國 立 成 功 大 學
資 訊 管 理 研 究 所

碩 士 論 文

Maximum flows in distribution networks: problem generator and algorithms

指導教授: 王 逸 琳 博士

研 究 生: 林 筑 軍

中 華 民 國 九 十 四 年 七 月

國　立　成　功　大　學
資　訊　管　理　研　究　所
碩　士　論　文

Maximum flows in distribution networks: problem generator and algorithms

指導教授：王逸琳　博士
研究生　：林筑軍

中華民國九十四年七月

國立成功大學

碩士論文

Maximum flows in distribution networks: problem generator and algorithms

研究生：林筑軍

本論文業經審查及口試合格特此證明

論文考試委員：

陳 文 智	陳文智
李 宇 欣	李宇欣
許 瑞 麟	許瑞麟
王 逸 琳	王逸琳

指導教授： 王 逸 琳

系(所)主管： 王 泰 裕

中華民國 九十四 年 五 月 二十七 日

博碩士論文授權書

(國科會科學技術資料中心版本 93.2.6)

本授權書所授權之論文為本人在＿＿＿＿＿＿＿＿＿＿大學(學院)＿＿＿＿＿＿系所＿＿＿＿＿＿組＿＿＿＿＿＿學年度第＿＿＿學期取得＿＿＿士學位之論文。

論文名稱：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

□同意　□不同意

本人具有著作財產權之論文全文資料，授予行政院國家科學委員會科學技術資料中心(或其改制後之機構)、國家圖書館及本人畢業學校圖書館，得不限地域、時間與次數以微縮、光碟或數位化等各種方式重製後散布發行或上載網路。

本論文為本人向經濟部智慧財產局申請專利(未申請者本條款請不予理會)的附件之一，申請文號為：＿＿＿＿＿＿，註明文號者請將全文資料延後半年再公開。

□同意　□不同意

本人具有著作財產權之論文全文資料，授予教育部指定送繳之圖書館及本人畢業學校圖書館，為學術研究之目的以各種方法重製，或為上述目的再授權他人以各種方法重製，不限地域與時間，惟每人以一份為限。

上述授權內容均無須訂立讓與及授權契約書。依本授權之發行權為非專屬性發行權利。依本授權所為之收錄、重製、發行及學術研發利用均為無償。上述同意與不同意之欄位若未鉤選，本人同意視同授權。

指導教授姓名:

研究生簽名:　　學號:

(親筆正楷)　　(務必填寫)

日期:民國　　年　　月　　日

1.本授權書（得自http://sticnet.stic.gov.tw/sticweb/html/theses/authorize.html 下載或至http://www.stic.gov.tw首頁右下方下載）請以黑筆撰寫並影印裝訂於書名頁之次頁。

2.授權第一項者，請確認學校是否代收，若無者，請自行寄論文一本至台北市(106)和平東路二段 106 號 1702 室 國科會科學技術資料中心 黃善平小姐。(本授權書諮詢電話:02-27377606　傳真:02-27377689)

ABSTRACT

Maximum flows in distribution networks

by
Ju-Chun Lin

The maximum flow (max-flow) problem is a fundamental network optimization problem which computes for the largest possible amount of flow sent through the network from a source node to a sink node. This problem appears in many applications and has been investigated extensively over the recent four decades. Traditional max-flow problem may require some modification in its constraints to deal with real-world applications. Fang and Qi propose a new max-flow model, named as manufacturing network flow model, to describe a network with special distillation nodes or combination nodes. Based on this new model, Ting proposes a multi-labeling method to solve the max-flow problem in a distribution network which contains both ordinary and distillation nodes. The approach identifies an augmenting subgraph connecting both source and sink nodes which can be further decomposed into several components where flows in each component can be expressed by a single variable and solved by a system of homogeneous linear equations. The method requires manual detection for components and thus is not trivial to implement. In this paper, we present a preprocessing procedure that reduces the size of a distribution network in polynomial time and then give detailed procedures for implementing the multi-labeling method. We also implement a random network generator that generates random distribution networks guaranteed to be acyclic, connected and in a compact form for our computational testing. Modification on the preflow-push algorithm and the max-flow min-cut theorem is investigated and discussed to provide more theoretical insights as well.

Key words : Maximum flow; Distribution network

ACKNOWLEDGEMENTS

這份論文能完成，首先必須要感謝指導教授王逸琳老師，在研究上一直給予我們細心的指導，培養我們作學問該有的踏實態度，在此致上最誠摯的謝意。而論文口試期間，承蒙許瑞麟老師、李宇欣老師和陳文智老師不吝提供的諸多寶貴意見，亦讓學生受益良多。另外，我要感謝研究所的各位同學，兩年來為我帶來充滿歡笑與淚水的難忘時光，感謝可愛的學弟妹，讓研究室充滿了活力，最重要的，要感謝家人一路上對我的支持，還有這些圍繞在我身邊的好朋友們，讓我總是可以快樂地迎向未來的每一天。

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

CHAPTER I

INTRODUCTION

1.1 Background and motivation

In the competitive business environment nowadays, cooperative and competitive relationship among customers, retailers, distributors, manufacturers, and vendors in a supply chain becomes much more complicated so as to achieve a better supply chain management.

For example, suppose one unit of product A is made of two units of material B and one unit of material C. The relationship between materials and a product can be illustrated as a *product tree* (see Figure 1.1) suggested by Chen and Chern (2000). Material B and C may be purchased from different vendors by different channels. There may be several manufacturers who make product A with different production rates and obtain materials B and C via different channels.

We may integrate product trees with the supply chain network. Using Figure 1.1(a) and Figure 1.2 as an example, there are two manufactures (M_1 and M_2) and three vendors (V_1, V_2 and V_3) denoted by white circles. The product tree in a supply chain network can be represented by gray circles and half-circles which specify the process of decomposing products or semi-products into materials. The materials or semi-products are then purchasable from several vendors. For each outgoing arc of a

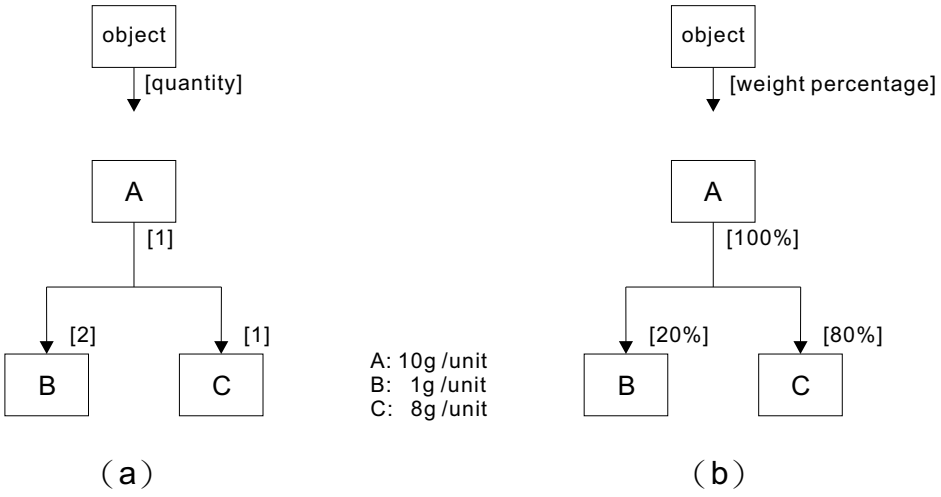


Figure 1.1: A product tree example (modified from Chen and Chern, 2000)

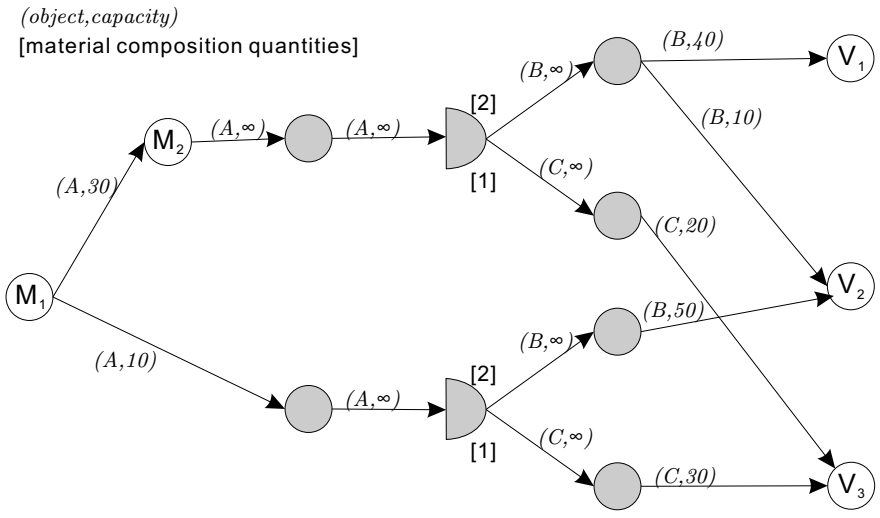


Figure 1.2: A supply chain example

gray half-circle, we associate a bracket with a number inside indicating the quantities of materials required for composing one unit of product (referring to Figure 1.1(a)). An arc connecting circles or half-circles may represent a process of production, decomposing, shipping, or outsourcing. We also associate a parenthesis for each arc which contains a letter in the left that indicates the object (i.e. material, semi-product, or product) to be processed and a number in the right representing the capacity (i.e. maximal number of objects processed) for the process. If a process has unlimited capacity, we use a "∞" to represent its capacity.

The scenario in Figure 1.2 is described as follows: Both M_1 and M_2 produce product A. The manager in M_1 will consider two strategies to fulfill the order: one is to produce products by his factory M_1, and the other is to outsource to M_2. One unit of A is made of two units of B and one unit of C. M_2 can purchase B from either V_1 or V_2, while M_1 can purchase B only from V_2.

In general, manufacturers either produce final products made of materials purchased from several vendors, or put part of their production out to contract with other manufacturers. Thus a final product may be made through several channels in different stages of a supply chain. Identifying the bottleneck (i.e. the set of channels that achieve their capacity and thus could not be improved) for a supply chain becomes an important issue to improve the entire efficiency. In other words, calculating the maximum throughput (i.e. product flow) for either the entire chain or part of the chain helps us to evaluate the chain's efficiency.

For example, to better understand the capability for the supply chain in Figure 1.2, the manager of M_1 could ask himself questions like–"At most how many units of A can I produce?, and in what way?" or "If I want to improve the maximum throughput of my production to a certain level, which processes should be improved?, and by

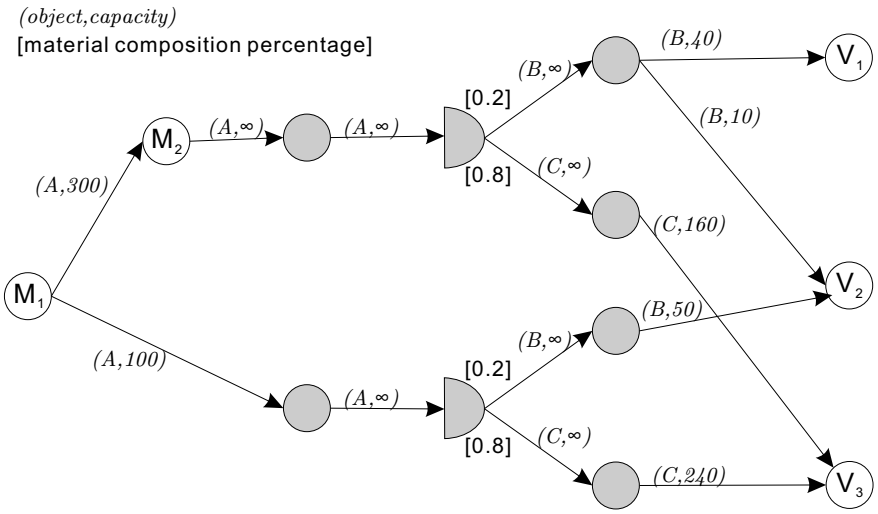


Figure 1.3: A supply chain example with unified units

how much?". The answers to these questions can be obtained by solving a maximum flow problem for this specific supply chain. However, the conventional maximum flow algorithms can not be directly applied to this supply chain since this problem differs from the conventional one by two points: first, units of flow are not unified in different arcs for this problem; and second, the amount of flow enters a gray half-circle has to be distributed by the predefined proportion.

To unify the flow units, we may use the weight instead of the quantities for different objects. In particular, the product tree in Figure 1.1(a) can be transformed into the product tree in Figure 1.1(b) since 10g of A (1 unit) can be made of 2g of B (2 units) and 8g of C (1 unit). Furthermore, we may integrate the distributed percentage of flow as shown in Figure 1.1(b) to transform Figure 1.2 into Figure 1.3 where the flow units are unified and the distributed flow percentage gives more intuitive illustration. For example, the capacity of arc (M_1,M_2) means M_2 can provide at most 300g of product A to M_1. To produce x g of product A, M_2 has to purchase $0.2x$ g of material B and $0.8x$ g of material C. Also, M_2 can purchase at most 40g and 10g of B from V_1 and

V_2, respectively. This transformed network thus only differs from the conventional one by the appearance of gray half-circles and the flow distribution constraints associated with their leaving arcs. Such a network model is called a *distribution network*, a special case of a manufacturing network flow model investigated by Fang and Qi (2003). In their model, a gray half-circle is called a *distillation* node or a *combination* node depending on its orientation.

This maximum flow example as shown in Figure 1.3 just illustrates possible relationships in a 2-stage supply chain. In practice, similar relationships and interactions among different stages in some supply chain do exist and make the problem much more complicated and difficult to solve.

To solve the maximum flow problem for a distribution network, Ting (2005) proposes an approach which identifies an augmenting subgraph connecting both source and sink nodes. The augmenting subgraph can be further decomposed into several components where flows in each component can be expressed by a single variable and solved by a system of linear equations. Their method requires manual detection for components and thus is difficult to implement by modern computers.

1.2 Objective and proposed approach

In this paper, we observe special properties for a distribution network and propose a method to compact the network which reduces the network size and simplifies the problem. Then we give detailed steps for Ting's approach. A random network generator that generates random graphs with random arc capacities and topologies is suggested and implemented for computational testing. Finally we investigate the possibility to modify the preflow-push algorithm (Goldberg and Tarjan, 1988) for solving the distribution network problem. We also observe a few properties which are possibly

related with the max-flow min-cut theorem for the optimal primal and dual solutions.

1.3 Scope and limitation

This research seeks efficient methods for solving the maximum flow problem on a special class of networks in which all the costs associated with nodes or arcs will not be considered. Instead, we focus on the capacity for arcs and the flow distribution percentage associated with each distillation node which forces the flow enters a distillation node to be distributed to its outgoing arcs by the predefined ratios. Our methodology is based on the following assumptions and limitations:

1. We assume no self-loop (i.e. an arc with the same end nodes) and parallel arcs (i.e. arcs with the same end nodes).

2. The flow distribution percentage associated with the leaving arcs of a distillation node must be a real number in $(0, 1)$, and their summation for the same distillation node must be 100%.

3. The flow in each arc has a zero lower bound and an integral upper bound (i.e. capacity). When we say an arc has unlimited capacity, we mean its capacity is a very large integer so that the arc will never be saturated in any case.

4. All the numbers associated with nodes and arcs are deterministic. That is, we only consider the case where the numbers are given beforehand and will not be changed at all times. Also, the network topology is fixed as given.

5. There is only one source node, but several sink nodes. The objective is to solve for the maximum flow the source node can send to all the sink nodes.

1.4 Overview of thesis

The rest of paper is organized as follows. Chapter 2 reviews the studies of the traditional and distributed maximum flow problems and algorithms. In Chapter 3, we propose a polynomial time network compaction algorithm and give detailed implementation for Ting's approach. A random network generator is implemented and discussed in Chapter 4. Experiences of modifying the preflow-push algorithm as well as observations on max-flow min-cut theorem are included in Chapter 5. Chapter 6 concludes this thesis, lists our contribution and gives suggestions for future research.

CHAPTER II

LITERATURE REVIEW

In this Chapter, we give notations and formulations for traditional maximum flow problems and distributed maximum flow problems, and then review their solution methods.

2.1 Notation and formulations

2.1.1 Traditional maximum flow problem

The traditional maximum flow problem was first formulated by Fulkerson and Dantzig (1955). Let $G = (N, A)$ be a directed graph with node set N and arc set A. We consider a capacitated network G where each arc (i, j) in A is associated with a nonnegative capacity u_{ij}. There are three types of nodes: S-node, T-node and O-node (see Figure 2.1). An S-node is a source node connected only by outgoing arcs. A T-node denotes a sink node connected only by incoming arcs. An O-node represents a transshipment node connected with both incoming and outgoing arcs. For each node i, we associate an integer $b(i)$ representing its supply/demand. In particular, $b(i) > 0$ if i is a supply node; $b(i) < 0$ if i is a demand node; and $b(i) = 0$ for each transshipment node i. Usually an S-node is a supply node, a T-node is a demand node, and any transshipment is an O-node.

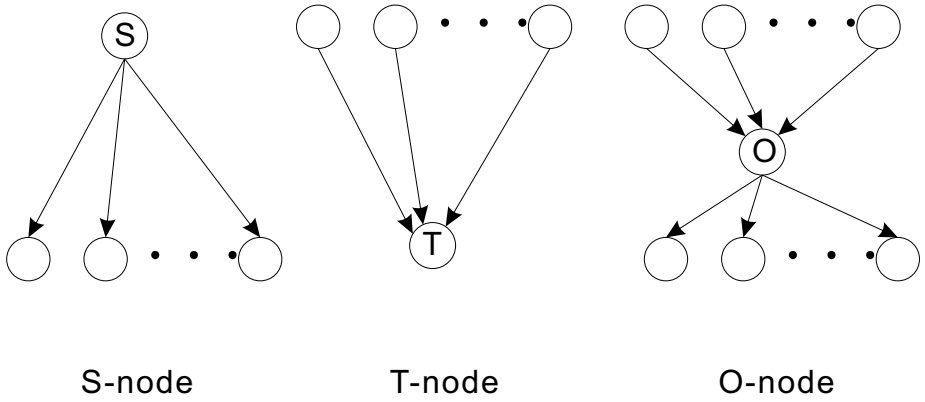


Figure 2.1: Illustration for an S-node, a T-node and an O-node (from Ting, 2004)

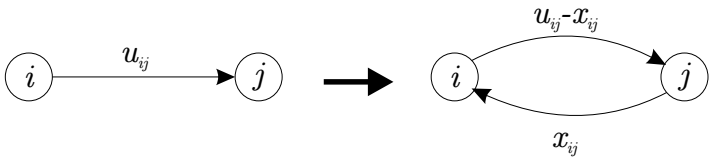


Figure 2.2: Constructing the residual network $G(x)$ (from Ahuja et al, 1993)

A network flow model has to obey the constraints for flow balance and bounds. In particular, for each node i, the amount of flow enters i minus the amount of flow leaves i should equal to b_i. Also, the flow passing through an arc (i, j) can not exceed its capacity u_{ij}, but must be more than its lower bound l_{ij}. In this thesis we assume $l_{ij} = 0$ for each arc (i, j). The traditional maximum flow problem seeks the maximum amount of flow, denoted by v, that is shipped from the S-node to the T-nodes while the flow assignment satisfies the arc capacities and flow balance constraints for all arcs and nodes. Let S, T, and O denote the set of S-nodes, T-nodes, and O-nodes, respectively.

Most maximum flow algorithms are applied on an induced graph $G(x)$ named as the *residual network* for a given flow vector x (see Figure 2.2). In particular, for each arc (i, j) with flow x_{ij}, we replace (i, j) by two arcs, (i, j) and (j, i) with *residual capacities* $r_{ij} = u_{ij} - x_{ij}$ and $r_{ji} = x_{ij}$, respectively. An arc $(i, j) \in A$ is *saturated* when $r_{ij} = 0$. The residual network only contains arcs with positive residual capacities. An

augmenting path from s to t is a path connecting s and t in the residual network. Therefore, if there exists an augmenting path in $G(x)$ from an S-node to a T-node, it means more flow can be shipped along this augmenting path and thus the current flow vector x is not optimal.

The traditional maximum flow problem can be formulated as follows:

$$\max \sum_{i \in S} b_i \tag{2.1}$$

$$s.t. \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = b_i \begin{cases} \geq 0 & \forall\ i \in S \\ = 0 & \forall\ i \in O \\ \leq 0 & \forall\ i \in T \end{cases} \tag{2.2}$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i,j) \in A \tag{2.3}$$

Assuming b_i and u_{ij} to be integral for each node i and arc (i,j), the optimal solution for this linear programming problem has to be integral due to the total unimodual property of the constraint matrix (1956). Algorithms to solve the traditional maximum flow problem will be surveyed in Section 2.2.1.

2.1.2 Distributed maximum flow problem

The distribution network model is first introduced by Fang and Qi (2003). Besides the three types of nodes (i.e. S-nodes, T-nodes, and O-nodes) as previously defined in Section 2.1.1, the distribution network model further introduces a type of distillation node, denoted by D-node, as the gray half-circle previously appeared in Figure 1.2.

A D-node i has only one incoming arc (l,i) and at least two outgoing arcs called *member arcs*. Each member arc (i,j) is associated with a positive real number, denoted by k_{ij}, to specify the percentage of the flow in (l,i) that is to be distributed into the

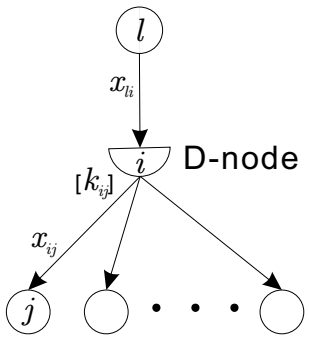
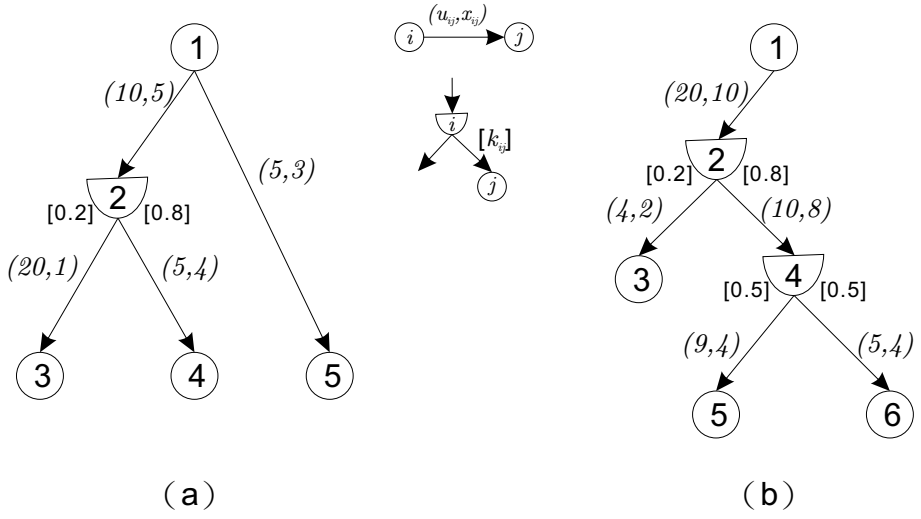


Figure 2.3: A D-node example (from Ting, 2004)

member arc (i, j). We also assume $\sum_{(i,j)\in A(i)} k_{ij} = 1 \quad \forall i \in D$ and $k_{ij} > 0$ for each member arc (i, j). Figure 2.3 illustrates a D-node i by half-circle. Let x_{ij} represents the flow in arc (i, j), then the flow along each member arc (i, j) for a D-node i with one incoming arc (l, i) can be calculated by $x_{ij} = k_{ij}x_{li}$. We call the relationship for flows on arcs connecting a D-node as "flow distillation", and k_{ij} as a *distillation factor*. For each member arc (i, j), we define a *normalized capacity* $\overline{u}_{ij} = u_{ij}/k_{ij}$ to represent the least amount of flow required in arc (l, i) to saturate arc (i, j).

