensure:** Shortest Paths from r to all other nodes in V **Require:** A digraph with arc costs, starting node r Initialize y and p $S \leftarrow V$ **while** $S \neq \emptyset$ **do** Choose $v \in S$ with minimum y_v $S \leftarrow S \setminus v$ **for** $\forall w, (v, w) \in E$ **do** **if** $y_v + c_{vw} < y_w$ (incorrect) **then** $y_w \leftarrow y_v + c_{vw}$ (correct it) $P(w) = v$ (set v as predecessor) **end if** **end for****end while**

Let d_{ij}^k denote the length of the shortest path from i to j such that all intermediate vertices are contained in the set $\{1, \dots, k\}$

In this case $d_{14}^5 = 5$

If the vertex k is not an intermediate vertex on p , then $d_{ij}^k = d_{ij}^{k-1}$, notice that $d_{15}^4 = -1$, node 4 is not intermediate, so $d_{15}^5 = -1$

If the vertex k is an intermediate on p , then $d_{ij}^k = d_{ik}^{k-1} + d_{kj}^{k-1}$, $d_{14}^5 = 0$ ($p = 1 \rightarrow 3 \rightarrow 5 \rightarrow 4$), i.e., $d_{14}^5 = d_{15}^4 + d_{54}^4 = 0$

Therefore $d_{ij}^k = \min\{d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}\}$

Input: graph $G = (V, E)$ with weight on edges Output: Shortest path between all pairs of vertices on existence of a negative cycle Step 1: Initialize

$$d_{ij}^0 = \begin{cases} c_{ij} & \text{distance from } i \text{ to } j \text{ if } (i, j) \in E \\ 0 & \text{if } i = j \\ \infty & \text{if } (i, j) \notin E \end{cases} \quad (17.3)$$

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

17.7 A* Algorithm**17.8 Floyd-Warshall Algorithm**

If all weights/distances in the graph are nonnegative then we could use Dijkstra within starting nodes being any one of the vertices of the graph. This method will take $O(n^3)$ If weight/distances are arbitrary and we would like to find shortest path between all pairs of vertices or detect a negative cycle we could use Bellman-Ford Algorithm with $O(n^4)$

We would like an algorithm to find shortest path between any two pairs in a graph for arbitrary weights (determined, negative, cycles) in $O(n^3)$

$$D^0 = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \quad (17.4)$$

$$\Pi^0 = \begin{bmatrix} & 1 & 1 & & 1 \\ & & & 2 & 2 \\ & 3 & & & \\ 4 & & 4 & & \\ & & & 5 & \end{bmatrix} \quad (17.5)$$

$$D^1 = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \mathbf{5} & -5 & 0 & \mathbf{-2} \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \quad (17.6)$$

$$\Pi^1 = \begin{bmatrix} & 1 & 1 & & 1 \\ & & & 2 & 2 \\ & 3 & & & \\ 4 & \mathbf{1} & 4 & & \mathbf{1} \\ & & & 5 & \end{bmatrix} \quad (17.7)$$

$$D^2 = \begin{bmatrix} 0 & 3 & 8 & \mathbf{4} & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \mathbf{5} & \mathbf{11} \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \quad (17.8)$$

$$\Pi^2 = \begin{bmatrix} & 1 & 1 & \mathbf{2} & 1 \\ & & & 2 & 2 \\ & 3 & & \mathbf{2} & \mathbf{2} \\ 4 & 1 & 4 & & 1 \\ & & & 5 & \end{bmatrix} \quad (17.9)$$

$$D^3 = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & \mathbf{-1} & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \quad (17.10)$$

$$\Pi^3 = \begin{bmatrix} & 1 & 1 & 2 & 1 \\ & & & 2 & 2 \\ & 3 & & 2 & 2 \\ 4 & \mathbf{3} & 4 & & 1 \\ & & & 5 & \end{bmatrix} \quad (17.11)$$