A *D-family* is a group of nodes other than D-nodes adjacent to the same D-node. We call the node in a D-family that sends flow to the D-node as its *mother node*, and the nodes that receive flow from the D-node in a D-family as its *member nodes*. Within a D-family, the *mother arc* connects the mother node to the D-node, while a *member arc* connects the D-node to a member node. By definition of a D-node, there is only one D-node, one mother node, and at least two member nodes in a D-family. For example, nodes 1, 2, 3, and 4 in Figure 2.4(a) form a D-family in which node 1 is the mother node, nodes 3 and 4 are member nodes, arc $(1, 2)$ is the mother arc, and arcs $(2, 3)$ and $(2, 4)$ are member arcs.

For cases where a D-node is adjacent to anther D-node, by definition they can not form a D-family. We call the set containing the maximum number of adjacent



(a) (b)

Figure 2.4: Two examples of distribution networks

D-nodes as a *D-group*. For example nodes 2 and 4 in Figure 2.4(b) form a D-group.

The distribution maximum flow problem solves for the maximum amount of flow that the T-nodes can receive from the S-node as long as the flow obeys the constraints of conservation, bounds, and distillation in a distribution network. We denote the set of D-nodes by D. A distributed maximum flow problem can be formulated as follows:

$$\max \sum_{i \in S} b_i \tag{2.4}$$

$$s.t. \sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = b_i \begin{cases} \geq 0 & \forall\ i \in S \\ = 0 & \forall\ i \in O \text{ or } D \\ \leq 0 & \forall\ i \in T \end{cases} \tag{2.5}$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i,j) \in A \tag{2.6}$$

$$x_{ij} = k_{ij} x_{li} \quad \forall (i,j) \in A,\ (l,i) \in A \text{ and } i \in D \tag{2.7}$$

Unlike the traditional maximum flow problem, assuming b_i and u_{ij} to be integral for each node i and arc (i,j) and k_{ij} to be rational for each member arc (i,j), the optimal solution for this linear programming problem is not guaranteed to be integral, since the property of total unimoduality has been affected by the flow distillation constraints.

We may view this problem as a traditional maximum flow problem with side constraints (i.e. flow distillation). This problem is still a linear programming problem and thus can be solved by any LP algorithms and software. We summarize the algorithms in the literature for solving the distributed maximum flow problem in Section 2.2.2.

2.2 Solution methods for maximum flow problems

Although the maximum flow problem is a LP which can be solved by any LP algorithm and software, specialized combinatorial algorithms are usually more efficient than the LP methods since they employ the specialized structure for network constraints. Here we review the combinatorial algorithms in the literature for solving the maximum flow problem.

2.2.1 Algorithms for traditional network model

Fulkerson and Dantzig (1955) formulate the maximum flow problem as a special minimum cost flow problem by introducing an additional arc (t, s) connecting the single sink node t to the single source node s with cost $c_{ts} = -1$, and then setting $c_{ij} = 0$ for all the original arcs (i, j). This specialized minimum cost flow problem will try to send flow via the arc (t, s) as much as possible to minimize the total cost, which means the flow from s to t via all the original arcs will be shipped as much as possible. This problem then can be solved by the network simplex method.

Ford and Fulkerson (1956) propose the first augmenting path algorithm to solve the maximum flow problem. Their idea is simple: as long as an augmenting path exists in the residual network, flow can be augmented along that path. The algorithm stops when no more augmenting path can be identified which means the maximum flow has been achieved.

The naive augmenting path algorithm only takes pseudo-polynomial time. Ed-

monds and Karp (1972) suggest an implementation which augments flows along a shortest path identified by a breadth-first-search algorithm beforehand in each iteration. Their algorithm runs in polynomial time. Similar idea to augment flows along a shortest path was also proposed by Dinic (1970) who introduces a shortest path network called *layered networks*. Sleator and Tarjan (1983) improve the complexity of Dinic's algorithm for sparse networks by dynamic tree data structure, which is the most attractive from a worse-case point of view.

Karzanov (1974) introduce the first preflow-push algorithm in a layered network. It violates the flow balance constraints at intermediate stages. That is, it permits the flow entering a node to exceed the flow leaving the node. When a node has a positive excess, it is marked as an active node, except for the source and the sink. All operations are performed using only local information. The goal of each iterative step is to choose an active node and to send its excess closer to the sink.

The algorithms by Goldberg and Tarjan (1988) are based on ideas similar to those presented in Karzanov (1974), but they use distance labels to direct flows closer to the sink instead of constructing layered networks. Their method maintains a preflow and proceeds by pushing flows to nodes that are closer to the sink. To estimate which nodes are closer to the sink, it maintains a distance label for each node that is a lower bound on the length of a shortest augmenting path to the sink.

Recently, Fuijshige (2003) propose a MA ordering algorithm, which can be regarded as an acceleration to the Edmonds-Karp algorithm (1972). This algorithm forms an induced acyclic subgraph rather than one augmenting path from the source to the sink, and then sends as much flow as possible in the subgraph by adapting Dijkstra's shortest path algorithm to find the maximum adjacent capacity for each node. The acyclic subgraph contains several augmenting paths.

Table 2.1: Summary of maximum flow algorithms

Algorithm	Complexity
Fulkerson and Dantzig	$O(n^2mU)$
Ford and Fulkerson	$O(nmU)$
Edmonds and Karp	$O(n^2m)$
Dinic	$O(n^2m)$
Sleator and Tarjan	$O(nm \log n)$
Karzanov	$O(n^3)$
Goldberg and Tarjan	$O(nm \log \frac{n^2}{m})$
Fuijshige	$O(n(m + n \log n) \log nU)$

n: the number of nodes, m: the number of arcs, $U = \max_{(i,j)\in A} |u_{ij}|$

(from Schrijver, 2003)

We summarize the complexity for these maximum flow algorithms in Table 2.1.

2.2.2 Algorithms for distribution network model

Although Fang and Qi (2003) firstly introduce the maximum flow problem in distributed networks, they do not provide its solution method. This maximum flow problem is more complicated and difficult to solve than the traditional one due to the flow distillation constraints. For example, suppose there exists an augmenting path connecting an S-node and T-node, but this path passes through a D-node. In such a case, we have to consider the D-node as another source node, and for each of its outgoing arcs we have to check whether there exists an augmenting path passing through that outgoing arc to a T-node. If we fail to find such an augmenting path from the D-node, we can not use the original augmenting path connecting an S-node and T-node, since sending flow via that path will force the D-node to send flows along all of its outgoing arcs and some of them will be "blocked out" somewhere in the network.

Ting (2005) proposes a *multi-labeling method* to solve this problem by adopting the concept of the Depth-First Search (DFS) which tries out every augmenting subgraph that goes to the sink or source and satisfies flow distillation constraints. Af-

ter finding such an augmenting subgraph, the algorithm then identifies components in the augmenting subgraph. Flow inside each component can be represented by a single variable. The flow balance constraints for all nodes that joint different components form a system of linear equations. Components joint with each other by nodes. Thus, the flow on an augmenting graph can be calculated by solving such a system of linear equations. The maximum flow can be calculated by iteratively identifying an augmenting subgraph, decomposing components, solving for variables associated with each component, and then calculating flows inside each component.

Ting's multi-labeling method enumerates all possible augmenting subgraphs in the residual networks. It is quite generic and requires manual detection for some procedures. In Chapter 3 we will give detailed implementation of this method.

2.3 Summary

Basic network notations, terminologies, and mathematical formulations about maximum flow problems in both the traditional network flow model and the distributed network model have been introduced. Literature survey on important maximum flow algorithms is also conducted and summarized. Finally we point out the difficulty to solve the distributed maximum flow problem by applying the conventional maximum flow algorithms, and summarize the recent development of solution methods for solving the distributed maximum flow problem.

CHAPTER III

NETWORK COMPACTION AND AUGMENTING PATH ALGORITHM

This Chapter give the details of our implementation for Ting's multi-labeling method (Ting, 2005). Here we first give a preprocessing procedure which compresses a distribution network to a more compact one and simplifies the problem. Then we propose detailed procedures and pseudo codes of our modified augmenting path algorithm.

3.1 Preprocessing

Any given distribution network may contain topology that is reducible. A preprocessing procedure can be applied to compact the original network to an equivalent one of smaller size, which simplifies network problems obviously. In addition, a compacted network would be beneficial to detect blocks along each member arc soon. As for every search algorithm applying in a distribution network, compacting networks may decrease the number of member arcs to speed up. This Section explains why solving the maximum flow on these two networks give the same answer and how the transformation can be conducted by a preprocessing procedure

3.1.1 Compacting rules

We observe that a distribution network may be further simplified by either merging several nodes or arcs, or by unifying the effect of the capacities for some arcs. In particular, we give five rules to detect whether a distribution network is compactible, and then explain the compacting procedures.

(C1) Compacting D-groups.

Nodes in the same D-group can be compacted into a D-node. A D-group contains the maximum set of adjacent D-nodes. Since flows over all arcs adjacent to a D-node are proportional to each other by some constants, we can easily calculate flow over an adjacent D-node which then can be used to calculate flows on all of its arcs. Therefore the arcs adjacent to a D-group have flows proportional to each other. The process of calculating flows and resetting capacities on all arcs adjacent to a D-group have the same result as merging all adjacent D-nodes. Thus an entire D-group can be replaced by a single representative D-node. Suppose there are q_r O-nodes (i_{q_1},i_{q_2},...,i_{q_r}) adjacent to a D-group with a representative D-node i_o. Suppose an O-node i_{q_w} is connected to i_o by a path $p_{i_o i_{q_w}}$ (i.e.$i_o \rightarrow ... \rightarrow i_{q_w}$) composed of $\left|p_{i_o i_{q_w}}\right|$ member arcs. The compacting process is as follows:

1. Among all the adjacent D-nodes, we retain the one (i_o) closest to the S-node and use i_o to represent the final merged D-node. For example, node 2 in Figure 3.1(a) is retained to be the representative D-node for that D-group.
2. Delete all original member arcs associated with this D-group and add new member arcs from $\{(i_o, i_{q_1}), (i_o, i_{q_2}), ..., (i_o, i_{q_r})\}$. In Figure 3.1(b), we create new arc $(2, 5)$ and arc $(2, 6)$.

3. The distillation factors on new member arcs can be calculated by multiplying the distillation factors of member arcs passing through intermediate D-nodes. In particular, $k_{i_o q_w} = \prod_{(i,j) \in p_{i_o q_w}} k_{ij}$. For example, the distillation factor for the new arc $(2, 5)$ in Figure 3.1(b) is $0.8 * 0.5 = 0.4$.

4. The capacities for the new member arcs can be calculated by

$$\min_{(y,y') \text{ lies on } p_{i_o w'},\ (w',q_w) \text{ lies on } p_{i_o q_w}} \left\{ u_{w' q_w},\ \prod_{(i,j) \text{ lise on } p_{y' q_w}} k_{ij} u_{yy'} \right\}$$

For example, the capacity of arc $(2, 5)$ in Figure 3.1(b) equals to $min\{u_{45}, k_{45}u_{24}\}$.

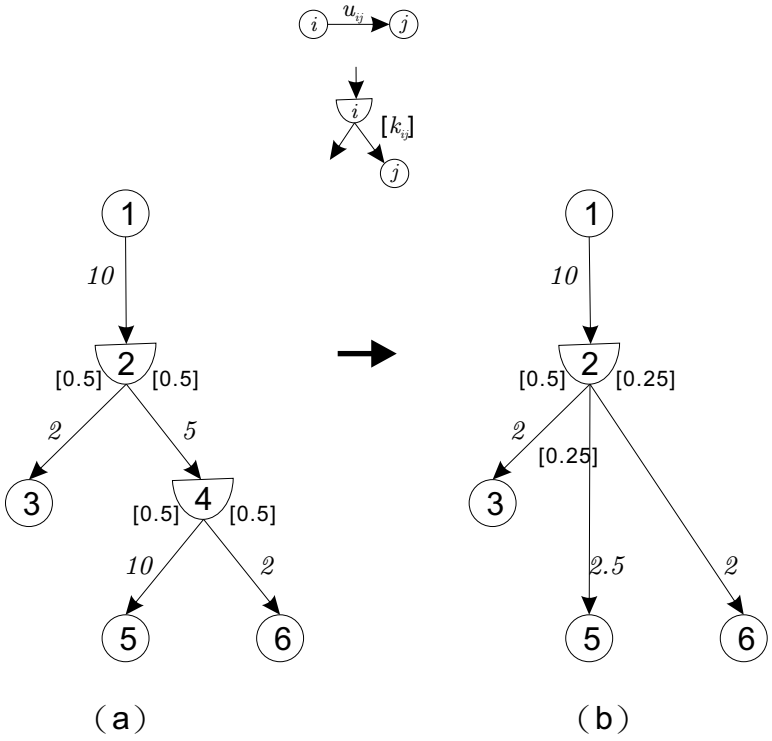


Figure 3.1: Adjacent D-nodes can be compacted into one D-node

Algebraically, flow distillation constraint $x_{45} = k_{45}x_{24}$ can be transformed into $x_{24} = \frac{x_{45}}{k_{45}}$, which can be plugged into the capacity constraint $x_{24} \leq u_{24}$. Thus we can combine the transformed constraint $x_{45} \leq k_{45}u_{24}$ with $x_{45} \leq u_{45}$ to set the new capacity for arc $(4, 5)$ to be $min\{u_{45}, k_{45}u_{24}\}$. This compacting process thus removes some constraint.

It takes $O(m)$ time to apply this compacting rule by a search algorithm. Each time when we compact a D-group, at least one D-node and one arc connecting D-nodes will be reduced.

(C2) Compacting single transshipment O-nodes.

A single transshipment O-node is an O-node with only one incoming arc and one outgoing arc, and such a node can be compacted. When an O-node with single incoming and outgoing arcs, it is simply used for transshipment and has no affection to the flows in those arcs. Thus we may remove this O-node and merge these two arcs into one single arc with new capacity equal to the minimum capacity of the original two arcs. Take the distribution network in Figure 3.2(a) for example, after removing node 4 and merging arc $(2, 4)$ and $(4, 5)$ by an arc$(2, 5)$ with capacity equal to $min\{8, 4\} = 4$, the reduced graph is shown in Figure 3.2(b). Algebraically, this operation is to use a new flow variable x_{25} to replace the original flow variables x_{24} and x_{45}. That is, $x_{25} = x_{24} = x_{45}$. The capacity constraints $x_{24} \leq u_{24}$ and $x_{45} \leq u_{45}$ can be integrated to set the new capacity for variable u_{25} to be $min\{u_{24}, u_{45}\}$. Thus we add one variable and one capacity constraint but remove two variables and two capacity constraints.

This compacting process takes $O(n)$ time since we only need to scan each node once. Each time when we compact a single transshipment O-node, this O-node and one arc connecting this O-node will be reduced.

(C3) Compacting self-loop arcs.

An arc with the same end nodes can be compacted. Although assuming no self-loop arcs in the network, they may be formed in some intermediate compacting process. Self-loop arcs are cycles in the network; therefore, they could be e-

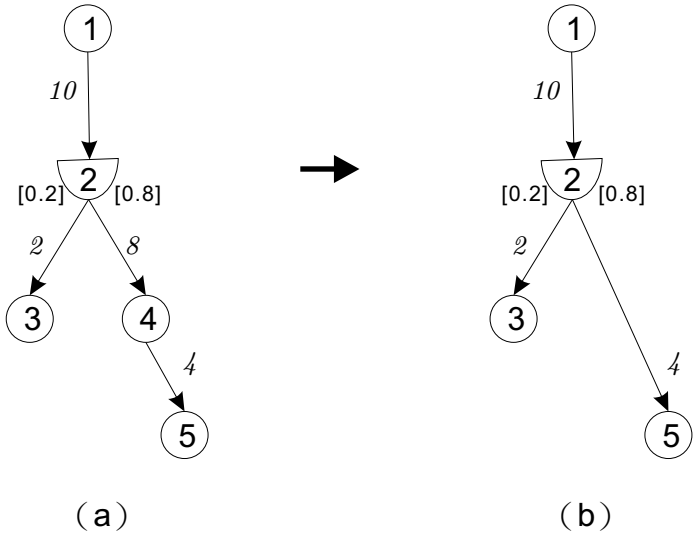


Figure 3.2: Any O-node with only one incoming arc and one outgoing arc can be compacted

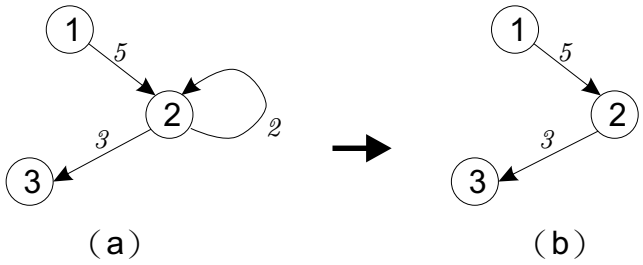


Figure 3.3: Self-loop arcs can be compacted

liminated totally. See Figure 3.3 for example. Algebraically, the flow balance constraint for node 2 is $x_{12} + x_{22} = x_{23} + x_{22}$, which equals to $x_{12} = x_{23}$.after eliminating x_{22}. Thus this compacting process removes one variable and takes $O(m)$ time by scanning all outgoing arcs for each node. Once we compact one self-loop arc, one arc will be reduced.

(C4) Compacting parallel arcs.

Parallel arcs connecting to the same end nodes can be compacted. Like self-loop arcs, although we assume no parallel arcs in the network, they may be formed in some intermediate compacting process as well. Since we may view those arcs

with the same tail and head nodes as one huge arc, we merge these parallel arcs into one new arc. There are two cases for parallel arcs. One case is that the parallel arcs connect to the same end O-nodes and we sum up their capacities to be the capacity of new arc. See Figure 3.4 for example. Algebraically, this operation is to use a new flow variable x_{12} to replace the original flow variables x'_{12} and x''_{12} by $x_{12} = x'_{12} + x''_{12}$. The capacity constraints $x'_{12} \leq u'_{12}$ and $x''_{12} \leq u''_{12}$ can also be integrated to be a new capacity $u'_{12} + u''_{12}$ for variable u_{12}. The other case is that the parallel arcs connect to the same D-node and O-node. In this case, we merge parallel member arcs into a single member arc and calculate its distillation factor by summing up the distillation factors of these parallel member arcs. Then, the new capacity can be calculated by finding out the minimum normalized capacity in these parallel member arcs and multiplying it with the new distillation factor. See Figure 3.5 for examples. The distillation factor of new arc $(2, 3)$ is $0.2 + 0.3 = 0.5$. The capacity of the new arc $(2, 3)$ will be $min\left\{\frac{1}{0.2}, \frac{3}{0.3}\right\} * 0.5 = 2.5$. Algebraically, this operation is to use a new flow variable x_{23} to replace the original flow variables x'_{23} and x''_{23} as well. By specifying $k_{23} = k'_{23} + k''_{23}$, we may write $x_{23} = x'_{23} + x''_{23} = k'_{23}x_{12} + k''_{23}x_{12} = (k'_{23} + k''_{23})x_{12} = k_{23}x_{12}$. Also, the capacity constraint for the new arc can be integrated to be $x_{23} \leq min\left\{\frac{u'_{23}}{k'_{23}}, \frac{u''_{23}}{k''_{23}}\right\} * k_{23}$. In either case, one new variable and constraint is added to replace two variables and constraints.

This compacting process takes $O(m)$ time by scanning all outgoing arcs for each node. Each time when we compact parallel arcs, at least one arc will be reduced.

(C5) Compacting mismatched capacities.

Any D-node connected with arcs of mismatched capacities can be compacted.

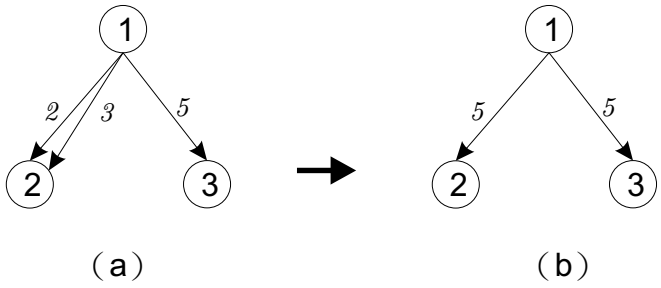


Figure 3.4: Parallel arcs connecting to the same end nodes can be compacted.

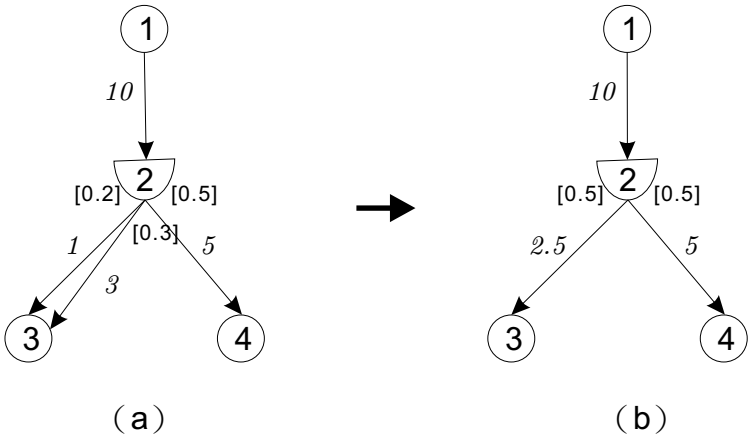


Figure 3.5: Parallel arcs connecting to the same D-node and O-node can be compacted.

According to the flow distillation constraints, the flow on each outgoing arc from a D-node is proportional to the flow on the incoming arc. Therefore we may easily identify the first saturated arc for a D-node by calculating the normalized capacity for each outgoing arc relative to the incoming arc. Take Figure 3.6(a) as an example, x_{12} has to be at least $\frac{20}{0.2} = 100$ to saturate arc $(2,3)$, and to saturate arc $(2,4)$. Thus the bottleneck happens on arc $(2,4)$ since it is the first arc to be saturated when we increase x_{12}. By this observation we can identify the minimum normalized capacity $\overline{u}_{\min}$ among all arcs adjacent to a D-node i, reset the capacity of its incoming arc (l,i) to be $\overline{u}_{\min}$, and then reset the capacity for each member arc (i,j) by $k_{ij}\overline{u}_{\min}$. For the example in Figure 3.6(a), $\overline{u}_{\min} = \min\left\{15, \frac{20}{0.2}, \frac{8}{0.8}\right\} = 10$. Thus arc $(2,4)$ is the bottleneck and we can reset $u_{12} = \overline{u}_{\min} = 10$ and $u_{23} = 0.2\overline{u}_{\min} = 2$ as shown in Figure 3.6(b). Algebraically, flow distillation constraints $x_{23} = k_{23}x_{12}$ and $x_{24} = k_{24}x_{12}$ can be plugged into the capacity constraints $x_{23} \leq u_{23}$ and $x_{24} \leq u_{24}$ to make $k_{23}x_{12} \leq u_{23}$ and $k_{24}x_{12} \leq u_{24.}$ Using these two constraints, together with the original capacity constraint $x_{12} \leq u_{12}$, we can set the new capacity for arc $(1,2)$ to be $min\left\{u_{12}, \frac{u_{23}}{k_{23}}, \frac{u_{24}}{k_{24}}\right\}$ and then reset the capacties for all the member arcs according to their distillation factors. This compacting process simplify the problem since once an arc adjacent to a D-node is saturated, all the other arcs will also be saturated. Thus a maximum flow algorithm may detect a bottleneck arc much earlier on this compacted network than the original one. This compacting process takes $O(m)$ time, but only needs to be conducted once in the end of the compacting process.

3.1.2 Inducing relations

These five compacting rules may induce each other. We list all possibilities as below.

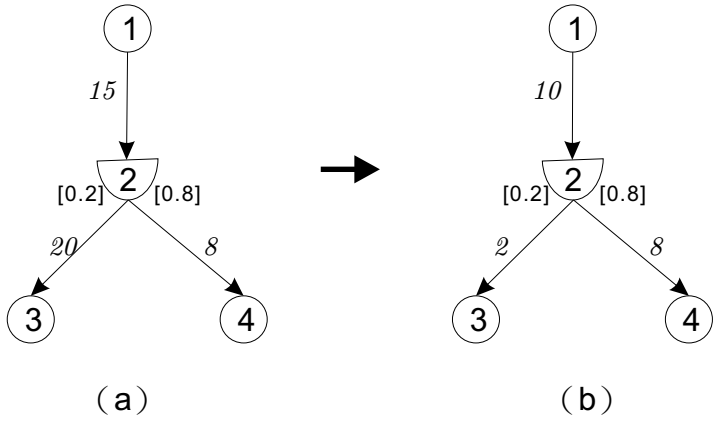


Figure 3.6: Any D-node with mismatch capacities of arcs can be compacted

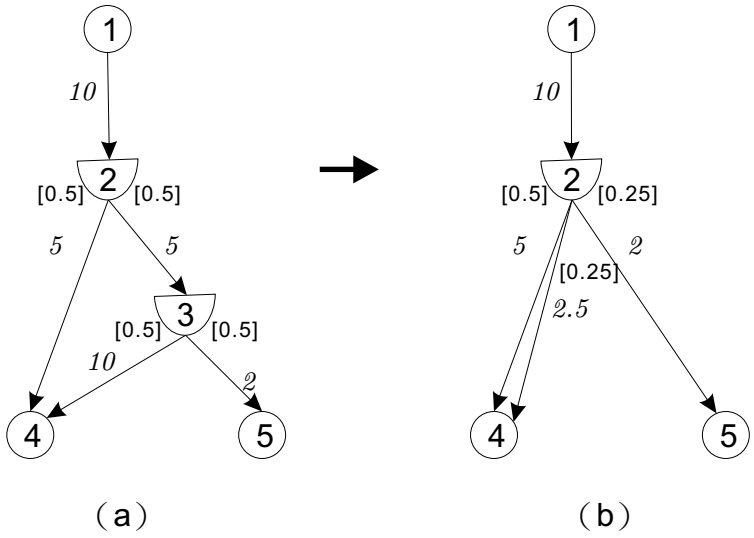


Figure 3.7: Compacting D-group may induce parallel arcs.

1. Compacting D-groups may induce parallel arcs.

   After compacting a D-group, the newly created member arcs may connect with the same O-node, which induces parallel arcs. See Figure 3.7 for example.

2. Compacting a single transshipment O-nodes may induce a D-group.

   When both of the arcs adjacent to a transshipment O-node that connect to two D-nodes with its incoming arc and outgoing arc, compacting such an O-node will induce a D-group. See Figure 3.8 for example.

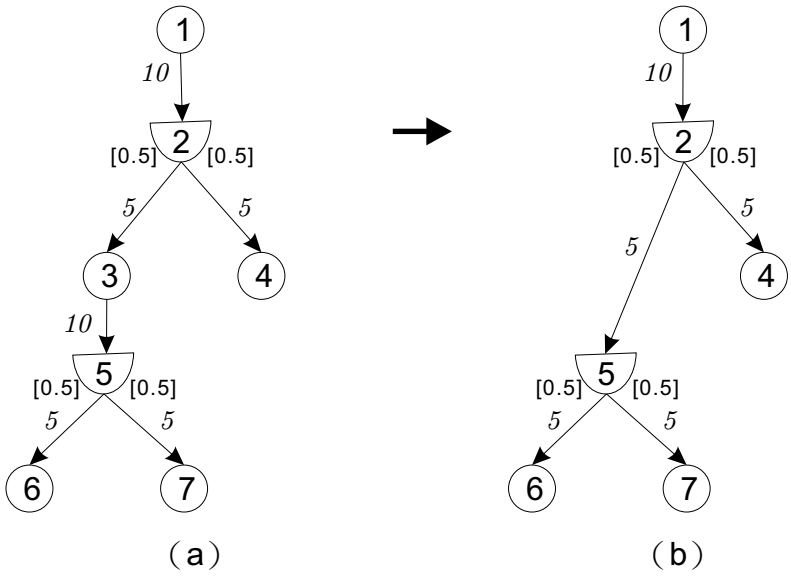


Figure 3.8: Compacting a single transshipment O-node may induce a D-group.

3. Compacting a single transshipment O-node may induce mismatched arcs.

   When one of the arcs adjacent to a single transshipment O-node that connects with a D-node, compacting such a single transshipment O-node may induce mismatched arcs. See Figure 3.9 for example.

4. Compacting a single transshipment O-node may induce self-loop arcs.

   After compacting a single transshipment O-node, an arc may connect to the same end nodes. See Figure 3.10 for example.

5. Compacting a single transshipment O-node may induce parallel arcs.

   After compacting a single transshipment O-node, some arcs may connect to the same end nodes. See Figure 3.11 for example.

6. Compacting parallel arcs may induce a single transshipment O-node.

   After compacting parallel arcs, some nodes adjacent with more than two arcs may become a single transshipment O-node. See Figure 3.12 for example.

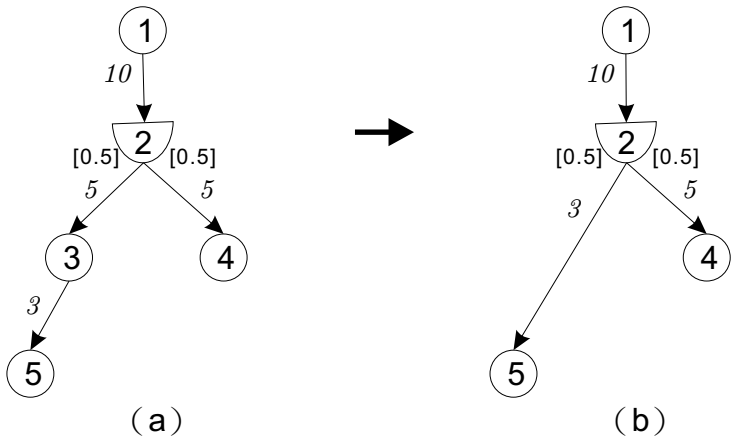


Figure 3.9: Compacting a single transshipment O-node may induce mismatched arcs.

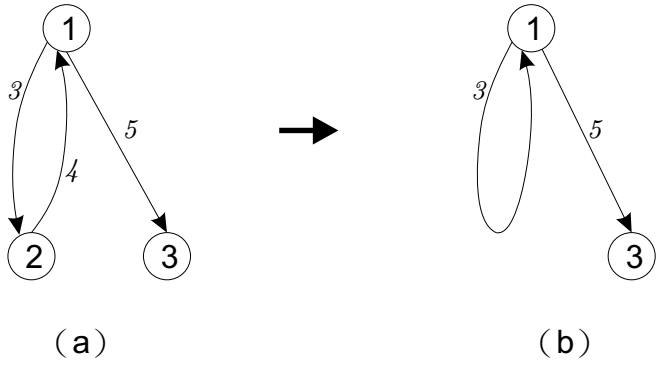


Figure 3.10: Compacting a single transshipment O-node may induce self-loop arcs.

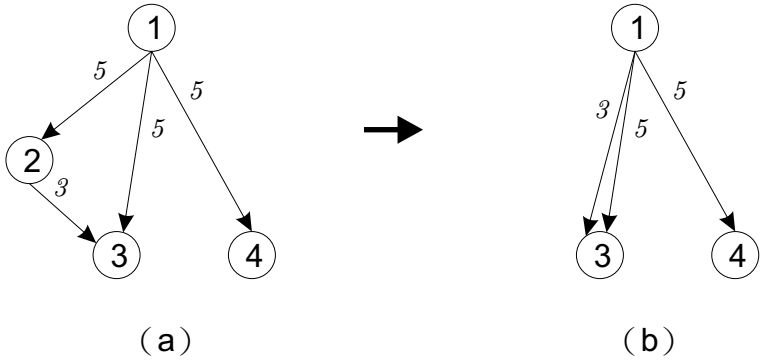


Figure 3.11: Compacting a single transshipment O-node may induce parallel arcs.

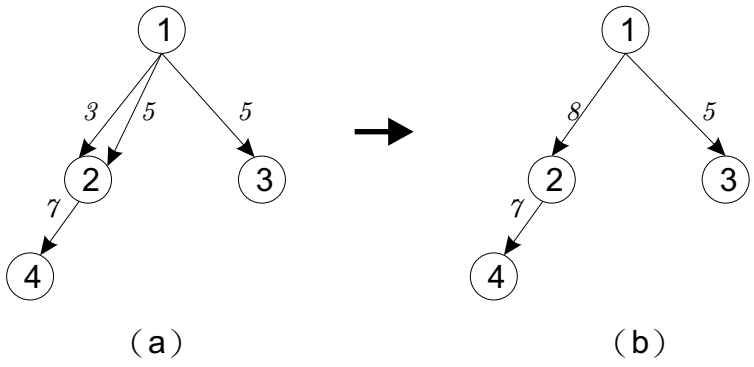


Figure 3.12: Compacting paralell arcs may induce single transshipment O-node.

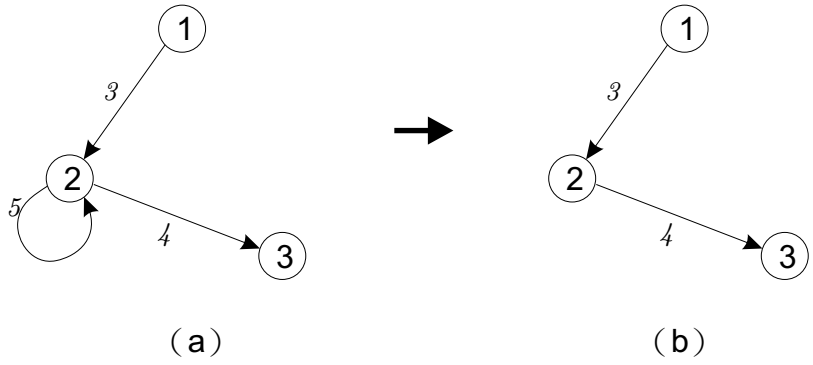


Figure 3.13: Compacting self-loop arcs may induce a single transshipment O-node.

7. Compacting self-loop arcs may induce a single transshipment O-node.

   After compacting self-loop arcs, some nodes adjacent with more than two arcs may become a single transshipment O-node. See Figure 3.13 for example.

Figure 3.14 illustrates the inducing relations between these five compacting rules. Note that there are three elementary directed cycles: (C1)-(C4)-(C2)-(C1), (C2)-(C3)-(C2) and (C2)-(C4)-(C2). Thus we may iteratively conduct (C1), (C4) and (C2), (C2) and (C3) or (C2) and (C4) until they finish their inducing relations and then conduct (C5) in the end of the process.

Each time when any of compacting rules (C1), (C2), (C3) or (C4) is applied, the network will be reduced by at least one arc or one node. Thus the entire compacting

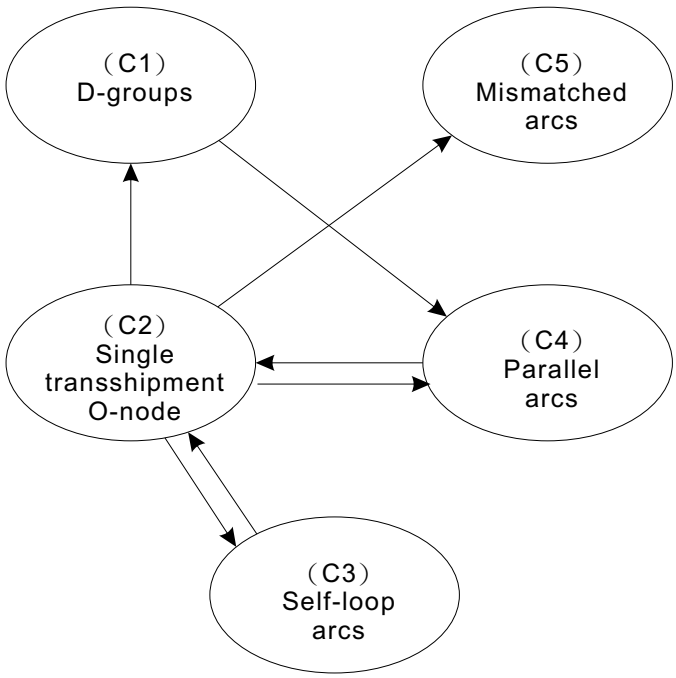


Figure 3.14: Inducing relations of different compacting rules

process takes $O(m^2)$ time and produces a simplified network which has smaller size and is easier to detect the bottleneck for solve a maximum flow problem.

In summary, we observe two properties for a compacted distribution network. First, every intermediate node (i.e. an O-node or a D-node) is adjacent to at least three arcs, including one incoming arc and one outgoing arc. Second, each D-node will not be adjacent to any other D-node. By these observations we may easily check whether a distribution network is compactible. For a distribution network not in its compact form, we may compact it using the following algorithm.

Algorithm 1 Compact(G)

while (G is not in compact form)
case 1: detect a D-group then apply (C1)
case 2: detect a single transhipment O-node then apply (C2)
case 3: detect self-loop arcs then apply (C3)
case 4: detect parallel arcs then apply (C4)
End while
case 5: detect capacity-mismatched arcs then apply (C5)

3.2 Implementing the multi-labeling method

Ting (2005) proposes an approach called multi-labeling method to solve maximum flow problems in a distributed network. Multi-labeling method requires manual detection for components, which is not trivial for implementation. In this Section, we reorganize procedures in multi-labeling method and propose an implementation.

First we introduce additional notation required for our algorithm. Let AG represent an augmenting subgraph, *flag*: indicate whether any AG exists in the current residual network, and *flow* be the total amount of flow that could be sent in an AG. The following is a modified multi-labeling method.

Algorithm 2 Modified_Augmenting_path

```
flow:=0
flag:=true
While (flag=true) do
    While (flow≤0 ) do          // if the AG is invalid, find another one
        Find another AG
        If (AG exists) then     // if there is another AG, find the flow
            Identify components in the AG
            Solve the linear equations
            Determine the flow could be sent
        Else then               // if there is no AG, exit the loop
            Break
            flag := false
    End while
    If (flag=true) then         // if there is a valid AG, send the flow
        Send flow
        Update the residual network
        flow:=0
    Else then Break             // if there is no AG, exit the loop
End while
```

The above algorithm illustrates steps for our implementation which includes five major procedures: (1) to construct an augmenting subgraph, (2) to identify components in an augmenting subgraph, (3) to solve a system of linear equations, (4) to determine the flow to be sent in an augmenting subgraph, and (5) to update the residual network.

We will describe these procedures in Section 3.2.1, 3.2.2, 3.2.3, 3.2.4, and 3.2.5.

3.2.1 Constructing an augmenting subgraph

Ting adopts the concept of the Depth-First Search (DFS) to try out every augmenting subgraph, which goes to the sink or source and satisfies flow balance, bounds, and distillation constraints. As we know, an augmenting path is a directed path from a source to a sink. Nevertheless, when an augmenting path in a distribution network reaches a sink, the algorithm has to track back any D-node in the augmenting subgraph and go along the outgoing arc of the D-nodes to satisfy flow distillation constraints, which may turn an augmenting path into an augmenting subgraph.

In some intermediate stage of constructing an augmenting subgraph, it is possible that the flow will be blocked out on some O-node. In that case, we have to retreat from that node and keep exploring another unvisited passable arc from a previously visited node. In other words, when retreating to an O-node, we have to check if there is any other unvisited arc with the same orientation as the removed one. Take Figure 3.15 for example. The augmenting subgraph in Figure 3.15(a) can not send further flow on node 4. So we retreat to node 3 and remove arc $(3, 4)$ from current augmenting subgraph as shown in Figure 3.15(b). The only passable arc in node 3 is arc $(3, 5)$, so the subgraph continues to grow along arc $(3, 5)$. Eventually, the augmenting subgraph connects a sink and the constructing process terminates as shown in Figure 3.15(c).

This DFS-like constructing algorithm becomes inefficient when retreating to a D-node. In particular, different searching orders on member arcs from a D-node will give different results. Take Figure 3.16 for example. The augmenting subgraph in Figure 3.16(a) can grow along arc $(6, 2)$ or arc $(5, 2)$. If we grow along arc $(6, 2)$, the resultant augmenting subgraph will be formed as in Figure 3.16(b), which is an invalid subgraph since there is no flow streaming into the T-node. However, when the subgraph grows

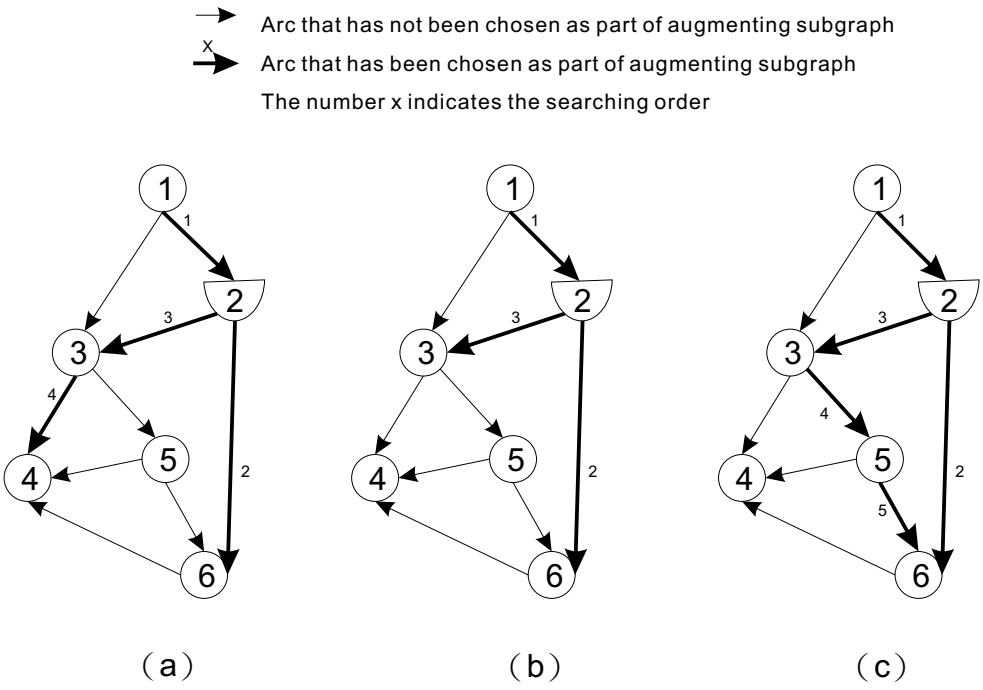


Figure 3.15: An example of retreating to an O-node.

along arc $(5, 2)$, we can obtain a valid subgraph as shown in Figure 3.16(c). Therefore, it is necessary to enumerate all possibilities of search order when retreating to a D-node. Therefore, for a D-node with α outgoing arcs, $(\alpha - 1)!$ searching orders need to be examined.

The constructing process stops when an augmenting subgraph connects a source to a sink. The multi-labeling method first finds a directed path connecting a source to a sink, then it iteratively identifies a directed path from a visited D-node passing some member arc and connects to a sink, or a source, or some visited node, until all member arcs from each visited D-node have been included in an augmenting subgraph. Therefore an augmenting subgraph may also contain a directed cycle either to a source or to some visited nodes. As long as the total net flow is positive, such an augmenting subgraph is valid and we can send positive flow on the residual network.