$$D^4 = \begin{bmatrix} 0 & 3 & \mathbf{-1} & 4 & -4 \\ \mathbf{3} & 0 & \mathbf{-4} & 1 & \mathbf{-1} \\ \mathbf{7} & 4 & 0 & 5 & \mathbf{3} \\ 2 & -1 & -5 & 0 & -2 \\ \mathbf{8} & \mathbf{5} & \mathbf{1} & 6 & 0 \end{bmatrix} \quad (17.12)$$

$$\Pi^4 = \begin{bmatrix} & 1 & \mathbf{4} & 2 & 1 \\ \mathbf{4} & & \mathbf{4} & 2 & \mathbf{1} \\ \mathbf{4} & 3 & & 2 & \mathbf{1} \\ 4 & 3 & 4 & & 1 \\ \mathbf{4} & \mathbf{3} & \mathbf{4} & 5 & \end{bmatrix} \quad (17.13)$$

$$D^5 = \begin{bmatrix} 0 & \mathbf{1} & \mathbf{-3} & \mathbf{2} & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix} \quad (17.14)$$

$$\Pi^5 = \begin{bmatrix} & \mathbf{3} & 4 & \mathbf{5} & 1 \\ 4 & & 4 & 2 & 1 \\ 4 & 3 & & 2 & 1 \\ 4 & 3 & 4 & & 1 \\ 4 & 3 & 4 & 5 & \end{bmatrix} \quad (17.15)$$

Time complexity $O(n^3)$

If during the previous processes, there exist an element of negative value in the diagonal, it means there exists negative cycle.

17.9 Johnson's Algorithm

Chapter 18

Maximum Flow Problem

18.1 Basic Concept

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

$$\max \quad v \quad (18.1)$$

$$\text{s.t.} \quad \sum_{h(e)=i} x_e - \sum_{t(e)=i} x_e = \begin{cases} -v, & \text{if } i = s \\ v, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases} \quad (18.2)$$

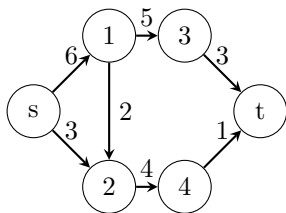
$$0 \leq x_e \leq u_e, \quad \forall e \in A \quad (18.3)$$

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

18.2 Solving Maximum Flow Problem in LP

Theorem 18.1. For $S \subseteq V$ we define (S, \bar{S}) to be a (s, t) -cut if $s \in S$ and $t \in \bar{S} = V - S$, the capacity of the cut, denoted $u(S, \bar{S})$ as $\sum \{u_e : e \in \delta^-(S)\}$ where $\delta^-(S) = \{e \in A : t(e) \in S \text{ and } h(e) \in \bar{S}\}$

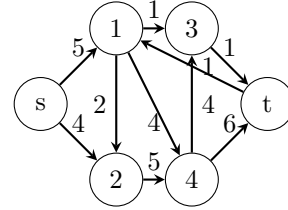
Example. For the following graph:



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

Definition 18.2.1. If (S, \bar{S}) has minimum capacity of all (s, t) -cuts, then it is called **minimum cut**.

Definition 18.2.2. Let $\delta^+(S) = \delta^-(V - S)$



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

Lemma 18.2. If x is a (s, t) flow of value v and (S, \bar{S}) is a (s, t) -cut, then

$$v = \sum_{e \in \delta^-(S)} x_e - \sum_{e \in \delta^+(S)} x_e \quad (18.4)$$

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

$$\sum_{i \in S} \left(\sum_{h(e)=i} x_e - \sum_{t(e)=i} x_e \right) = -v \quad (18.5)$$

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

$$\sum_{e \in \delta^+(S)} x_e - \sum_{e \in \delta^-(S)} x_e = -v \quad (18.6)$$

□

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

Corollary 18.2.1. If x is a feasible flow of value v , and (S, \bar{S}) is an (s, t) -cut, then

$$v \leq u(S, \bar{S}) \quad (\text{Weak duality}) \quad (18.7)$$

Definition 18.2.3. Define an arc e to be **saturated** if $x_e = u_e$, and to be **flowless** if $x_e = 0$

Corollary 18.2.2. Let x be a feasible flow and (S, \bar{S}) be a (s, t) -cut, if $\forall e \in \delta^-(S)$ is saturated, and $\forall e \in \delta^+(S)$ is flowless, then x is a maximum flow and (S, \bar{S}) is a minimum cut. (Strong duality)

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

$$\sum_{e \in \delta^-(S)} x_e = \sum_{e \in \delta^-(S)} u_e \quad (18.8)$$

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

$$\sum_{e \in \delta^+(S)} x_e = 0 \quad (18.9)$$

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

18.3 Prime and Dual of Maximum Network Flow Problem

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

$$\max \quad f = [0 \quad 0 \quad \cdots \quad 0 \quad 1] \begin{bmatrix} \mathbf{x} \\ f \end{bmatrix} \quad (18.10)$$

$$\text{s.t.} \quad [\mathbf{A} \quad \mathbf{F}] \begin{bmatrix} \mathbf{x} \\ f \end{bmatrix} = \mathbf{0} \quad (18.11)$$

$$\mathbf{I}\mathbf{x} \leq \mathbf{u} \quad (18.12)$$

$$\begin{bmatrix} \mathbf{x} \\ f \end{bmatrix} \geq 0 \quad (18.13)$$

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

$$\mathbf{A} = \mathbf{A}_{|E| \times |V|} = [a_{ij}], \text{ where } a_{ij} = \begin{cases} 1, & \text{if } v_i = h(e_j) \\ -1, & \text{if } v_i = t(e_j) \\ 0, & \text{otherwise} \end{cases} \quad (18.14)$$

$$\mathbf{F} = [1 \quad \cdots \quad 0 \quad \cdots \quad -1]^\top \quad (18.15)$$

$$\mathbf{u} = [u_1 \quad u_2 \quad \cdots \quad u_{|E|}]^\top \quad (18.16)$$

Then, we take the dual of LP

$$\min \quad \mathbf{u}\mathbf{w}_\mathbf{E} \quad (18.17)$$

$$\text{s.t.} \quad [\mathbf{w}_\mathbf{V} \quad \mathbf{w}_\mathbf{E}] \begin{bmatrix} \mathbf{A} \\ \mathbf{I} \end{bmatrix} \geq 0 \quad (18.18)$$

$$[\mathbf{w}_\mathbf{V} \quad \mathbf{w}_\mathbf{E}] \begin{bmatrix} \mathbf{F} \\ \mathbf{0} \end{bmatrix} = 1 \quad (18.19)$$

$$\mathbf{w}_\mathbf{V} \text{ unrestricted} \quad (18.20)$$

$$\mathbf{w}_\mathbf{E} \geq \mathbf{0} \quad (18.21)$$

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

$$\mathbf{w}_\mathbf{V} = [w_1 \quad w_2 \quad \cdots \quad w_{|V|}]^\top \quad (18.22)$$

$$\mathbf{w}_\mathbf{E} = [w_{|V|+1} \quad w_{|V|+2} \quad \cdots \quad w_{|V|+|E|}]^\top \quad (18.23)$$

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

18.4 Maximum Flow Minimum Cut Theorem

Definition 18.4.1. Let P be a path, (not necessarily a dipath), P is called **unsaturated** if every **forward** arc is unsaturated ($x_e < u_e$) and ever **reverse** arc has positive flow ($x_e > 0$). If in addition P is an (s, t) -path, then P is called an **x-augmenting path**

Theorem 18.3. A feasible flow x in a digraph D is maximum iff D has no augmenting paths.