Take a residual network in Figure 3.17 for example. The numbers next to arcs

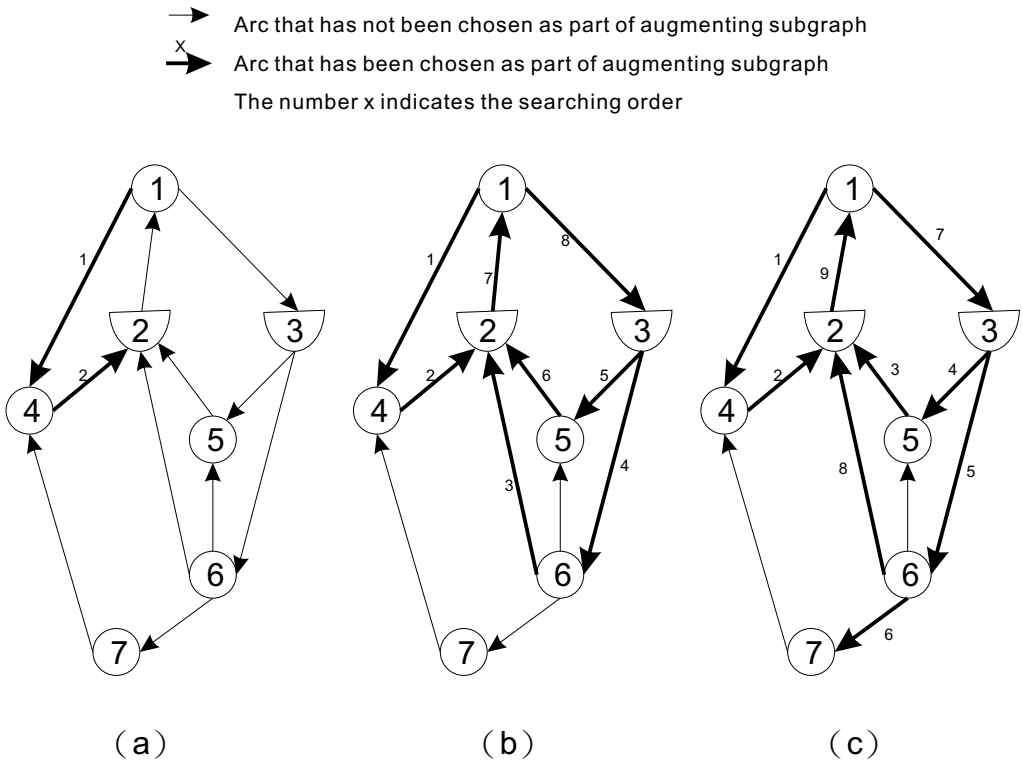


Figure 3.16: Different searching orders lead to different subgraphs.

specify the residual capacities r_{ij}.and the numbers in brackets define the distillation factors for member arcs from D-nodes. The bold arc in Figure 3.18(a) denotes the augmenting subgraph that reaches a T-node but has not tracked back other member arcs from D-nodes. After tracking other member arcs from a D-node, we get an augmenting subgraph using bold arcs in Figure 3.18(b). We repeat the tracking back operations for each D-node in an augmenting subgraph. Finally, a complete augmenting subgraph is determined as shown in Figure 3.18(c).

In implementation, we use a modified DFS which stops probing whenever a T-node, S-node, or a node that has been visited is encountered. It then continues probing from a visited D-node as long as it has member arcs not yet probed. This procedure takes $O(m\Pi_{D_i} \deg(D_i)!)$ where D_i is a D-node in the resultant augmenting subgraph with degree $\deg(D_i)$. Unfortunately, this complexity is obviously exponential and in-

evitable, to the best of our knowledge.

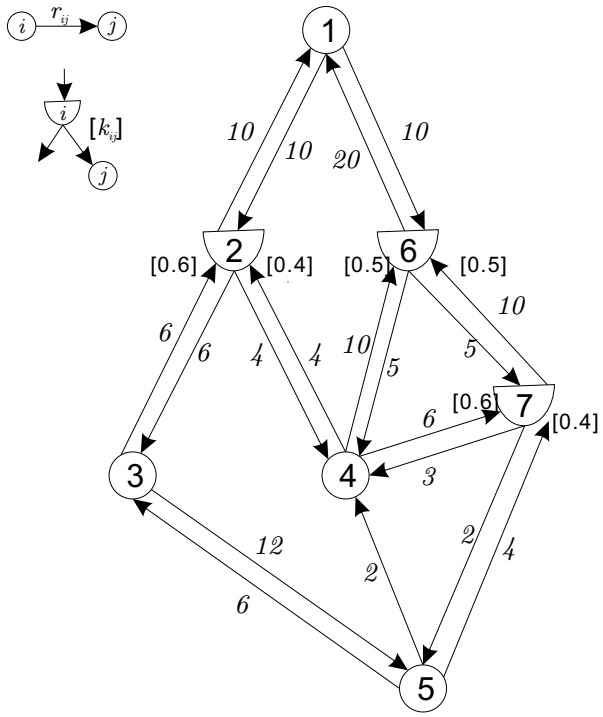


Figure 3.17: An example of residual network

3.2.2 Identifying components in an augmenting subgraph

After finding such an augmenting subgraph, the algorithm then identifies components in the augmenting subgraph. A component is a maximum subgraph in which arc flows can be expressed by linear functions of a single variable. In other words, once the amount of flow on one of the arcs inside a component is determined, all the other arc flows in the same component can be determined right away. Components are separated with each other by O-nodes which have at least three adjacent arcs in an augmenting subgraph. If an O-node has only two adjacent arcs, which must be one incoming arc and one outgoing arc (otherwise an O-node becomes an S-node or a T-node, which violates its definition), the flows on these two arcs must be the same and could be represented by a single variable which means these two arcs belong to

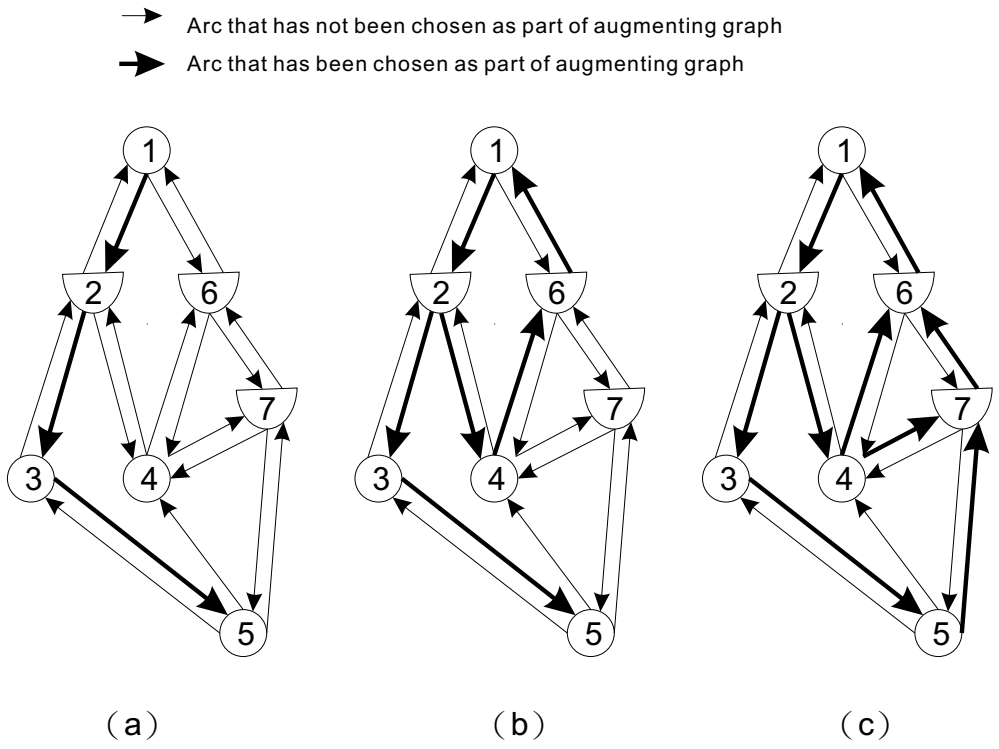


Figure 3.18: An example of constructing an augmenting subgraph

the same component. Therefore, arcs with flows represented by functions of the same variable belongs to the same component.

Continuing on the above example in Figure 3.18, suppose the flow in arc $(1, 2)$ is a, then the flows in arc $(2, 3)$ and arc $(2, 4)$ become $0.6a$ and $0.4a$ respectively. As for node 3, because the flow in arc $(2, 3)$ equals to the flow in arc $(3, 5)$, we will know the flow in arc $(3, 5)$ is $0.6a$. As for node 4, we can not determine the flows on arc $(4, 6)$ and arc $(4, 7)$, so we may first assign two more variables to represent them but later we will find the flows on $(4, 6)$ and $(4, 7)$ are proportional to each other since they are in the same component. The result is shown in Figure 3.19(a). The flow balance constraints for all nodes that joint different components form a system of linear equations. The equation associated with node 4 is $0.4a = 0.5b + 0.3b$.

In implementation, we first identify the O-nodes with more than two arcs in

an augmenting subgraph and define these nodes as *boundary nodes*. Starting from a source, Breadth-First-Search (BFS) can identify all of its reachable nodes. We may modify the BFS to backtrack whenever a boundary node is encountered. Thus we can determine the flow value for each arc in the same component as a function of a single variable. We repeat the same procedures by starting from a different boundary node as long as it has an adjacent arc that has not been visited by the modified BFS. This procedure takes $O(n + m)$ time.

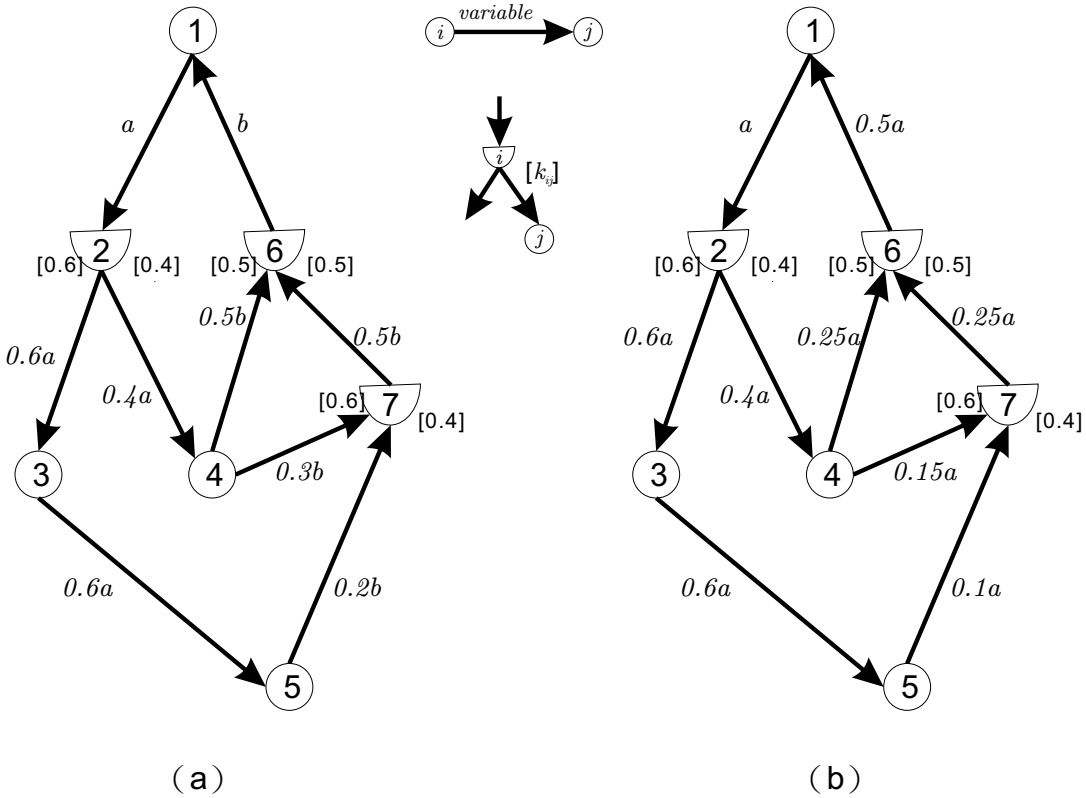


Figure 3.19: Identify components in an augmenting graph and replace all variables with a single variable

3.2.3 Solving a system of linear equations

By the flow balance constraints associated with each O-node jointing different components, we can obtain a system of homogeneous linear equations. Solving this

system of linear equation can help us to unify the variables of different components. In particular, the objective is to replace all variables with a single variable instead of solving for the arc flow value directly. However, it is possible that this system of linear equation may have rank higher than the total number of components, which means the system of linear equations may only have the trivial solution (i.e. zero is the only solution). In such a case, the augmenting subgraph is invalid since there is zero flow to be augmented. Otherwise, we can replace all variables with a single variable.

Using the example in Figure 3.19, there are two components jointed at node 4. The flow balance constraint at node 4 shows $0.4a = 0.5b + 0.3b$ which can be reduced to $b = 0.5a$, and then we can replace all variable b with a as shown in Figure 3.19(b). The total flow we could send in the augmenting subgraph is the total incoming flow of the source minus the outgoing flow of it. In the example of Figure 3.19(b), it will be $a - 0.5a = 0.5a$.

In implementation, we may detect the existence of nontrivial solution by counting the number of components and jointed O-nodes. To solve the system of homogeneous linear equations, we may use directive method such as Gaussian elimination to express each variable as a linear function of one universal variable. The time of this process takes $O(q^3)$ where q represents the total number of components.

3.2.4 Calculating the flow to be sent on an augmenting subgraph

After unifying the arc flows by linear functions of a single variable, we normalize the capacity again for each arc. In particular, suppose an augmenting subgraph $G' = (N', A')$ and we use the flow on arc (i', j'), x, to be the single variable to express flows over all other arcs. Suppose $x_{uv} = k'_{uv}x$ where k'_{uv} is a real constant. Then the normalized capacity of arc (u, v) relative to arc (i', j') becomes u_{uv}/k'_{uv} which means the amount of flow (i', j') should contain to saturate arc (u, v). Exploit-

ing the property of normalized capacity, we can identify the bottleneck arcs easily by calculating $\min_{(i,j)\in A'}\{u_{ij}/k'_{ij}\}$. Refer to Figure 3.17, the flow a can be calculated by $min\{\frac{10}{1}, \frac{6}{0.6}, \frac{12}{0.6}, \frac{4}{0.4}, \frac{20}{0.5}, \frac{10}{0.25}, \frac{10}{0.25}, \frac{6}{0.15}, \frac{4}{0.1}\} = 10$, the maximum flow over the entire augmenting subgraph is $0.5a = 5$ and arc $(1, 2)$ is the first arc to be saturated when we increase the flow over the augmenting subgraph.

In implementation, this procedure can be done in $O(m)$ time.

3.2.5 Updating the residual network

After calculating the maximum flow along an augmenting subgraph, we update the residual network in $O(m)$ time and repeat the above steps until there is no more augmenting subgraph. The maximum flow can be calculated by iteratively constructing an augmenting subgraph, decomposing components, solving for variables associated with each component, calculating flows inside each component, and then updating the residual network.

Since the complexity in constructing an augmenting subgraph is exponential, this multi-labeling method is not efficient.

3.3 Summary

In this Chapter we have introduced a preprocessing procedure that simplifies a distribution network, proposed detailed implementation for Ting's multi-labeling method which can be viewed as a modified augmenting path algorithm.

CHAPTER IV

RANDOM NETWORK GENERATOR

This Chapter explains how to generate random distribution networks for computational test. Here a network generator that could generate various distribution networks is implemented. Popular network generators in the literature or on the internet such as GENRMF (1988), WASHINGTON (1993) and AK (1997) that generate random max-flow problems only produce ordinary networks that contain only O-nodes. Here in this thesis, we implement a random network generator which produces a compacted and acyclic distribution network. Since the distribution problem we consider focus on the hierarchical relationship that goes from the supply side to the demand side (or vice versa), we only consider unidirectional (i.e. acyclic) networks in our computational tests.

4.1 Notations for our random network generator

There are several parameters such as n (the total number of nodes), $u_{\max}$ (the maximum capacity for an arc) and $(outdeg\_D_{\max}, outdeg\_O_{\max})$ (the maximum outgoing-degree for a D-node and an O-node) that a user can specify to generate suitable networks. These parameters can not be arbitrarily set. For example, n must be larger than 3 because three nodes can not form a compacted network. The example in Figure 4.1

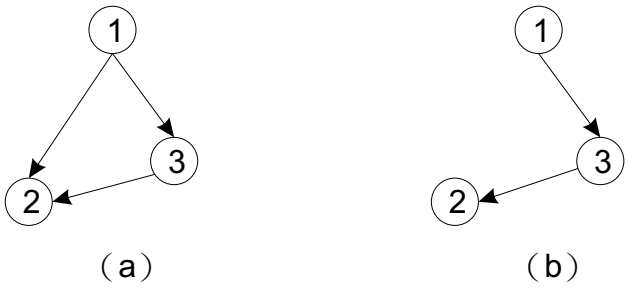


Figure 4.1: A network with only three nodes is not a compact network.

shows that a simple network formed by only three nodes can be compacted. Besides, $outdeg\_D_{\max}$ must be larger than 1 because each D-node has at least two outgoing arcs. The network should be connected, so every node except the T-node has at least one outgoing arc. For this reason, $outdeg\_O_{\max}$ must be larger than 0.

We group the nodes into two set USE and $UNUSE$. The set USE contains the nodes that have been added into the network, while $UNUSE$ refers to the nodes that still wait to be added into the network. Every node i has a distance label $d(i)$ indicating its distance away from the source. We use the maximum number of arcs from the source to a node as its distance label. The notation used for constructing a random network generator is summarized as below.

$at\_least\_one\_D$: flag to record if there is any D-node

$d(i)$: the distance from source to node i.

USE: the set of nodes that has been put into the network

$UNUSE := N \setminus USE$: the set of nodes that have not been put into the network,

$LEAF$: the set of nodes in USE without outgoing arcs

$FARTHER(i)$: the set of O-nodes in USE with distance label equal to or larger than $d(i)$

$AVA(i) := UNUSE \cup FARTHER(i)$: the set of nodes that node i can connect to,

$outdeg(i)$: the outgoing-degree for node i

$outdeg_{\max}(i)$: the maximum outgoing-degree for node i

$outdeg_{\min}(i)$: the minimum outgoing-degree for node i

4.2 Constructing a random distribution network

We describe how to generate a random network as below.

1. Remove a node i from $LEAF$ and construct $AVA(i)$.

   Since our generator generates an acyclic network, an admissible node that one could connect to must be either an O-node of equal or larger distance labels in the current network or a node in $UNUSE$. In particular, a D-node in the current network is not admissible since it has only one incoming arc (and thus it can not receive flow from another node). $AVA(i)$ is used to denote the set of admissible nodes which node i could connect to. Suppose a random network generated in some intermediate stage is shown in Figure 4.2(a). The number besides a node i denotes its distance label $d(i)$. Suppose we set $outdeg\_O_{\max} = 4$ and $UNUSE = \{6\}$. The set of nodes in the current network that node 2 could connect to should be node 4 and node 5, denoted by $FARTHER(2) = \{4, 5\}$, since $d(4) \geq d(2)$ and $d(5) \geq d(2)$. Node 3 does not belong to $FARTHER(2)$ because it is a D-node. Therefore, $AVA(2) = \{4, 5, 6\}$.

2. Decide the head nodes for arcs outgoing from node i.

   Since n and $(outdeg\_D_{\max}, outdeg\_O_{\max})$ are already specified by the user and we have to make sure every node has sufficient admissible nodes to connect with, we set $outdeg_{\max}(i)$ to be $\min\{|AVA(i)|, outdeg\_D_{\max}\}$ if i is a D-node, or $\min\{|AVA(i)|, outdeg\_O_{\max}\}$ if i is an O-node. Because each D-node must have at least two outgoing arcs and each O-node must have at least one outgoing

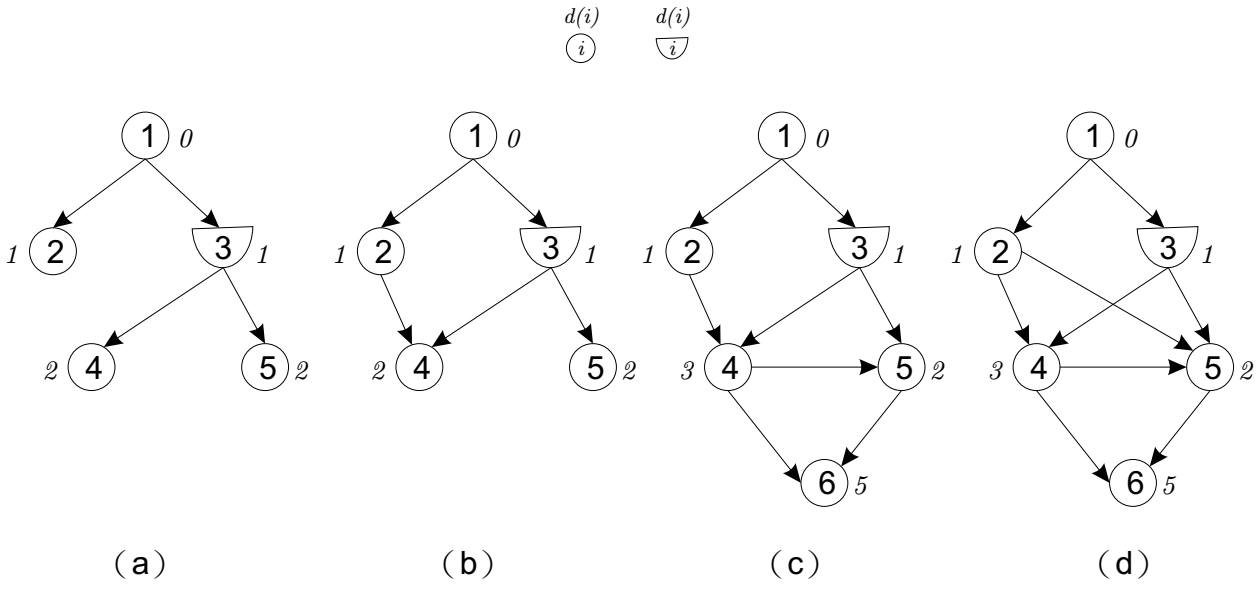


Figure 4.2: An example random network generated in some intermediate stage.

arc, we set $outdeg_{\min}(i)$ to be 2 if i is a D-node, or 1 if i is an O-node. We may then generate a random integer $outdeg(i)$ from the range $[outdeg_{\min}(i), outdeg_{\max}(i)]$ to be its number of outgoing arcs, then we randomly pick head nodes from $AVA(i)$ for these outgoing arcs. See Figure 4.2(a) for example. $outdeg_{\max}(2) = min\{|\{4,5,6\}|, 4\} = 3$. Then we may generate $outdeg(2)$ from $[1,3]$. Suppose $outdeg(2) = 1$, and we select node 4 to be the head node of one outgoing arc from node 2. The result is illustrated in Figure 4.2(b).

3. Update the distance label for a head node j that connects from node i.

   Once a head node j is selected, $d(j)$ will be updated immediately. If node j is removed from $UNUSE$, we set $d(j) = d(i) + 1$. Otherwise, node j must already exist in the current network with some finite distance label $d(j) \geq d(i)$. We then set $d(j) = max\{d(i)+1, d(j)\}$. Then we will check all reachable nodes from node j to see if their distance labels require updating. See Figure 4.2(b) for example. Since node 4 has an original distance label $d(4) = 2$, the updated distance label

of node 4 will be $max\{d(2)+1, d(4)\} = max\{2, 2\} = 2$. That is, $d(4)$ remains 2.

4. Decide the node type for a head node j from $UNUSE$.

   If a head node j is not from $UNUSE$, it must already exist in the current network and has a node type (i.e. either an O-node or a D-node). Otherwise, it is newly added to the current network and has to specify its node type. When $|AVA(j)|$ is 1, which means there is only one eligible head node for node j to connect to, so node j must be an O-node. When node j is the T-node, it must be an O-node. In other situations, the node type for node j will be randomly decided. In any case, a leaf should be an O-node and a D-node must connect with an O-node since the random network should be in a compact form. Therefore, if a new node which is just removed from $UNUSE$ is decided to be a D-node by the generator, we change it to be a non-leaf by immediately deciding all of its outgoing arcs and force their head nodes to be O-nodes. Otherwise, the new node just removed from $UNUSE$ is decided to be an O-node and we will add it into $LEAF$.

5. Modify the topology of the random network whenever necessary.

   Repeat step 1 to 4 until $LEAF$ becomes empty. Since the node type is decided randomly, the generator may generate a network containing only O-nodes. In that case, the network will be erased and the generator will be re-run until a random network that contains a D-node is produced. Because we want to generate a compacted random distribution network, every incompact situation requires correction. In particular, when we detect a single transshipment O-node, the generator will add artificial arcs from a node of smaller or equal distance label to a single transshipment O-node. See Figure 4.2(c) for example. Suppose it contains a random network generated in the stage of empty $LEAF$. Since node

2 is a single transshipment O-node, we will randomly add an arc, say, from node 2 to node 5 to ensure the resultant network is in a compact form.

The pseudo code of our random network generator is presented as follows.

Algorithm 3 Network_Generator

```
at_least_one_D:=FALSE
While (at_least_one_D=FALSE) do
    Initialization
    While (LEAF ≠NULL) do
        Pop node i from LEAF
        If (i ≠ t) then
            Construct AVA(i)
            Decide the outgoing-degree for node i
            Choose head nodes randomly for node i
            For each head node j
                Update information
                If (j is from UNUSE) then
                    Construct AVA(j)
                    Decide node type
                    If (j is O-node) then
                        Put node j into LEAF
                    Else then
                        at_least_one_D:=TRUE
                        Decide the outgoing-degree for node j
                        Choose head nodes ks randomly for node j
                        Update information, set nodes ks as O
                        Put nodes ks into LEAF
    End while
    Correct single transshipment O-nodes
End while
```

More details about the data structures of our generator and a step-by-step illustrative example are provided in Appendix A.

4.3 Dicussion on random network generator

Our random network generator will randomly generate a distribution network of n nodes which contains one source node (S-node) and one sink node (T-node). Our randomly generated network will always be acyclic since a directed cycle means the

objects (might be some material or product) will be processed over and over again along a directed cycle which seems not so applicable for real applications. Another reason for generating acyclic random networks is to guarantee the feasibility of a distributed max-flow problem. In order to guarantee the feasibility, each node in a randomly generated network must connect to the T-node by at least one path. Since our generator always connects a node of smaller distance label to a node of larger distance label, the generated network thus has to be acyclic since there exists no directed cycle.

We also guarantee our randomly generated networks must be in compact forms. In particular, the generator always connects a D-node to an O-node, randomly adds arcs to any single transshipment O-node, avoids parallel arcs and self-loops, and assigns random capacities on arcs connecting to each D-node according to their distillation factors to avoid mismatched arcs.

First we list several important properties for any random distribution network generated by our random network generator.

Property 1. *For each arc (i, j) in the generated network, $d(i) < d(j)$*

Proof. There are four possible situations when an new arc (i, j) is created: (a) when node $i \neq t$ with a finite $d(i)$ is removed from $LEAF$ and node j is removed from (a1) $UNUSE$ and $j \neq t$, then $d(j) := d(i) + 1$; (a2) $UNUSE$ and $j = t$, then $d(j) := n-1$; (a3) $AVA(i) \setminus UNUSE$, then $d(j) = max\{d(i)+1, d(j)\}$; (b) when (i, j) with $d(i) < d(j)$ is added to prevent i (or j) from being a single transshipment O-node. Therefore $d(i) < d(j)$ remains to hold. □

Property 2. *Each generated network is acyclic.*

Proof. If there exists a directed cycle $i_1 - i_2 - \ldots - i_k - i_1$ of k arcs, by Property 1 we obtain a contradiction $d(i_1) < d(i_2) < \cdots < d(i_k) < d(i_1)$. □

Property 3. *The S-node has no incoming arc and the T-node has no outgoing arc. For each node other than the S-node and the T-node, it must have both incoming and outgoing arcs.*

Proof. The generator guarantees that there is only one S-node and one T-node. Furthermore, the S-node has no incoming arc and the T-node has no outgoing arc by the setting of the generator.

For each node i other than the S-node and the T-node, it must be either an O-node or a D-node and initially stored in $UNUSE$. If i is an O-node when the generator creates an arc (k, i), it will be moved to $LEAF$, which means i must have at least one incoming arc. Since the generator removes one node from $LEAF$ in each iteration and there are finitely many nodes, $LEAF$ will eventually become empty. Thus each node i in $LEAF$ will eventually be removed when an arc (i, j) is created, which means i must have at least one outgoing arc. If i is a D-node when the generator creates an arc (k, i), it will directly be removed from $UNUSE$ and then the generator will identify its outgoing arcs (i, j)s, which means i must have at least one incoming arc and more than one outgoing arcs. Thus for each node other than the S-node and the T-node, it must have both incoming and outgoing arcs. □

Property 4. *Each generated network is connected with n nodes.*

Proof. There exists no isolated node in each generated network by Property 3. Suppose there exist two (or more) components $COMP_1$ and $COMP_2$ in a generated network. By Property 2 we know both $COMP_1$ and $COMP_2$ are acyclic, and thus there must exist at least one node in $COMP_1$ and one node in $COMP_2$ that has no outgoing arcs (or incoming arcs), which contradicts Property 3. Thus each generated network is connected with n nodes. □

Property 5. *There always exists a path from the S-node to each node and a path from each node to the T-node.*

Proof. Starting from a node i in a generated network, we may conduct a DFS algorithm along an outgoing arc (i, j) which labels j and then keeps labeling a node k along some arc (j, k) if it exists. The DFS will keep labeling an unvisited node until the T-node is labeled since there exists no directed cycle by Property 2 and each intermediate node other than the T-node has at least one outgoing arc by Property 3. Thus each node can reach the T-node in a generated network.

Similarly, we may apply a reverse DFS algorithm starting from a node i which searches along an incoming arc (k, i) and labels k in a reverse fashion. Using similar arguments as described in previous paragraph, the reverse DFS algorithm will eventually reach the S-node, which means the S-node can reach any other node in a generated network. □

Property 6. *Each O-node except the S-node and the T-node in a generated network must be connected by at least three arcs which include at least one incoming arc and one outgoing arc.*

Proof. For each O-node other than the S-node and the T-node, by Property 3 we know it has at least one incoming arc and one outgoing arc. The generator will also check the existence of a single transshipment O-node (i.e. an O-node connected with exactly one incoming arc and one outgoing arc) and add new arcs for such an O-node so that in the end each O-node will have at least three adjacent arcs. □

Property 7. *There exist no self-loops, parallel arcs, arcs of mismatched capacities and D-groups in a generated network.*

Proof. Property 1 guarantees no self-loop.

When the generator decide an outgoing arc (i, j) for either an O-node i removed from

$LEAF$ or a D-node i removed from $UNUSE$, it will always choose different head node j. When adding a new arc (k, i) or (i, j) to a single transshipment O-node i where node k or node j are O-nodes, the generator also prevent creating parallel arcs. Thus there exists no parallel arc in a generated network.

Whenever a D-node i is created from a mother node k which is an O-node, the generator will immediately connect i to at least two member nodes which are also O-nodes chosen from $AVA(i)$. So the generator will not form a D-group at any time.

When the generator assigns the capacity for arcs connecting to a D-node, it will be proportional to the assigned distillation factor and thus no arcs of mismatched capacity will be created. □

Property 8. *Each generated network is in a compact form.*

Proof. It follows directly from Property 6 and Property 7. □

We also list several properties that give more insights about our generator. We skip some trivial proofs.

Property 9. *Each node in $LEAF$ must be an O-node associated with at least one incoming but no outgoing arc.*

Property 10. *Throughout the generating process, each node with a finite distance label must already exist in the current network, whether it is in $LEAF$ or not. Furthermore, such a node is in $LEAF$, if and only if it has no outgoing arc at that time.*

Property 11. *In any intermediate stage, the node with the largest finite distance label must be in $LEAF$.*

Proof. Suppose node i is the node in the current network with the largest finite distance label and it is not in $LEAF$, then i must have an outgoing arc (i, j) connecting to a

node j with finite distance label $d(j) > d(i)$ by Property 1. This contradicts the assumption that $d(i) \geq d(j)$. □

Property 12. *$UNUSE$ will become empty before $LEAF$ becomes empty.*

Proof. Suppose $LEAF$ contains only one node i which is not the T-node (otherwise it is trivial). Node i must be an O-node by Property 9. When we remove i from $LEAF$, we will immediately identify an outgoing arc (i, j) leaving from i, and j must be removed from $UNUSE$ instead of $FARTHER(i)$ (which means j is in $LEAF$) since otherwise $d(j) \geq d(i)$ by Property 11 which contradicts the assumption that i is the only node in $LEAF$. □

Property 13. *Each O-node stays in $LEAF$ once. A D-node j connected from an O-node i must be removed directly from $UNUSE$, instead of $FARTHER(i)$.*

Property 14. *Both $LEAF$ and $UNUSE$ will eventually become empty.*

Proof. The generator removes one node from $LEAF$ in every iteration. No node will be added to $LEAF$ and $UNUSE$ throughout the generating process. Since there are finitely many nodes, each node stays at $LEAF$ or $UNUSE$ at most once and the generator removes at least one node from either $UNUSE$ or $LEAF$ at each iteration, by Property 12 finally $LEAF$ and $UNUSE$ will become empty. □

Property 15. *The T-node is the last node removed from $UNUSE$, but may not be the last node removed from $LEAF$.*

Property 16. *Whenever a node i is to be removed from $LEAF$, it must be able to connect to some node j with $d(j) \geq d(i)$.*

Property 17. *For each generated D-node, there must exist sufficient number of O-nodes to be its member nodes.*

Proof. For an unknown-type node i, only when $|AVA(i)| \geq 2$, it will become eligible to be a D-node. Once node i is decided to be a D-node, its upper bound of outgoing degrees, $outdeg_{\max}(i)$, will be $\min\{|AVA(i)|, outdeg\_D_{\max}\}$, which guarantees at least two outgoing arcs for node i. □

4.4 Summary

In this Chapter we introduce how our random network generator works. After a user specifies the total number of nodes, the maximum capacity for an arc and the maximum outgoing-degree for a D-node and an O-node, our generator will give a random distribution network which contains only one S-node and one T-node and is acyclic, connected and in a compact form. Mathematical properties together with their proofs are also discussed to provide more insights in the design of our generator.

CHAPTER V

PREFLOW-PUSH ALGORITHM AND MAX-FLOW MIN-CUT THEOREM

In this Chapter, we investigate the applicability of preflow-push algorithm and discuss the max-flow min-cut theorem for solving the distributed maximum flow problem. In particular, we have encountered several difficulties in applying the preflow-push algorithm and propose some modification strategies. Although our modified preflow-push algorithm only solves some classes of max flow problems, we list our proposed strategies and illustrate the difficulties that one may encounter when using preflow-push algorithm.

Due to the flow distillation constraints, the traditional max-flow min-cut theorem can no longer hold. In the second part of this Chapter, we record optimal primal and dual solutions for some distributed max-flow problems and illustrate our observations which may suggest modifications to the conventional max-flow min-cut theorem and provide more theoretical insights.

5.1 Preflow-push algorithm

Preflow-push algorithm (Goldberg and Tarjan, 1988) pushes flows via individual arcs at each step instead of via augmenting paths. It allows flow to be accumulated in transshipment nodes and thus the flow balance constraints may be violated at

intermediate stages. The algorithm first saturates all outgoing arcs of the source s, accumulates flows in the head nodes of those outgoing arcs of s, and then tries to push flows from these nodes to the sink via arcs satisfying some distance and capacity criterion.

In particular, it maintains a *preflow* which makes the flow entering a node not less than (i.e. may be equal to or larger than) the flow leaving a node. For a given preflow x, we define the *excess* for each node i as $e(i)$. When a node other than the source and sink has a positive excess, we say it is an *active* node. Initially, the algorithm saturates all the arcs leaving the source and creates some active nodes. For each node i, the algorithm maintains a distance label $d(i)$ to record a lower bound of shortest directed path from node i to the sink. In a residual network, suppose (i, j) has positive residual capacity r_{ij}, the algorithm must maintain the property $d(i) \leq d(j) + 1$ at any time. We call this property as a *downhill property*. This property plays an important role in guaranteeing the correctness of the algorithm. We say an arc (i, j) in the residual network is *admissible* if it satisfies the condition $d(i) = d(j) + 1$. Initially we set $d(s) = n$ and $d(i)$ equals to the minimum number of arcs required to connect node i to the sink t. We may calculate the initial $d(i)$ by conducting a reverse Bredth-First-Search algorithm starting from the sink t.

The algorithm iteratively chooses an active node p and send its excess only via admissible arcs to its neighbor nodes closer to the sink, which is called a *push* operation. When there is no admissible arc for an active node p, the algorithm increases $d(p)$ to be the $min\{$distance label of outgoing-neighbors $+1\}$ to produce at least one admissible arc, which is called a *relabel* operation. Note that this relabel operation still satisfies the downhill property $d(i) \leq d(j) + 1$ for each arc (i, j) with $r_{ij} > 0$ in the residual network.

The preflow-push algorithm maintains a preflow and pushes excess flows from an active node to the sink via admissible arcs. If an active node i has $d(i) \geq n$, the excess flow on node i must be shipped back to the source s since there exists no augmenting path in the residual network from i to t.

The preflow-push algorithm is efficient and has strongly polynomial time complexity, especially when we only requires the optimal amount of max-flow since it can obtain a maximum preflow and identify a min-cut earlier. Eventually, the algorithm will terminate when there is no active node in the residual network. See Chapter 7 in (Ahuja, Magnanti, and Orlin, 1993) for more details about the preflow-push algorithms

In a distribution network, the conventional preflow-push algorithm required to be modified to deal with the additional distillation constraints.

5.1.1 Modifications for preflow-push algorithm

In a distribution network, the distillation constraints associated with D-nodes make the preflow-push process different from the conventional one and intrigue several interesting and challenging problems.

First, flow passing through a D-node from its mother node has to be pushed simultaneously to all its member nodes along all admissible member arcs, unlike the conventional preflow-push algorithm in which flow is pushed only along one arc at a time. What if not all the member arcs are admissible? Do we still push the flow anyway via some inadmissible arcs? or Should we relabel the other member nodes as well? If so, in what way should we relabel those member nodes? Second, suppose in a D-family an active member node tries to push flow back to its mother node via two admissible arcs passing through the D-node, the flow may not be directly pushable unless all other member nodes also become active and have admissible arcs passing through the D-node. Therefore, when we relabel a member node to create an admissible arc back

to its D-node in the residual network, we may not only consider the distance label but also need to check whether other member nodes can help the relabeled member node to push flow simultaneously or not. In other words, a relabel operation may have to be accomplished by series of pushing or relabeling operations on other nodes. How should we update the distance label to guarantee correctness of the preflow-push algorithm? Also, for an active node connecting with more than one admissible arcs, which admissible arc to be selected for pushing flow may lead to different results.

To resolve the difficulties that we encountered in solving the distributed max-flow problem by preflow-push algorithm, we propose four possible modifications and extensions to the original preflow-push algorithms. These modifications will resolve some difficulties but may induce other difficulties as well.

(M1) Arcs connecting to a D-node can be treated as a multi-exit arc.

Since the role of a D-node is to distribute flows according to the prespecified distillation factors and we wish to satisfy the flow distillation constraints, we suggest not to let flow accumulated in any D-node and treat a D-node as an invisible gate with multiple exits (each corresponds to a member arc). Functionally speaking, we can treat arcs connecting to a D-node as a multi-exit arc as shown in Figure 5.1, and give no distance label for a D-node since it accumulates zero flow at any time.

(M2) An active node may have negative excess.

The excess for each active node must be positive in original preflow-push algorithm because preflow is always an upper bound for feasible flow in any intermediate stage. Positive excess will always be pushable from an active node via some admissible arc (if there exists no admissible arc, a relabel operation will guaran-

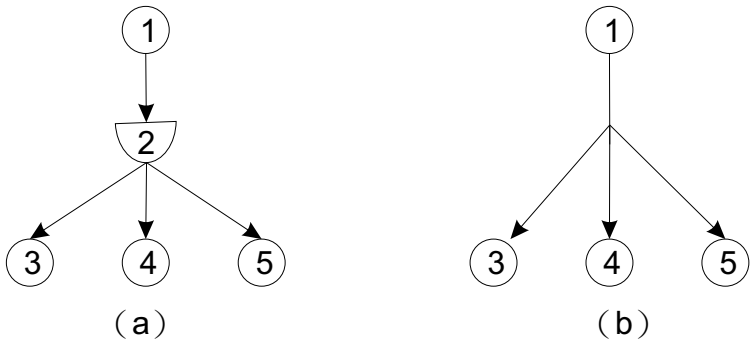


Figure 5.1: Arcs connecting to a D-node is viewed as a multi-exit arc.

tee to create at least one admissible arc.). In a distributed max-flow problem, sometimes we may have to push flow from an active member node k back to its mother node i via a D-node j. We call such an operation as a flow streaming into a D-node *in reverse direction*. If all the member nodes are active and connect with the D-node by admissible arcs, the push operation can be conducted simultaneously without any problem at all.

On the other hand, if not all the member nodes are active, we could not push flows in reverse direction since the flow distillation constraint should be obeyed. Such a difficulty may be resolved by pushing the excess from an active member node k to one of its adjacent O-nodes, l (if l exists and (k, l) is admissible.). However, if there exists no such an adjacent O-node l, or if (k, l) is not admissible, we have to make a decision: either to relabel k such that (k, l) becomes admissible, or to push flow back to its mother node along (k, j) and (j, i). This decision will also affect the downhill property. For example, if (k, l) is not admissible but (k, i) and (i, j) are (by (M1) we do not store $d(j)$), relabeling $d(k) = d(l) + 1$ will make $d(k) > d(i) + 1$ for (k, i) and violate the downhill property.

In order to maintain the distance property, we generalized the range of excess so that one node may have negative excess. We call a node with negative excess

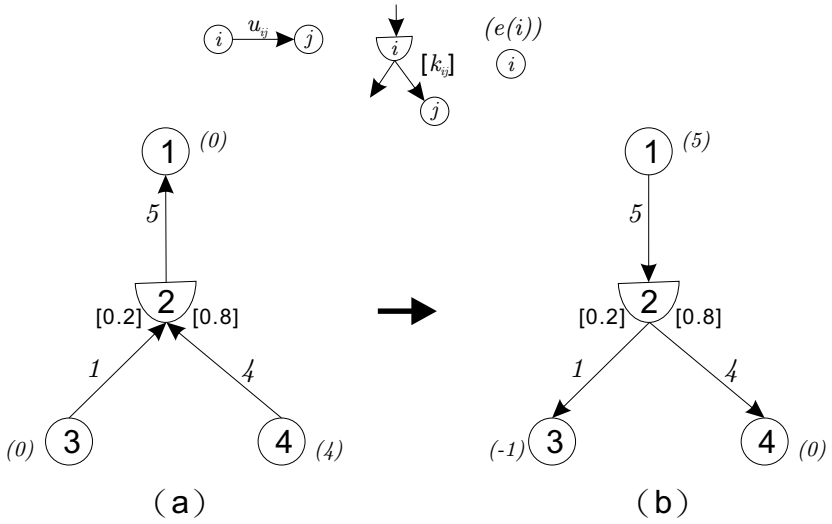


Figure 5.2: Flow could be borrowed in advance.

as a *negative active* node. Unlike a conventional active node that has positive excess whose relabel operations raises its distance label, a negative active node wishes to receive flow from its neighbors via admissible arcs. In particular, a negative active node i may receive flow from its neighbor node j, if (j, i) satisfies the condition $d(j) = d(i) + 1$. The negative excess flow streaming along an admissible arc is called *sending negative flow*. When there is no admissible arc for a negative active node i, we relabel the $d(i)$ to be the $max\{$distance of incoming-neighbors$-1\}$.

For the case where one wishes to push positive excess from a positive active member node back to its mother node while some other member nodes do not have positive excess, using the concept of negative excess, one may still pushes flow from the active member node and make other member nodes become negative active. We may view such a process as if the nodes with negative excess borrow flow from somewhere in advance and wish to receive flow from other nodes afterwards for flow balance. See Figure 5.2.

(M3) An active mother node may send flow to all of its member nodes simultaneously even though some member arcs are not admissible.

In the original preflow-push algorithm, the downhill property will always be maintained. Here in a distributed max-flow problem, it is possible that the downhill property has to be destroyed. For example, to maintain the distillation constraint, an active mother node pushes flow into one member node through a D-node, the excess flow must be sent to all other member nodes at the same time even thought these arcs might not be admissible. Such a result might lead to a *distance gap* between node i and j connected by (i, j) with positive residual capacity r_{ij} but $d(i) > d(j)+1$. Take Figure 5.3 for example. Once the active node 1 attempts to send flow to node 3 through D-node 2, flow will be sent to member node 4 as well. It leads $d(1)$ to be $d(4) + 1 = 7$. The distance gap occurs in arc $(1, 5)$, which is $d(1) > d(5) + 1$.

We realize this modification might destroy the downhill property and thus the correctness of the modified preflow-push algorithm may no longer hold. Unfortunately, at this moment we could not provide good remedy. We suggest future researchers to investigate ways to push flow and relabel nodes without violating the downhill property.

(M4) An active member node may balance flow to other reachable member nodes before pushing flow back to its mother node.