Proof. (Prove by contradiction)

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

$$D_1 = \min\{u_e - x_e : e \text{ forward in } P\} \quad (18.24)$$

$$D_2 = \min\{x_e : e \text{ backward in } P\} \quad (18.25)$$

$$D = \min\{D_1, D_2\} \quad (18.26)$$

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

$$\hat{x}_e = \begin{cases} x_e + D & \text{if } e \text{ is forward in } P \\ x_e - D & \text{if } e \text{ is backward in } P \\ x_e & \text{otherwise} \end{cases} \quad (18.27)$$

It is easy to see that \hat{x} is feasible flow and that the value is $V + D$, a contradiction.

(\Leftarrow) Suppose D admits no x-augmenting path, Let S be the set of vertices reachable from s by x-unsaturated path clearly $s \in S$ and $t \notin S$ (because otherwise there would be an augmenting path). Thus, (S, \bar{S}) is a (s, t) -cut.

Let $e \in \delta^-(S)$ then e must be saturated. For otherwise we could add the $h(e)$ to S

Let $e \in \delta^+(S)$ then e must be flow less. For otherwise we could add the $t(e)$ to S .

According to previous corollary, that x is maximum. \square

Theorem 18.4. (*Max-flow = Minimum-cut*) For any digraph, the value of a maximum (s,t) -flow is equal to the capacity of a minimum (s,t) -cut

18.5 Ford-Fulkerson Method

Finding augmenting paths is the key of max-flow algorithm, we need to describe two functions, labeling and scanning a vertex.

A vertex is first labeled if we can find x -unsaturated path from s , i.e., (s,v) -unsaturated path.

The vertex v is scanned after we attempted to extend the x -unsaturated path.

This algorithm is incomplete/incorrect, needs to be fixed

Algorithm 10 Labeling algorithm

Ensure: Max-flow x with value v

Require: Digraph with source s and sink t , a capacity function u and a feasible flow (could be $x_e = 0$)

Initialize, $v \leftarrow x$

Designate all vertices as unlabeled and unscanned

Label s

while There exists vertex unlabeled or unscanned **do**

Let i be such a vertex, for each arc e with $t(e) = i$, $x_e < u_e$ and $h(e)$ unlabeled, label $h(e)$

For each arc e with $h(e) = i$, $x_e > 0$ and $t(e)$ unlabeled, label $t(e)$, designate i as scanned.

If t is not label

end while

x is the maximum.

Algorithm 11 Ford-Fulkerson algorithm

Ensure: Max-flow x with value v

Require: Digraph with source s and sink t , a capacity function u and a feasible flow (could be $x_e = 0$)

Initialize, $v \leftarrow x$

Designate all vertices as unlabeled and unscanned

Label s

while There exists vertex unlabeled or unscanned **do**

Let i be such a vertex, for each arc e with $t(e) = i$, $x_e < u_e$ and $h(e)$ unlabeled, label $h(e)$

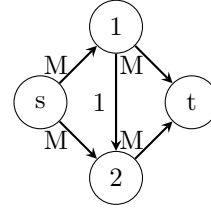
For each arc e with $h(e) = i$, $x_e > 0$ and $t(e)$ unlabeled, label $t(e)$, designate i as scanned.

If t is not label

end while

x is the maximum.

Labeling algorithm can be exponential, the following is an example



18.6 Polynomial Algorithm for max flow

Let (D, s, t, u) be a max flow problem and let x be a feasible flow for D , the **x-layers** of D are define be the following algorithm

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

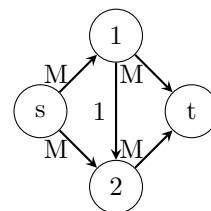
Step 1: Set $V_0 = \{s\}$, $i \leftarrow 0$ and $l(x) = 0$ Step 2: Let R be the set of vertices w such that there is an arc e with either:

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

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

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

Example. For the following graph



$$\begin{aligned}
& \text{Second iteration} & (18.28) \\
V_0 = \{s\}, i = 0, l(x) = 0 & (18.29) \\
R = \{1, 2\} & (18.30) \\
V_1 \leftarrow \{1, 2\}, i = 1, l(x) = 1 & (18.31) \\
R = \{3, 4, 5\} & (18.32) \\
V_2 \leftarrow \{3, 4\}, i = 2, l(x) = 2 & (18.33) \\
R = \{1, 5, 6, 3\} & (18.34) \\
V_3 \leftarrow \{5, 6\}, i = 3, l(x) = 3 & (18.35) \\
R = \{4, t\} & (18.36) \\
V_4 = \{t\} & (18.37) \\
A_1 = \{(s, 1), (s, 2)\} & (18.38) \\
A_2 = \{(1, 3), (2, 4)\} & (18.39) \\
A_3 = \{(3, 5), (4, 6)\} & (18.40) \\
A_4 = \{(5, t), (6, t)\} & (18.41)
\end{aligned}$$

The layer network D_x is defined by $V(D_x) = V_0 \cup V_1 \cup V_2 \cdots \cup V_{l(x)}$

Suppose we have computed the layers of D and $t \in V_l(x)$, the last layer (last layer I am going to V_e)

For each $i, 1 \leq i \leq l$, define a set of arcs A_i and a function \hat{u} on A_i as following. For each $e \in A(D)$

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

Let \hat{u} be the capacity function on D_x and let the source and sink of D_x be s and t

We can think of D_x as being made of arc shortest (in terms of numbers of arcs) x -augmenting paths.

A feasible flow in a network is said to be maximal (does not mean maximum) if every (s, t) -directed path contains at least one saturated arc.

For layered algorithm V_0, V_1, \dots, V_L

Arcs:

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

Maximal Flow: If every directed (s, t) -path has at least one saturated arc.

Computing maximal flow is easier than computing maximum flow, since we never need to consider canceling flows on reverse arcs,

Let \hat{x} be a maximal flow on the layered network D_x , we can define new flows in $D(x')$ by

$$x'_e = x_e + \hat{x}_e, \quad \text{If } t(e) \in V_{i-1}, h(e) \in V_i \quad (18.42)$$

$$x'_e = x_e - \hat{x}_e, \quad \text{If } h(e) \in V_{i-1}, t(e) \in V_i \quad (18.43)$$

18.7 Dinic Algorithm

Input: A layered network (D_x, s, t, \hat{u}) and a feasible flow x Output: A maximal flow \hat{x} from D_x

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

Theorem 18.5. *Dinic algorithm runs in $O(|E||V|^2)$*

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

Chapter 19

Minimum Weight Flow Problem

19.1 Transshipment Problem

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

$$\min \quad wx \quad (19.1)$$

$$\text{s.t.} \quad Nx = b \quad (19.2)$$

$$x \geq 0 \quad (19.3)$$

Where N is a vertex-arc incident matrix. For a feasible solution to LP to exist, the sum of all b s must be zero. Since the summation of rows of N is zero. The interpretation of the LP is as follows.

The variables are defined on the edges of the digraph and that x_e denote the amount of flow of some commodity from the tail of e to the head of e

Each constraints

$$\sum_{h(e)=i} x_e - \sum_{t(e)=i} x_e = b_i \quad (19.4)$$

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

19.2 Network Simplex Method

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

Lemma 19.1. *Let C_1 and C_2 be distinct cycles in a graph G and let $e \in C_1 \cup C_2$. Then $(C_1 \cup C_2) \setminus e$ contains a cycle.*

Proof. Case 1: $C_1 \cap C_2 = \emptyset$. Trivia.

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

Theorem 19.2. *Let T be a spanning tree of G . And let $e \in E \setminus T$ then $T + e$ contains a unique cycle C and for any edge $f \in C$, $T + e - f$ is a spanning tree of G*

Let (D, b, w) be a transshipment problem. A feasible solutions x is a **feasible tree solution** if there is a spanning tree T such that $\|x\| = \{e \in A, x_e \neq 0\} \subseteq T$.