When we wish to push flow from some active member nodes in a D-group to the mother node, by (M2) we may create new negative active member nodes. Then, if we select any negative active member node that is just created to receive flow from the mother node, we may just receive the same amount that was just pushed in previous iteration. Thus, the algorithm may keep pushing flows between member

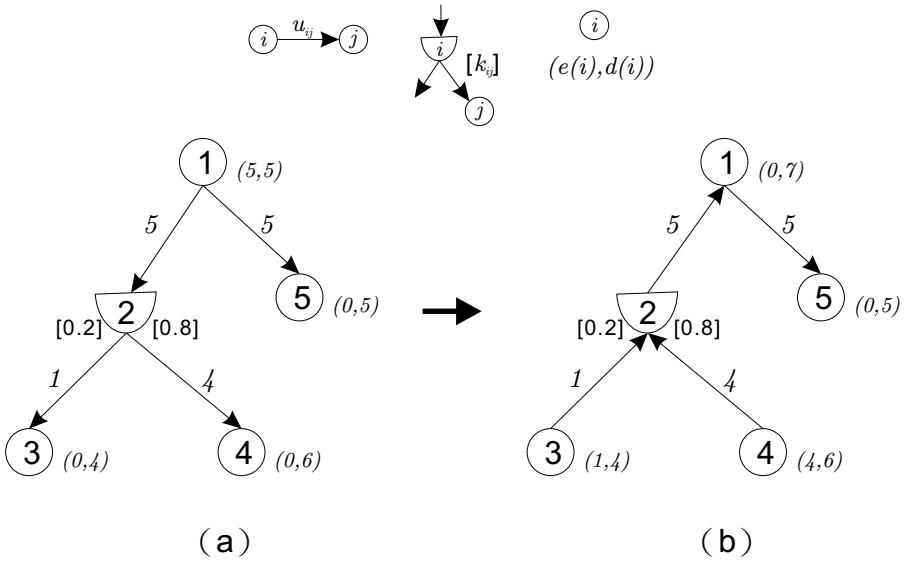


Figure 5.3: An active mother node may push flow to all member nodes via inadmissible arcs.

nodes and mother node every other iteration and changes distance labels until later some active node may push flow to other nodes. Although the distance labels for these nodes will also change, the excess and flow configurations may be the same every other iteration.

To shorten the operations, we suggest to probe other member nodes before an active member nodes pushes flow to its mother node. In particular, an active member node may first check whether it can push flows to other member nodes of the same D-group via arcs in the residual network. If so, we distribute the excess of these member nodes proportionally according to their distillation factors, and then the push operations can be conducted simultaneously from all these member nodes to the mother node. The probing operation can be done by a Depth-First-Search (DFS) algorithm. Note that by this modification we may also violate the downhill property since the node excess may be distributed via inadmissible arcs to other member nodes.

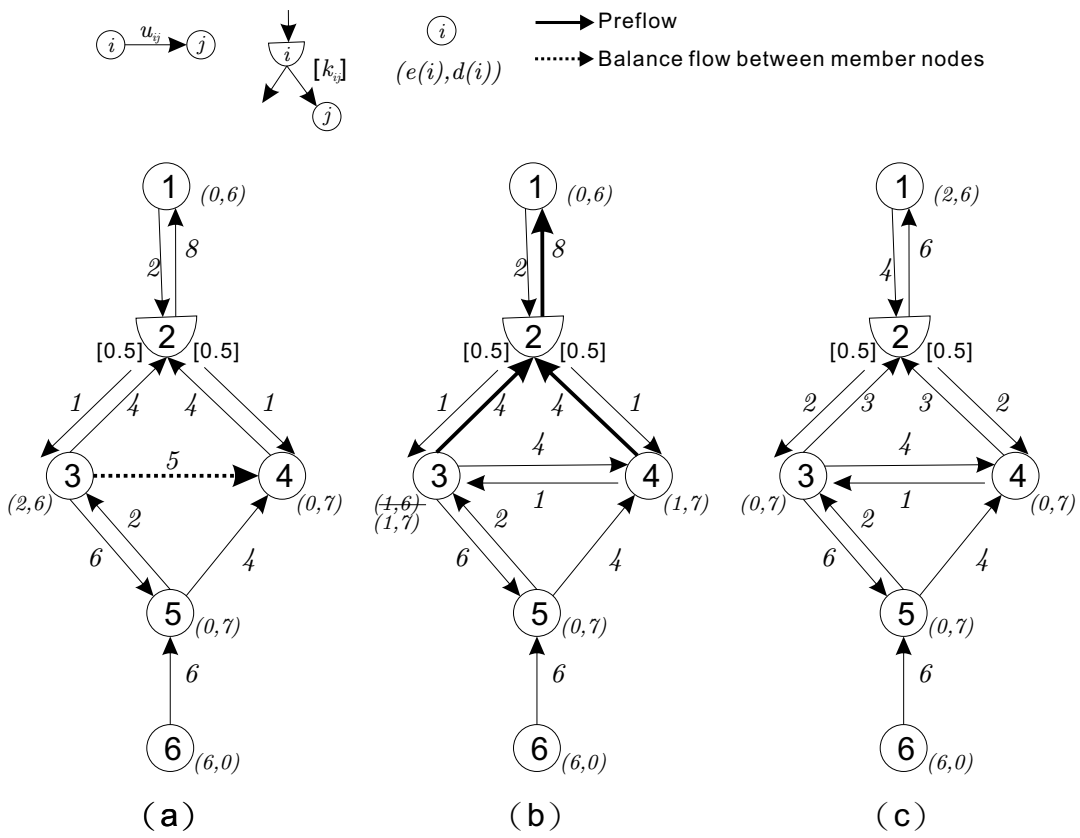


Figure 5.4: An active member node may balance flow to other reachable member nodes in advance.

See Figure 5.4 for example. Before the active node 3 sends flow to node 1, we allow it to send β flow to another member node 4. Then β can be calculated by solving the equation $(2-\beta):\beta = 0.5:0.5$ according to the distillation factors. Figure 5.4(a) shows that node 3 first sends 1 unit of flow to node 4. Then node 3 and node 4 send flow to node 1 as shown in Figure 5.4(b) and Figure 5.4(c).

All of these four modifications are proposed to deal with the additional distillation constraints. Some of the modifications may violate the distance property. Instead of providing a solution method that guarantees correctness, here we propose several interesting ideas and hope to intrigue more researchers to devote themselves to this topic.

5.1.2 Procedures of modified preflow-push algorithm

With our modifications to the preflow-push algorithm, in this Section we solve some distributed max-flow problems, record the difficulties we encounter, and propose several solution strategies.

We first compute the initial distance labels by performing a reverse breadth-first search (BFS) on the network starting from the sink. We set $d(s) = |O|$, $d(t) = 0$, and ignore the D-nodes. Note that there are more than one way to set the initial distance labels, for example, $d(i) = 0$ for each node i except $d(s) = |O|$.

Then we saturate all arcs adjacent to the source, which is to send as much flow as possible from s. Since we consider arcs connecting to D-nodes as multi-exit arcs by (M1) in Section 5.1.1, flows entering a D-node will not be accumulated but be pushed to all of its outgoing arcs immediately. After the initialization phase, we iteratively pick one active node, positive or negative, and conduct a push operation if there is an admissible arc. If a selected active node has no adjacent admissible arcs, we conduct a relabel operation which should create at least one more admissible arc. The push and relabel operations are conducted iteratively until all nodes are not active, then we terminate the algorithm.

To be more specific, we have two different relabel operations depending on whether the selected active node contains positive or negative excess. For a positive active node, we will pick the neighbor node that can receive flows from the active node with minimum distance label. Likewise, for a negative active node, the neighbor node that can send flow to this active node with maximum distance label. After this relabel operation, there will be at least one admissible arc for the active node. In some sense we generalize the conventional relabeling operation which only raises the distance label for a positive active node.

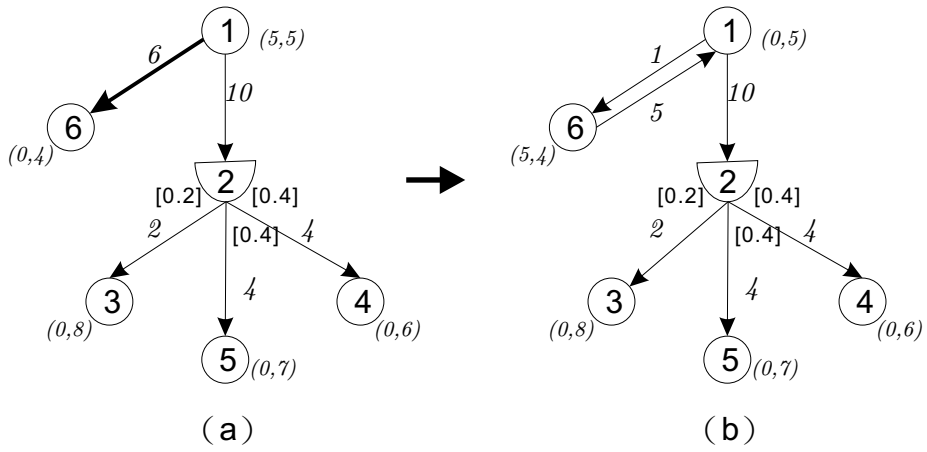


Figure 5.5: Pushing positive flow from a positive active node without passing through a D-node.

Push operation is performed along an admissible arc from an active node. After picking an admissible arc, different push operations are performed according to the types of admissible arc and the active node. Suppose we have an active node i with excess flow $e(i)$ and an admissible arc (i, j), various push operations can be summarized as below.

1. **Pushing flows without passing through a D-node:**

   For a positive active node, we send $min\{u_{ij}, e(i)\}$ flow to its neighbor node. For a negative active node, we receive $min\{u_{ij}, -e(i)\}$ flow from its neighbor node. See Figure 5.5 and Figure 5.6 for examples.

2. **Pushing flows via a D-node from an active mother node:**

   For a positive active mother node i connecting to a D-node k, if there exists a member node j with $d(i) = d(j) + 1$ and positive r_{ik} and r_{kj}, we can send $\min\{r_{ik}, e(i)\}$ flow from i to all member nodes of k according to the distillation constraints. If $d(i) < d(j) + 1$, we may relabel $d(i) = \max_{\text{member node } j} \{d(j) + 1\}$.

   On the other hand, for a negative active mother node i connecting to a D-

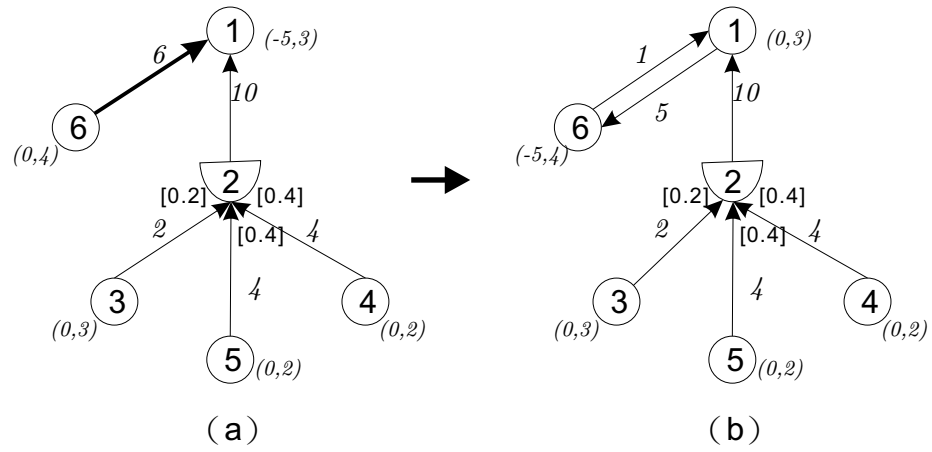


Figure 5.6: Pushing flow to a negative active node without passing through a D-node.

node k, if there exists a member node j with $d(i) = d(j) - 1$ and positive r_{ki} and r_{jk}, we can receive $\min\{r_{ki}, -e(i)\}$ flow from all member nodes of k to i according to the distillation constraints. If $d(i) > d(j) - 1$, we may relabel $d(i) = \min_{\text{member node } j} \{d(j) - 1\}$. See Figure 5.7 and Figure 5.8 for examples.

Note that the operation surely works when all member arcs are admissible. However, in the case when not all member arcs are admissible, whether we should perform the push operations may lead to different results. In particular, some strategy that guarantees correctness of the algorithm should be investigated in the future research for the case where some member arcs are not admissible.

3. **Pushing flows via a D-node from an active member node:**

   For a positive active member node i connecting to a D-node k, if the mother node j has $d(i) = d(j)+1$ and positive r_{ik} and r_{kj}, we can send $\min\{r_{ik}, e(i)\}$ flow from i to the mother node j. In the mean time, all other member nodes also have to send flows to j by the distillation constraints. Such an operation may force some member nodes to become negative active since we relax the nonnegative bound for excess by (M2) in Section 5.1.1. If $d(i) < d(j) + 1$, we may relabel $d(i)$ to be

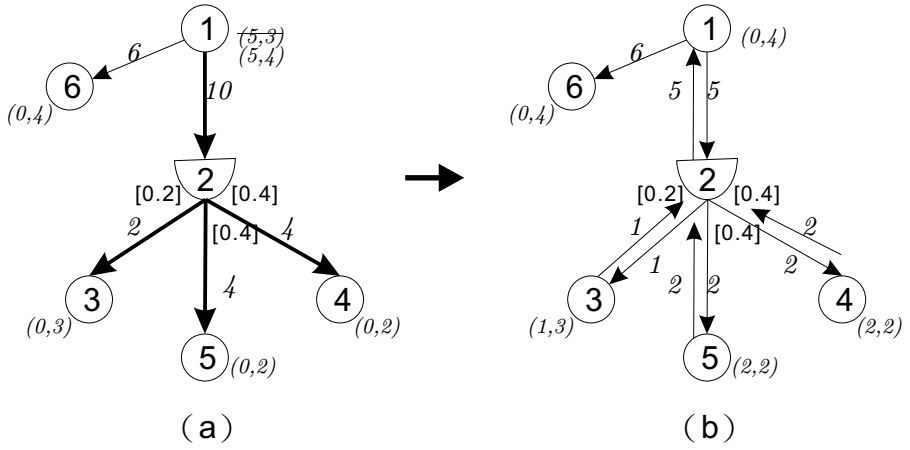


Figure 5.7: Pushing flows via a D-node from a postive active mother node.

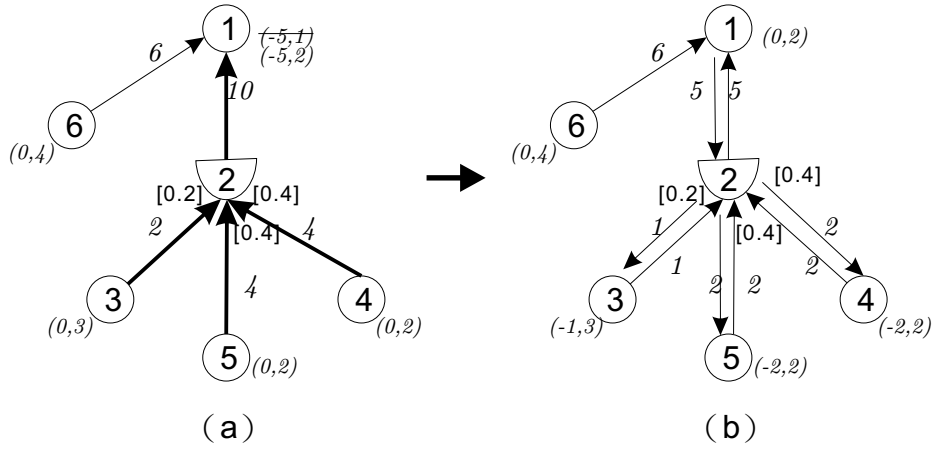


Figure 5.8: Pushing flows via a D-node from a negative mother node.

$d(j) + 1$.

On the other hand, for a negative active member node i connecting to a D-node k, if the mother node j has $d(i) = d(j) - 1$ and positive r_{jk} and r_{ki}, node i can receive $\min\{r_{ki}, -e(i)\}$ flow from the mother node j. In the mean time, all other member nodes also have to receive flows from j following the distillation constraints. Such an operation may force some member nodes to become positive active. If $d(i) > d(j) - 1$, we may relabel $d(i)$ to be $d(j) - 1$. See Figure 5.9 and Figure 5.10 for examples.

Similar to the case of pushing flow from the mother node to the member nodes, the operation surely works when all member arcs are admissible. However, in the case when not all member arcs are admissible, whether we should perform the push operations may lead to different results. Here by (M2) we propose a strategy that allows the push operation continues even if some nodes have no sufficient positive excess. However, we are not absolutely sure that this strategy will work for all cases.

Here we will propose 3 more strategies to push flows from member nodes to the mother node. Again, these strategies are proposed here for a first-try purpose, rather than a correctness guarantee. Some strategy that guarantees correctness of the algorithm should also be investigated in the future research.

(Strategy 1) Decreasing the distance label of mother node.

Since flow has to stream downhill, when distillation constraints force other member node to send flow to its mother node, the distance label of a mother node should be smaller then the distance label of its member nodes. Thus a

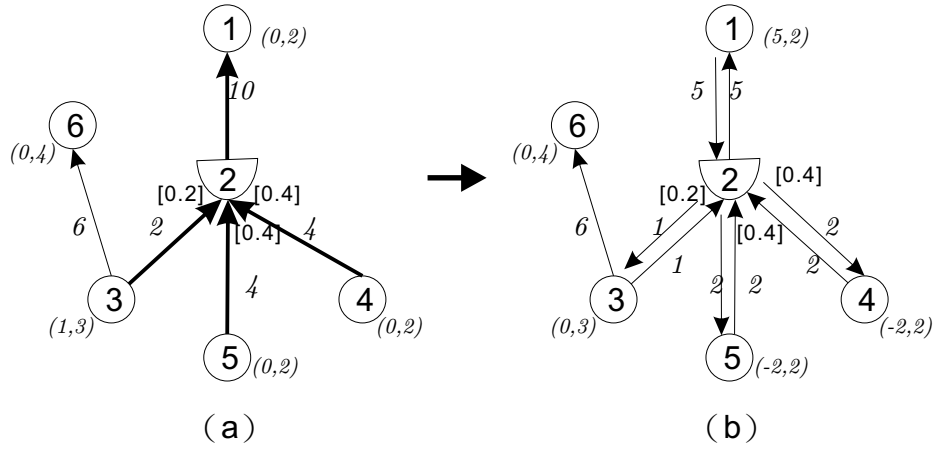


Figure 5.9: Pushing flows via a D-node from a positive active member node.

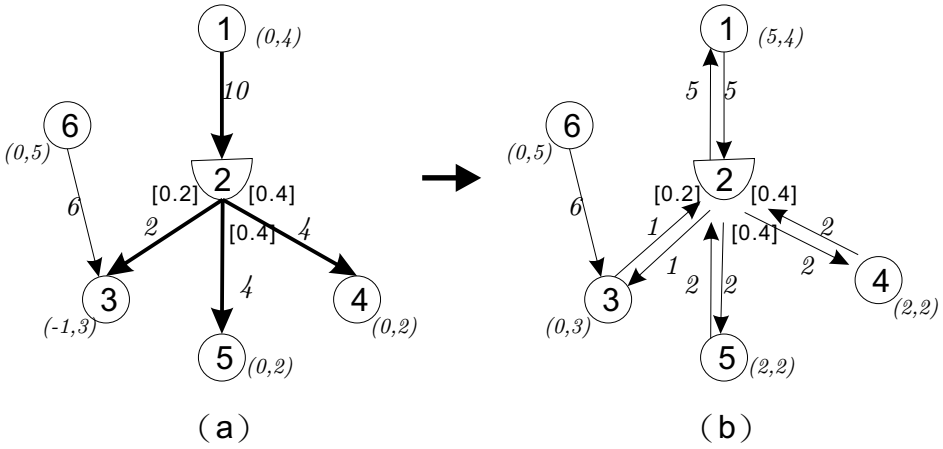


Figure 5.10: Pushing flows via a D-node from a negative active member node.

possible strategy is to decrease the distance label for a mother node. However, in our tests, not all cases work. For example, an S-node could be a mother node, and changing its distance may lead to a non-optimal result.

Figure 5.11 illustrates the processes of our modified preflow push algorithm that uses this strategy. Notice that there are two choices for the active node 4 to push flow in Figure 5.11(h), since $d(6) = d(1) = d(4) - 1$. If we push flow without passing through a D-node, i.e. along arc $(6, 4)$, the maximum flow is 3 as shown in $e(6)$ in Figure 5.11(i). Alternatively, if we push flow through a D-node, i.e. along arc $(2, 4)$ as illustrated in Figure 5.12, the maximum flow becomes 4.5 as shown in Figure 5.12(e). In this case, the optimal solution is obtained by the latter choice that pushes flow along arc $(2, 4)$, but we could not find rules to rule out the first choice that leads to a non-optimal solution. Similarly, for the example in Figure 5.13, which obtains the maximum flow as 7; if we choose to push flow through a D-node in Figure 5.13(d), the result illustrated in Figure 5.14 gives a non-optimal flow value as 6.

Therefore, the suggestion of decreasing the distance label for a mother node may not work for some cases and require more investigation for future research.

(Strategy 2) Increasing the distances label of all member nodes.

To obey the downhill property, besides the previous strategy that decreases the distance label of the mother node, another way is to increase the distances of all member nodes so that all member arcs become admissible. Thus in this strategy we increase the distance labels of all member nodes to be one unit smaller than the distance label of the mother node. Unfor-

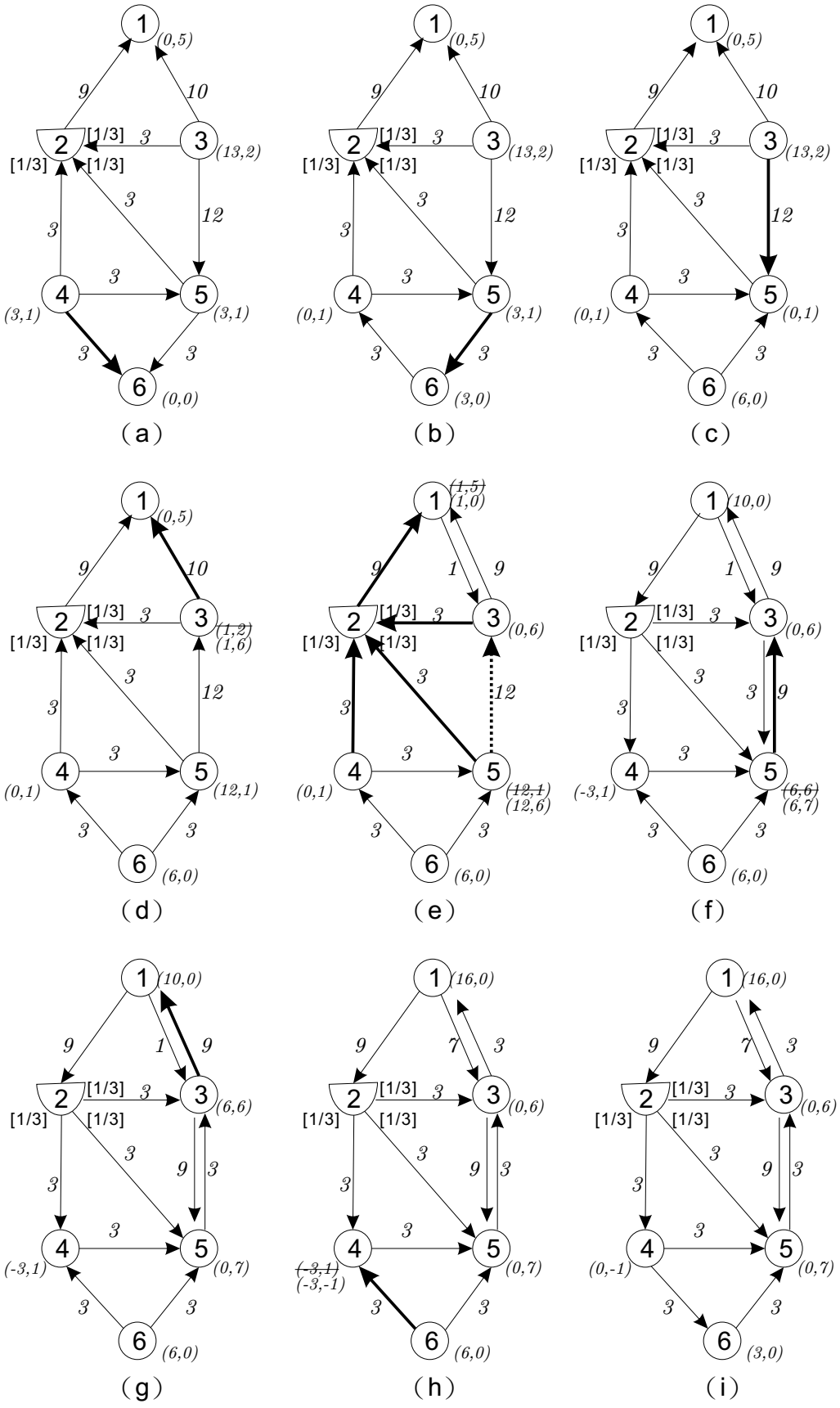


Figure 5.11: A non-optimal case results from flow streaming along the arc toward an O-node directly.

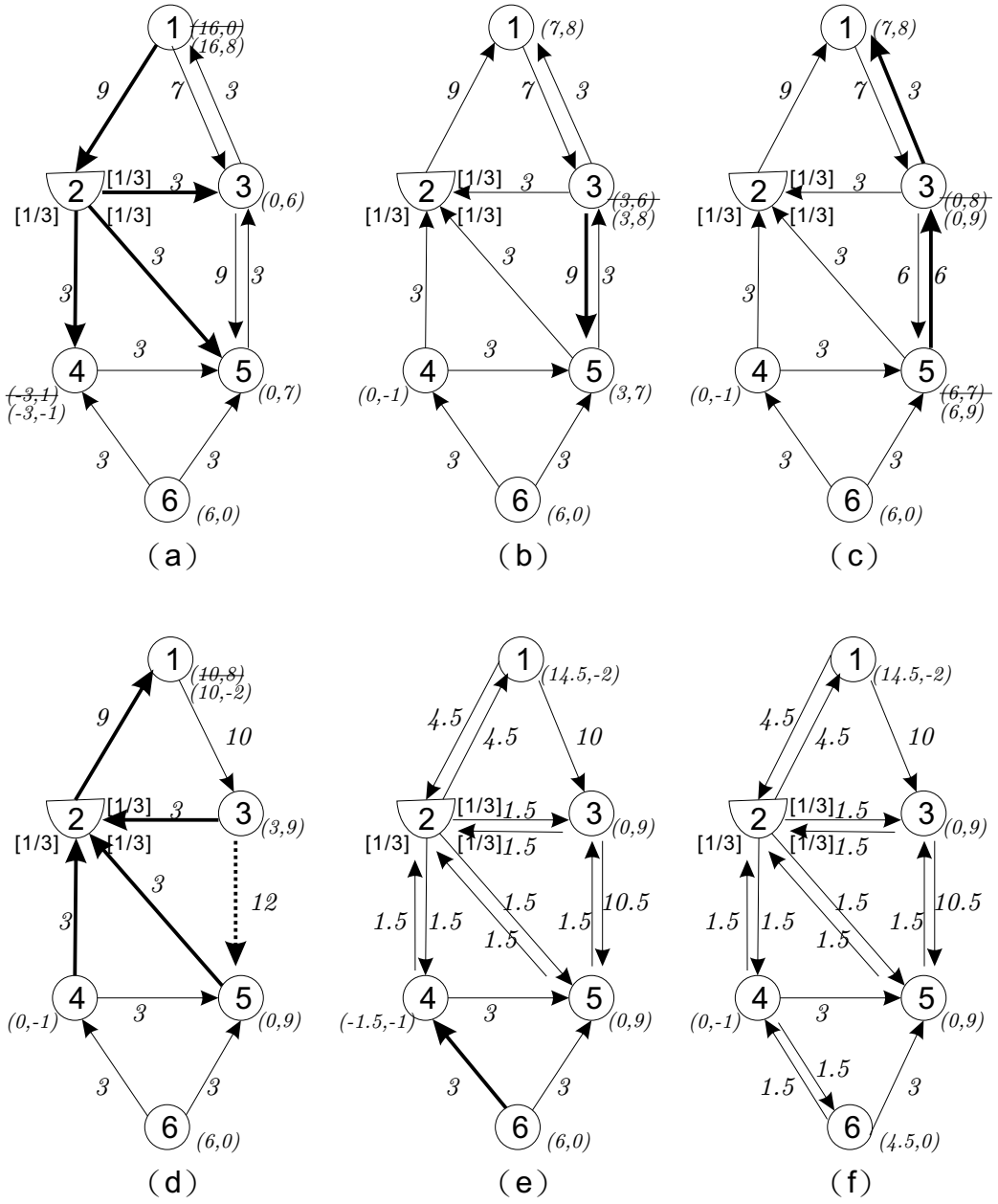


Figure 5.12: An optimal case results from flow streaming along the arc toward an O-node directly.

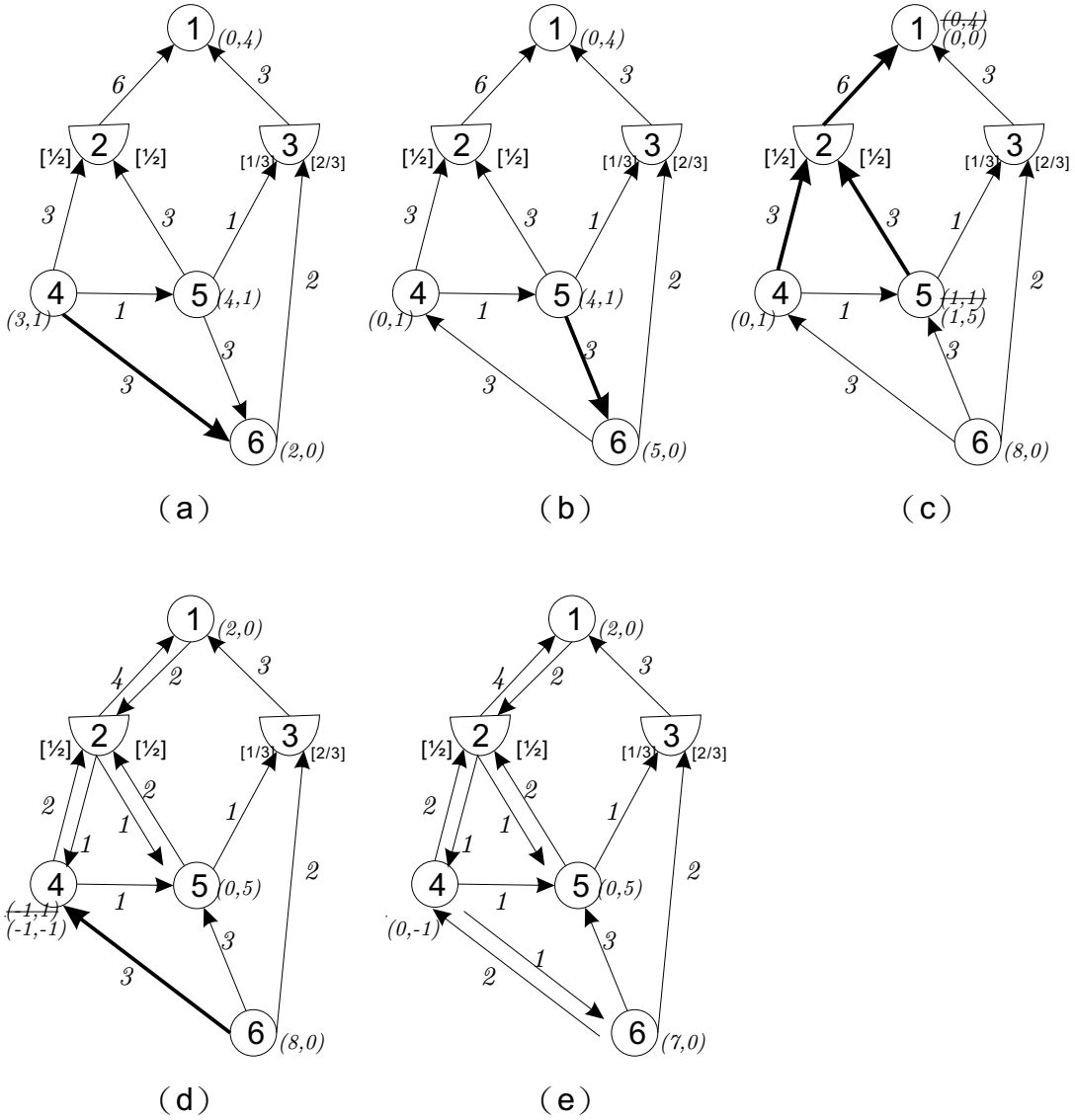


Figure 5.13: A non-optimal case results from pushing through a D-node, instead of O-nodes only.

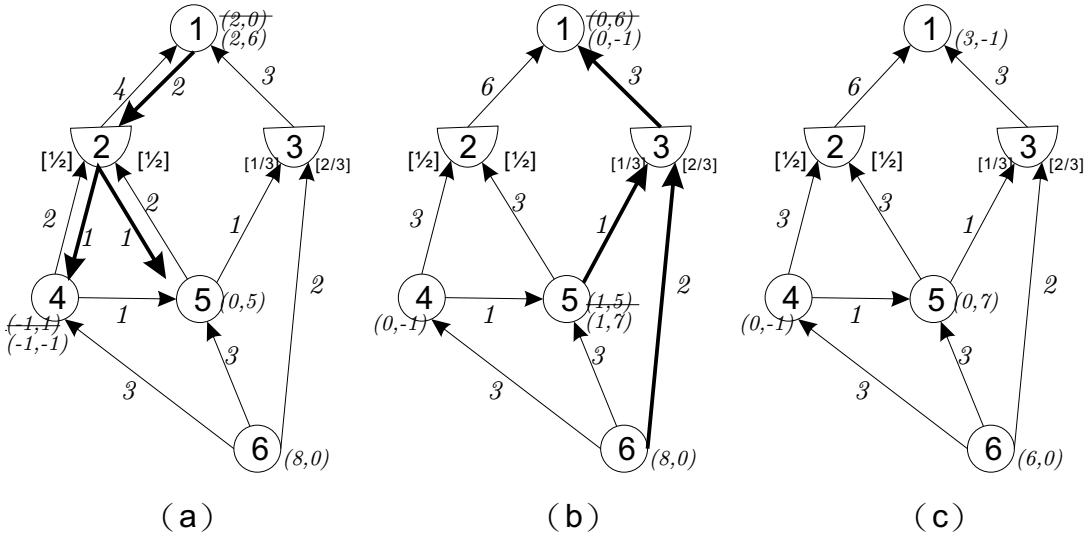


Figure 5.14: An optimal case results from pushing flow through a D-node, instead of O-nodes only.

tunately, this strategy might lead to a cycling in our tests. In particular, Figure 5.15(d) is exactly the same as Figure 5.15(f) even after the iteration of Figure 5.15(e). So, using this strategy may make the algorithm repeats some operations infinitely. Intuitively, this may be a good sign since at least it will not lead a non-optimal solution as other strategies may affect the algorithm. However, how to give a anti-cycling rule is still not clear and requires further investigation, if this strategy is applied.

(Strategy 3) Keeping distance label unchanged for nodes that ship flow passively.

Since the above two strategies may still fail, we propose a somehow strange by interesting idea: we classify the member nodes into two groups, one "active" group contains the active member node only, while the other "passive" group contains all other member nodes. For the active member node, since it is the one to push flows, of course it requires relabel operation when necessary. For other passive member nodes, since they simply ship flow passively

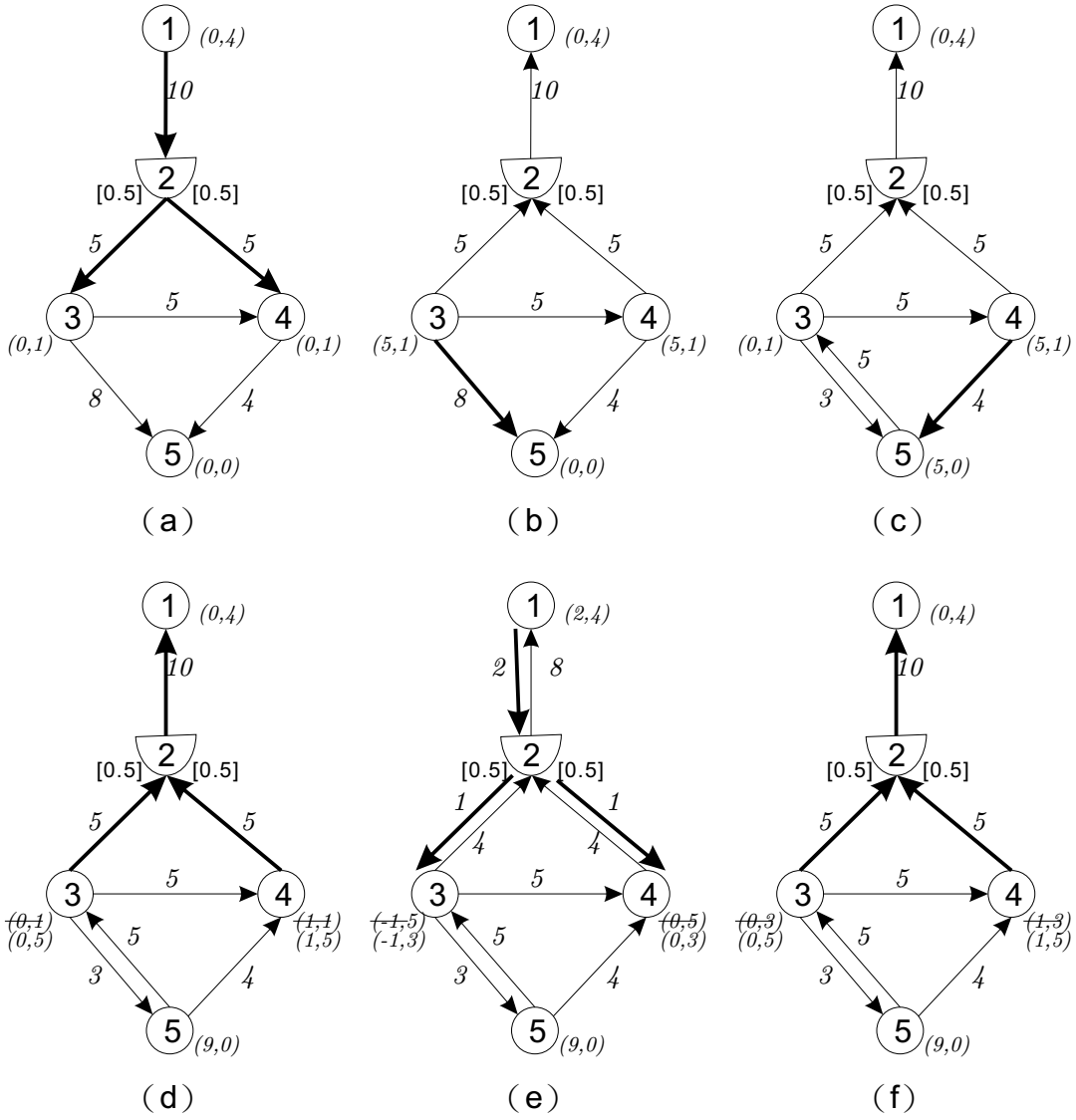


Figure 5.15: A cycling is encountered by applying strategy 2.

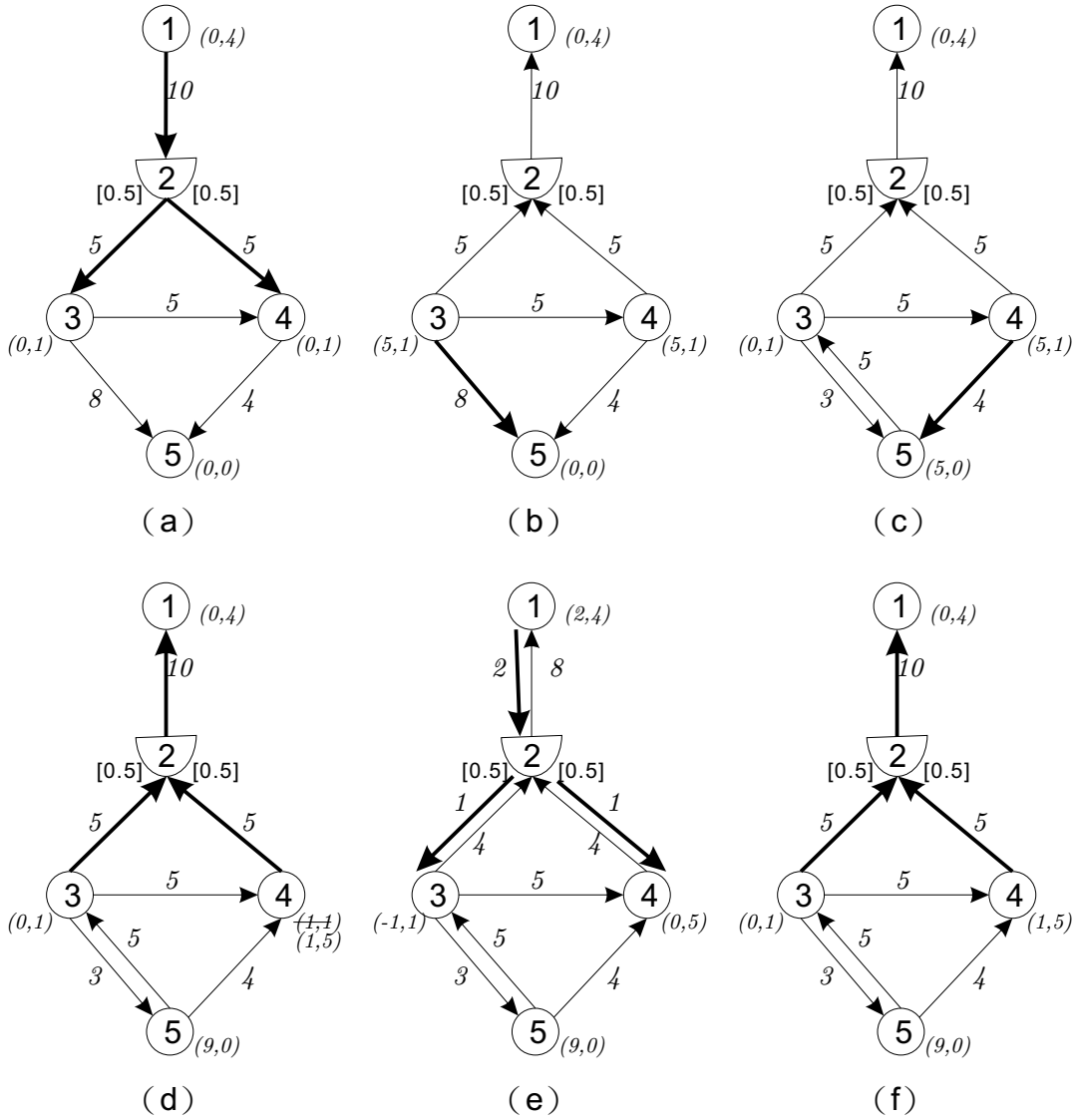


Figure 5.16: A cycling is encountered by applying strategy 3.

to obey the flow distillation constraints, we will not relabel their distance label.

Unfortunately, in our tests it may not work sometimes. For example, a cycling has been encountered as shown in Figure.5.16(d), Figure.5.16(e) and Figure.5.16(f), when this strategy is applied.

5.1.3 Summary of modified preflow-push algorithm

We have encountered problems when modifying preflow push algorithm to solve the maximum flow problem in a distribution network. The original preflow-push algorithm always maintains a condition of $d(i) \leq d(j) + 1$ for each arc (i, j) in a residual network, which guarantees the algorithm to terminate in optimality. However, in our cases, this condition may not hold any more due to the distillation constraints. We have proposed several ways to adjust the distance label so that the downhill property can be hold, but our methods still can not give optimality guarantees.

The appearance of D-nodes truly make the maximum flow problem much harder to solve. We suggest the future researchers to focus on problems such as how to satisfy the distillation constraints and downhill property at the same time, and how to make the correct choice when one encounters problems such as which active nodes or which admissible arcs to choose to guarantee the optimality condition.

5.2 Max-flow min-cut theorem in distribution networks

One may use the max-flow min-cut theorem to guarantee the optimality for a max-flow algorithm for solving conventional max-flow problems. In particular, when a $s - t$ maximum flow has been obtained, one must be able to identify several cuts whose removal will disconnect the s and t, and among those cuts there exist a cut set with the minimal sum of capacity which must equal to the maximum amount of flow shippable from s to t. Here for the max-flow problem in distribution networks, we also wish to derive some theorem similar to the max-flow min-cut theorem so that we can use it to guarantee the correctness of our max-flow algorithm.

In traditional networks, there is a clear $s - t$ cut to separate the source and the sink. However, in a distribution network it is possible that in a max-flow there still

exists some path in a residual network. Take Figure 5.17(a) for example, the flow is already optimal, yet in its residual network there exists a path $1 \to 3 \to 4 \to 7 \to 9$ or $1 \to 3 \to 4 \to 7 \to 8 \to 9$ connecting the source node 1 and sink node 9. This makes the conventional max-flow min-cut theorem fail. Mathematically speaking, this observation is an affect of the distillation constraints. Take a closer look at Figure 5.17(a), only arcs $(1, 2)$ and $(5, 8)$ are saturated. Arc $(5, 8)$ makes paths $1 \to 3 \to 4 \to 7 \to 9$ and $1 \to 3 \to 4 \to 7 \to 8 \to 9$ not be able to augment flows from 1 to 9, or otherwise the distillation constraints associated with node 3 or the flow balance associated with node 5 will be violated.

Although there is no clear $s - t$ cut for a max-flow in a distribution network, by using the optimal dual variables associated with nodes and arcs, we are able to obtain interesting observations where a distribution network may be divided into several components and s and t will be separated in different components. In some sense, we think this observation is similar the conventional $s - t$ cut where s and t are located in different components after we remove all the arcs in a min-cut.