For any tree T of D and for $e \in A \setminus T$, it follows from above theorem that $T + e$ contains a unique cycle. Denote that cycle $C(T, e)$ and orient it in the direction of e , define $w(T, e) = \sum\{w_e : e \text{ forward in } C(T, e)\} - \sum\{w_e : e \text{ reverse in } C(T, e)\}$.

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

Algorithm 12 Network Simplex Method Algorithm

Ensure: An optimal solution or the conclusion that (D, b, w) is unbounded

Require: A transshipment problem (D, b, w) and a feasible tree solution x containing to a spanning tree T

```

while  $\exists e \in A \setminus T, w(T, e) < 0$  do
  let  $e \in A \setminus T$  be such that  $w(T, e) < 0$ .
  if  $C(T, e)$  has no reverse arcs then
    Return unboundedness
  else
    Set  $\theta = \min\{x_f : f \text{ reverse in } C(T, e)\}$  and set
     $f = \{f \in C(T, e) : f \text{ reverse in } C(T, e), x_f = \theta\}$ 
    if  $f$  forward in  $C(T, e)$  then
       $x_f \leftarrow x_f + \theta$ 
    else
       $x_f \leftarrow x_f - \theta$ 
    end if
    Let  $f \in F$  and  $T \leftarrow T + e - f$ 
  end if
end while
Return  $x$  as optimal
  
```

The following is an example of cycling

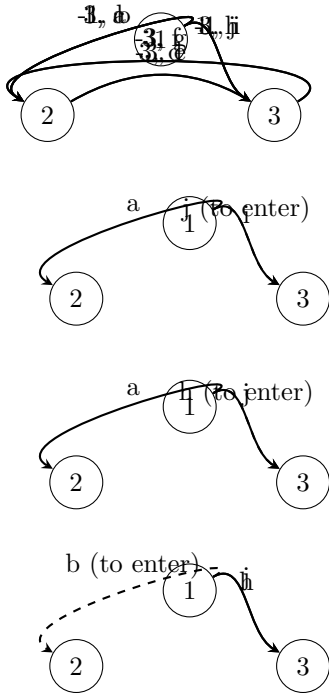
Then for

$$w(T, j) = w_j - w_i = -3 - 3 = -6$$

$$w(T, h) = w_h + w_j - w_i = 3 - 3 - 1 = -1$$

Keep going and we will find cycling.

To Avoid cycling we will introduce the modified network Simplex Method. Let T be a **rooted** spanning tree. Let

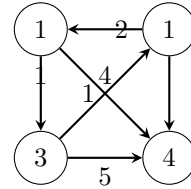


f be an arc in T , we say f is **away** from the root r if $t(f)$ is the component of $T - f$. Otherwise we say f is **towards** r .

Let x be a feasible tree solution associated with T , then we say T is a **strong feasible tree** if for every arc $f \in T$ with $x_f = 0$ then f is away from $r \in T$.

Modification to NSM:

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



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

Finding initial strong feasible tree.

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

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

Chapter 20

Matchings

20.1 Hall's "Marriage" Theorem

20.2 Transversal Theory

20.3 Menger's Theorem

20.4 The Hungarian Algorithm

Chapter 21

Colorings

21.1 Edge Chromatic Number

21.2 Vizing's Theorem

21.3 The Timetabling Problem

21.4 Vertex Chromatic Number

21.5 Brooks' Theorem

21.6 Hajós' Theorem

21.7 Chromatic Polynomials

21.8 Girth and Chromatic Number

Chapter 22

Independent Sets and Cliques

22.1 Independent Sets

22.2 Ramsey's Theorem

22.3 Turán's Theorem

22.4 Schur's Theorem

Chapter 23

Matroids