In particular, we associate the flow balance constraints for node i by dual variables $-\pi_i$, capacity constraints for arc (i, j) by $-\gamma_{ij}$ and flow distillation constraints by $-\rho_{ij}$. By adding artificial arcs (t, s) with $c_{ts} = -1$ and $u_{ts} = \infty$ from each sink node t in T back to each source node s in S, the dual formulation of a distributed maximum flow

problem can be formulated as follows:

$$\min \sum u_{ij}\gamma_{ij} \tag{5.1}$$

$$s.t. \pi_j - \pi_i \leq \gamma_{ij} \quad \forall (i,j) \in A, \text{ and } i \in O,\ j \in O \tag{5.2}$$

$$\pi_j - \pi_i + \sum k_{jk}\rho_{jk} \leq \gamma_{ij} \quad \forall (i,j) \in A, \text{ and } i \in O,\ j \in D \tag{5.3}$$

$$\pi_j - \pi_i - \rho_{ij} \leq \gamma_{ij} \quad \forall (i,j) \in A, \text{ and } i \in O,\ j \in D$$

$$\pi_j - \pi_i \leq -1 \quad \forall i \in T, \forall j \in S \tag{5.4}$$

$$\pi_i, \rho_{ij}\ free \tag{5.5}$$

$$\gamma_{ij} \geq 0 \tag{5.6}$$

In this research, we have implemented a random network generator in Chapter 4 to generate many compacted distributed networks. Then we use $CPLEX$ to solve the distributed max-flow problem and get its optimal primal and dual solutions. We observe that removing member arcs (i,j) of positive ρ^*_{ij} as well as its mother arc (l,i) and removing arcs (i,j) of nonzero γ^*_{ij} would divide a distribution network into several components. See Figure 5.17 for example. Suppose node 1 is the S-node and node 9 is the T-node. Figure 5.17(a) lists optimal primal solutions (i.e. arc flows x^*) and Figure 5.17(b) lists the optimal dual solutions $-\pi^*$, $-\rho^*$ and $-\gamma^*$. In Figure 5.17(b), we use dotted arcs to represent arcs with $\gamma^* \neq 0$ and dashed arcs for both member arcs with $\rho^* > 0$ and its mother arc. Notice that $-\rho^*_{34}$ associates with both arc $(3,4)$ and arc $(1,3)$, since dual variable $-\rho^*_{34}$ corresponds to flow distillation constraint $x_{34} = 0.2x_{13}$. After we remove both the dotted arcs and dashed arcs, the network can be divided into several components as shown in Figure 5.17(c). We observe that the nodes in the same component will have the same π^* value and the arcs in the same component will have $\gamma^* = 0$ and $\rho^* = 0$. The results show that the S-nodes and the T-nodes will be located in different components, implying that they can not communicate any more, if

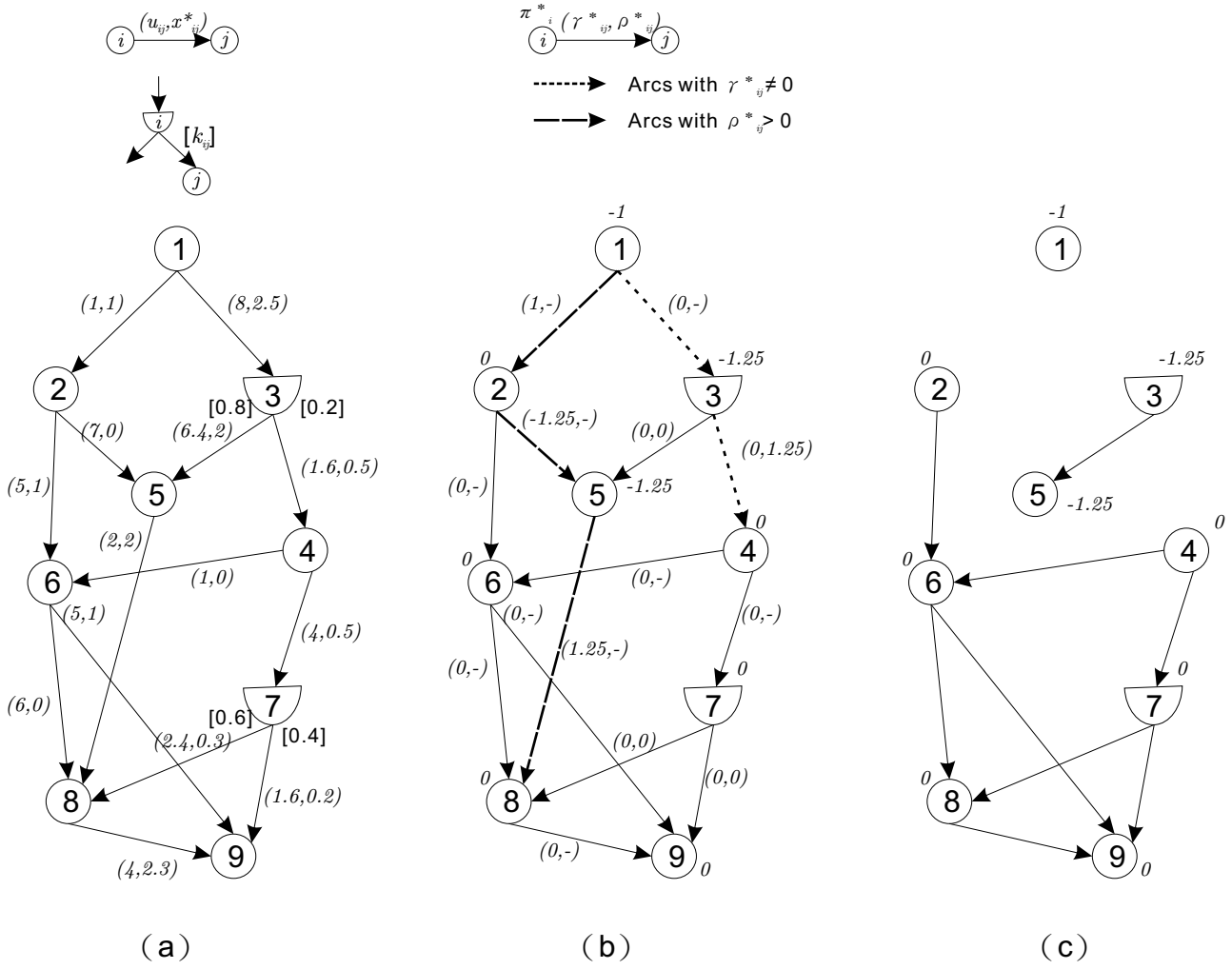


Figure 5.17: An example of components for a distributed max-flow problem.

we remove those dotted arcs and dashed arcs. Thus the set of those dotted arcs and dashed arcs is similar to a conventional $s-t$ cut. We have tested several cases and this observation holds for all of our tests. We conjecture this observation may always hold and it may also be a good lead to a new max-flow min-cut theorem. One may have to redefine the cut set for a distributed max-flow problem.

5.2.1 Summary

The first part of this Chapter proposes several strategies to modify the original preflow-push algorithm for solving the distributed max-flow problem. Although the proposed strategies do not work for all cases, we hope our experience could encourage more researchers to investigate this hard problem.

The second part of this Chapter records our observations and suggestions for interpreting the max-flow min-cut theorem that does not hold for an optimal distributed max-flow solution.

CHAPTER VI

CONCLUSION

Optimization problems in a distributed network are important and applicable for real applications. For example, a manager of a logistics company may estimate the maximum amount of materials purchasable or products manufacturable for the supply chain network of his company by solving a distributed max-flow problem.

This thesis focus on issues in solving the distributed max-flow problems. We first contribute a polynomial time network compaction algorithm in Chapter 3 which can be used as a preprocessing procedures to transform the original problem into an equivalent problem of smaller size. The techniques of our network compaction algorithm can also be generalized to solve the *min-cost distribution network flow problems* which are min-cost network flow problems on distribution networks. Another major contribution of this thesis is the construction of a random distribution network generator in Chapter 4. With our random network generator, we are able to generate compacted distributed max-flow problems for computational tests and evaluation. Again, the techniques we developed for our random network generator can also be generalized for producing compacted distribution networks for min-cost distribution network flow problems. The theoretical properties of our network generator that we have demonstrated and shown in Chapter 4 are useful and can be used for related future research. The third contribution

of this thesis is to give more detailed procedures of Ting's multi-labeling method (2005). Although we have failed in solving the distributed max-flow problem using the preflow-push algorithm, we have proposed several possible modifications. Furthermore, we observe a possible way to define a cut so that the max-flow min-cut theorem may be modified to hold for an optimal solution in a distributed max-flow problem.

With our limited experience, we suggest the following topics for future research in the related fields:

1. Modification on max-flow min-cut theorem

   To guarantee the correctness of any combinatorial max-flow algorithm, one should develop an optimality condition and show the solution obtained by the algorithm satisfies the optimality condition. The max-flow min-cut theorem gives optimality condition for conventional max-flow problems, but has to be modified for solving distributed max-flow problems. Deriving a correct modified max-flow min-cut theorem will help to design correct combinatorial algorithms for solving distributed max-flow problems.

2. Efficient combinatorial-type distributed max-flow algorithms

   Ting's multi-labeling algorithm (2005) is the only combinatorial-type algorithm in the literature for solving a distributed max-flow problem, but it is still not efficient. A distributed max-flow problem surely can be solved in polynomial time, if an efficient interior point method is applied. However, since combinatorial-type algorithms are usually efficient in practice for solving network related problems, the development of efficient combinatorial-type algorithms for solving this problem should be important and challenging.

REFERENCES

Ahuja, R. K., Magnanti, T. and Orlin, J. *Network flows: theory, algorithms and applications.* Englewood Cliffs, New Jersey, U.S.A.: Prentice Hall, 1993.

Anderson, R. J. and Setubal, J. C. 1993. Goldberg's algorithm for maximum flow in perspective: a computatioinal study. In D. S. Johnson and C. McGeoch (Eds.), *Network flows and matching: first dimacs implementation challenge* (pp. 1–17). American Mathematical Society.

Chen, S. Y. and Chern, C. C. 2000. *Network flow problems for supply chain management with product tree structure.* Unpublished doctoral dissertation, National Taiwan University.

Cherkassky, B. V. and Goldberg, A. V. On implementing push-relabel method for the maximum flow problem. *Algorithmica*, **19**, 390–410, 1997.

Dinic, E. A. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl.*, **11**, 1277–1280, 1970.

Edmonds, J. and Karp, R. M. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the Association for Computation Machine*, **19**, 248–264, 1972.

Fang, S. C. and Qi, L. Manufacturing network flows: A generalized network flow

model for manufacturing process modeling. *Optimization Methods and Software*, **18**, 143–165, 2003.

Ford, L. R. and Fulkerson, D. R. Maximal flow through a network. *Canadian Journal of Mathematics*, **8**, 399–404, 1956.

Fujishige, S. A maximum flow algorithm using ma orderings. *Operations Research Letters*, **31**, 176 –178, 2003.

Fulkerson, D. R. and Dantzig, G. B. Computation of maximum flow in networks. *Naval Research Logistics Quarterly*, **2**, 277 – 283, 1955.

Goldberg, A. V. and Tarjan, R. E. A new approach to the maximum flow problem. *Journal of the Association for Computation Machine*, **35**, 921–940, 1988.

Goldfarb, D. and Grigoriadis, M. D. A computational comparison of the dinic and network simplex methods for maximum flow. *Annals of Operations Research*, **13**, 83–123, 1988.

Hoffman, A. and Kruskal, J. 1956. Integral boundary points of convex polyhedra. In H. Kuhn and A. Tucker (Eds.), *Linear inequalities and related systems* (pp. 233–246). Princeton, NJ: Princeton University Press.

Karzanov, A. V. Determining the maximal flow in a network by the method of preflows. *Soviet Math. Dokl*, **15**, 434–437, 1974.

Sleator, D. D. and Tarjan, R. E. A data structure for dynamic trees. *Journal of Computer and System Sciences*, **24**, 362–391, 1983.

Ting, M. J. 2005. *Maximum flow problem in the distribution network flow model.* Unpublished master's thesis, National Cheng Kung University.

APPENDICES

APPENDIX A

Illustartion of Random Distribition Network Generator

A.1 Input parameters

n: the total number of nodes

$u_{\max}$: the maximum capacity for an arc

$outdeg\_D_{\max}$: the maximum outgoing-degree for a D-node

$outdeg\_O_{\max}$: the maximum outgoing-degree for an O-node

A.2 Notation

$at\_least\_one\_D$: flag to record if there is any D-node

$d(i)$: the distance from source to node i.

USE: the set of nodes that has been put into the network

$UNUSE := N \setminus USE$: the set of nodes that have not been put into the network,

$LEAF$: the set of nodes in USE without outgoing arcs

$FARTHER(i)$: the set of O-nodes in USE with distance label equal to or larger than $d(i)$

$AVA(i) := UNUSE \cup FARTHER(i)$: the set of nodes that node i can connect to,

$outdeg(i)$: the outgoing-degree for node i

$outdeg_{\max}(i)$: the maximum outgoing-degree for node i

$outdeg_{\min}(i)$: the minimum outgoing-degree for node i

A.3 Data structure

nodeTable

	{O,D} type	{int} distance	{int} degree	{link list} headnode
1				
2				
...				
n				

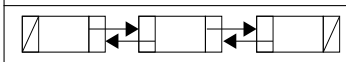
type: the type of node i

distance: the maximum number of arcs from source to node i

degree: the sum of in-degree and out-degree of node i

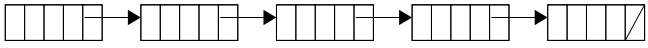
headnode: record all head nodes of node i

distanceTable

	{double link list} node
0	
1	
...	
n-1	

node: record all nodes with the same distance label in the current network

arcTable{single link list}



{int}	{int}	{float}	{float}	{pointer}
tailnode	headnode	factor	capacity	ptr

tailnode: record the tail node of an arc

headnode: record the head node of an arc

factor: record the distillation factor od an arc.

capacity: trecord the capacity of an arc

ptr: a pointer pointing to next arc

LEAF{array of n elements}

{int}	{int}	...	{int}
		...	

UNUSE{array of $n-1$ elements}

{int}	{int}	...	{int}
2	3	...	n-1

AVA{dynamic array }

{int}	{int}	...	{int}
		...	

A.4 Algorithm

Algorithm 4 Network_Generator

```
at_least_one_D:=FALSE
While (at_least_one_D=FALSE) do
    Initialization
    While (LEAF ≠NULL) do
        Pop node i from LEAF
        If (i ≠ t) then
            Construct AVA(i)
            Decide the outgoing-degree for node i
            Choose head nodes randomly for node i
            For each head node j
                Update information
                If (j is from UNUSE) then
                    Construct AVA(j)
                    Decide node type
                    If (j is O-node) then
                        Put node j into LEAF
                    Else then
                        at_least_one_D:=TRUE
                        Decide the outgoing-degree for node j
                        Choose head nodes ks randomly for node j
                        Update information, set nodes ks as O
                        Put nodes ks into LEAF
    End while
    Correct single transshipment O-nodes
End while
```

Procedure: Initialization

```
s:=1
t:=n
nodeTable (s).distance:= 0
nodeTable (s).degree := 0
nodeTable (s).type := 'O'
nodeTable (s).headnode = {NULL}
distancetable (0).node = {s}
Put s into LEAF
```

Procedure: Construct AVA(i)

Add those O-nodes in the current network whose distance $\geq$ node i into AVA
Add those nodes in $UNUSE$ into AVA

Procedure: Decide the outgoing-degree for node i

If (i is an O-node) **then**
    If ($|AVA(i)| \geq outdeg\_O_{\max}$) **then**
        Pick a number between 1 ~ $outdeg\_O_{\max}$
    Else then
        Pick a number between 1 ~ $|AVA(i)|$
Else then
    If ($|AVA(i)| \geq outdeg\_D_{\max}$) **then**
        Pick a number between 2 ~ $outdeg\_D_{\max}$
    Else then
        Pick a number between 2 ~ $|AVA(i)|$

Procedure: Update information

Update nodeTable
Update distanceTable
Update arcTable

Procedure: Decide node type

If ($|AVA(i)| \geq 2$ and $i \neq t$) **then** randomly choose type from $\{O, D\}$
Else then type must be O

Procedure: Correct single transshipment O-nodes

For each node i, except for s and t
    If (degree of node $i = 2$) **then**
        Randomly pick another O-node j which is not adjacent to node i
        Add an arc from the node of smaller distance label to the node of larger distance label
        Update information

A.5 Example

nodeTable

	type	distance	degree	headnode
1	O	0	0	NULL
2				
3				
4				
5				
6				

distanceTable

	node
0	1
1	
2	
3	
4	
5	

$AVA(i) = FARTHER(i) + UNUSE$

arcTable

tail	head	factor	capacity

$UNUSE$

2	3	4	5	6

$LAEF$

1					

user input

$n = 6$

$u_{\max} = 10$

$outdeg\_D_{\max} = 3$

$outdeg\_O_{\max} = 4$

$s = 1, t = 6$

Put s into $LEAF$

nodeTable

	type	distance	degree	headnode
1	O	0	2	2,3
2	O	1	1	NULL
3	D	1	1	NULL
4				
5				
6				

distanceTable

	node
0	1
1	2,3
2	
3	
4	
5	

$AVA(1) = FARTHER(1) + UNUSE$

2	3	4	5	6

$UNUSE$

2	3	4	5	6

$LAEF$

1					

arcTable

tail	head	factor	capacity
1	2	1	4
1	3	1	10

Pop node 1 from $LEAF$

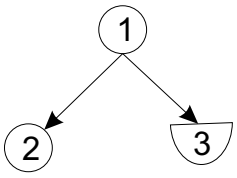
$outdeg(1) = 2$

head nodes=$2, 3$

update information

nodeTable(2).type=O, put node 2 into $LEAF$

nodeTable(3).type=D



nodeTable

	type	distance	degree	headnode
1	O	0	2	2,3
2	O	1	1	NULL
3	D	1	3	4,5
4	O	2	1	NULL
5	O	2	1	NULL
6				

distanceTable

	node
0	1
1	2,3
2	4,5
3	
4	
5	

$AVA(3) = FARTHER(3) + UNUSE$

2	4	5	6

$UNUSE$

		4	5	6

$LAEF$

2					

arcTable

tail	head	factor	capacity
1	2	1	4
1	3	1	10
3	4	0.3	3
3	5	0.7	7

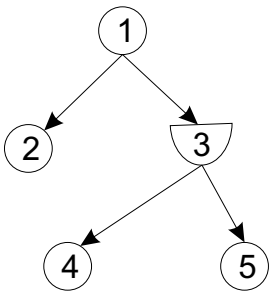
$outdeg(3) = 2$

head nodes=$4, 5$

update information

nodeTable(4).type=O, put node 4 into $LEAF$

nodeTable(5).type=O, put node 5 into $LEAF$



nodeTable

	type	distance	degree	headnode
1	O	0	2	2,3
2	O	1	2	4
3	D	1	3	4,5
4	O	2	2	NULL
5	O	2	1	NULL
6				

distanceTable

	node
0	1
1	2,3
2	4,5
3	
4	
5	

$AVA(2) = FARTHER(2) + UNUSE$

4	5	6

$UNUSE$

				6

$LAEF$

2	4	5			

arcTable

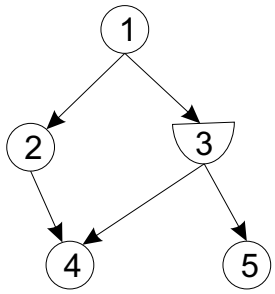
tail	head	factor	capacity
1	2	1	4
1	3	1	10
3	4	0.3	3
3	5	0.7	7
2	4	1	5

Pop node 2 from $LEAF$

$outdeg(2) = 1$

head nodes=4

update information



nodeTable

	type	distance	degree	headnode
1	O	0	2	2,3
2	O	1	2	4
3	D	1	3	4,5
4	O	2	4	5,6
5	O	2	2	NULL
6	O	5	1	NULL

distanceTable

	node
0	1
1	2,3
2	4,5
3	
4	
5	6

$AVA(4) = FARTHER(4) + UNUSE$

5	6

$UNUSE$

				6

$LAEF$

4	5				

arcTable

tail	head	factor	capacity
1	2	1	4
1	3	1	10
3	4	0.3	3
3	5	0.7	7
2	4	1	5
4	5	1	7
4	6	1	10

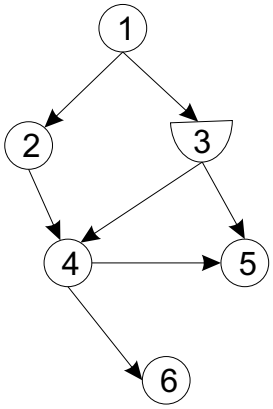
Pop node 4 from $LEAF$

$outdeg(4) = 2$

head nodes=$5, 6$

update information

nodeTable(6).type=O



nodeTable

	type	distance	degree	headnode
1	O	0	2	2,3
2	O	1	2	4
3	D	1	3	4,5
4	O	2	4	5,6
5	O	2	3	6
6	O	5	2	NULL

distanceTable

	node
0	1
1	2,3
2	4,5
3	
4	
5	6

$AVA(5) = FARTHER(5) + UNUSE$

6

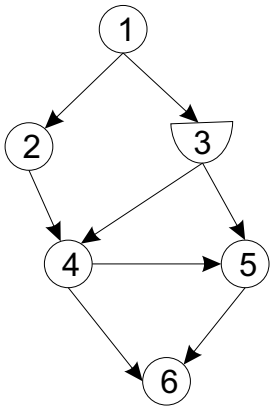
$UNUSE$

$LAEF$

5					

arcTable

tail	head	factor	capacity
1	2	1	4
1	3	1	10
3	4	0.3	3
3	5	0.7	7
2	4	1	5
4	5	1	7
4	6	1	10
5	6	1	8



Pop node 5 from $LEAF$

$outdeg(5) = 1$

head nodes=6

update information

nodeTable(6).type=O, put node 6 into $LEAF$

nodeTable

	type	distance	degree	tailnode
1	O	0	2	2,3
2	O	1	2⇒3	4,5
3	D	1	3	4,5
4	O	2	4	5,6
5	O	2	3⇒4	6
6	O	5	2	NULL

distanceTable

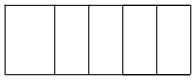
	node
0	1
1	2,3
2	4,5
3	
4	
5	6

$AVA(i) = FARTHER(i) + UNUSE$

arcTable

head	tail	factor	capacity
1	2	1	4
1	3	1	10
3	4	0.3	3
3	5	0.7	7
2	4	1	5
4	5	1	7
4	6	1	10
5	6	1	8
2	5	1	6

$UNUSE$



$LAEF$

Pop node 6 from $LEAF$

Because node 6 = t, skip node 6

Now $LEAF$ is empty

Check if any transshipment O-node exists

nodeTable(2).degree=2

randomly chosen node=5

Then connect node 2 with node 5

Update information