Summer 2017

Applications of Flow Network Models in Finance

Angel J. Woods

APPLICATION OF FLOW NETWORK MODELS IN FINANCE

by

ANGEL WOODS

(Under the Direction of Hua Wang)

ABSTRACT

In this thesis we explore the applications of flow networks in practical problems in finance. After introducing basic definitions and background information, we first survey some known applications of flow networks in theoretical mathematics. We also briefly comment on their potential applications in the setting of financial flow networks. We then construct networks from practical financial flows and present the construction, reasoning, and known applications. Lastly, we show a design of financial flow networks that takes time into consideration and discuss its applications.

INDEX WORDS: Flow Network, Finance

2009 Mathematics Subject Classification: 90B10

APPLICATION OF FLOW NETWORK MODELS IN FINANCE

by

ANGEL WOODS

B.S., Spelman College, 2013

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

STATESBORO, GEORGIA

APPLICATION OF FLOW NETWORK MODELS IN FINANCE

by

ANGEL WOODS

Major Professor:	Hua Wang
Committee:	Zhan Cheng
	Ionut Iacob

Electronic Version Approved:
July 2017

DEDICATION

This thesis is dedicated to my family and friends who have supported me throughout my academic journey.

ACKNOWLEDGMENTS

I wish to acknowledge my family, friends, and academic advisor.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Background and motivation

Networks are natural modeling tools to show how people conduct their economic activities such as the flow of transportation, modeling currents in an electrical circuit, and representing financial networks. This paper examines the use of flow networks in practical applications in theoretical mathematics including edge and vertex disjoint paths, maximum-matching problems, and project selection problems. We then use flow networks to model financial flow networks and present the construction and reasoning behind the application. Taking time into consideration when modeling financial flow networks, our construction may be used to analyze and solve financial problems that may occur in a financial network.

1.2 Basic Graph Theory

We begin with some basic definitions and terminologies used throughout the paper.

Definition 1. *A graph G consists of a pair (V, E) such that V is the set of vertices and E is the set of edges. Henceforth, we will write $V(G)$ and $E(G)$ for the set of vertices and edges in the graph G, respectively. We denote two vertices, u and v, joined by an edge e as uv or vu. If e is an edge and u and v are vertices such that $e = uv$, then e is said to join u and v.*

For instance, Example 1.2.1 defines a graph and Figure 1.1 shows its structure.

Example 1.2.1. *Let*

$$G = (V(G), E(G))$$

where

$$V(G) = (v_1, v_2, v_3, v_4),$$

$$E(G) = (e_1, e_2, e_3, e_4, e_5, e_6).$$

And $e_1, \ldots, e_6$ *are defined by*

$$e_1 = v_1v_2, \quad e_2 = v_2v_3, \quad e_3 = v_3v_4, \quad e_4 = v_4v_1, \quad e_5 = v_1v_3, \quad e_6 = v_2v_4.$$

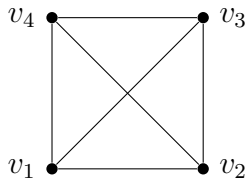

Figure 1.1: Example 1.2.1

Example 1.2.2 defines a graph shown in Figure 1.2.

Example 1.2.2. *Let*

$$H = (V(H), E(H))$$

where

$$V(H) = (v_1, v_2, v_3),$$
$$E(H) = (e_1, e_2, e_3).$$

And $e_1, \ldots, e_3$ *are defined by*

$$e_1 = v_1v_2, \quad e_2 = v_2v_3, \quad e_3 = v_3v_1.$$

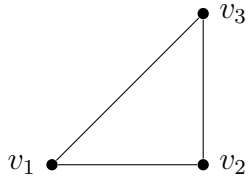

Figure 1.2: Example 1.2.2

Definition 2. *A graph $H = (V(H), E(H))$ is a subgraph of graph $G = (V(G), E(G))$ if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$, where $V(H)$ and $H(H)$ are the sets of the vertices and edges of the graph H, respectively. For example, the graph in Figure 1.2 is a subgraph of that in Figure 1.1.*

A walk in a graph is a finite or infinite sequence of edges that connect a sequence of vertices with the property that each vertex in the sequence is adjacent to the next vertex in the sequence. A walk that does not repeat vertices is called a path. In Figure 1.1, the sequence v_1, e_1, v_2, e_2, v_3 is considered a path in the graph G.

We call a graph a *directed graph* if each edge has a direction associated with it. For instance, Figure 1.3 shows the graph from Example 1.2.2 with directed edges. More specifically, a directed path is a sequence of vertices in which there is a directed edge from each vertex in the sequence to the next vertex in the sequence.

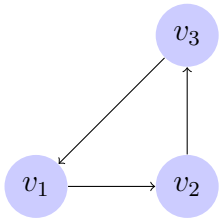

Figure 1.3: The graph defined in Example 1.2.2.

1.3 Network flows

A flow network is a directed graph in which each edge has a capacity and each edge receives a flow. It consists of a sink node s and source node t, in which (s) only has outgoing flow and (t) only has incoming flow.The flow in the flow network is a value assignment to each edge in the network such that:

(a) the flow cannot exceed the capacity assigned to an edge;

(b) the total inflow of each node that is neither the sink node nor the source node must be equal the total outflow of that node.

Flow networks can be used to model traffic in road systems, fluid pipes, currents in an electrical circuit, the representation of financial networks, etc.

For example, in the case of the state aid distribution, we present an example of the network flow system. A typical state will have four such nodes including: county, city, special district and school district. Table 1.1 explains the basic purposes of each type of local governments and the transfer of funds between them.

Government type	**Purpose**	**Transfers funds to**
County	General services	City, School and Special Districts
City	General services	School and Special Districts
School Districts	Education	None
Special Districts	Specific function	None

Table 1.1: Major types of local governments

Figure 1.4 shows the network presentation for the structure described in such a system. Here the source node s represents the state government with dotted edges to each local district, and the sink node t represents the state government that collects revenue from each local district also through dotted edges. The nodes in between s and t represent the local districts, and the solid edges represent the possible transfer of funds between them.

1.4 A More Detailed Example

Imagine that you are a courier service that wants to deliver cargo from one city to another. You decide that you want to deliver this cargo using different flights. However, each flight has allotted you a limited amount of space to be used for your deliveries. Our question then

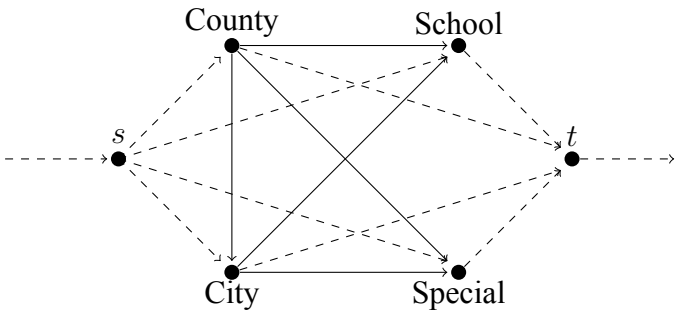

Figure 1.4: The constructed flow network of Table 1.1

becomes: How much of your cargo can be shipped to each destination using the various flights available? To answer this, we can use a flow network to model such a problem.

Recall, that a network contains a source node (s) and a sink node (t). In this case, our source node will be the main location that the cargo is being shipped from, and the sink node is the final destination city. Each vertex in the graph will represent a city where we can send or receive cargo, and each edge will have a finite capacity that represents how much cargo we can send on each flight. Figure 1.5 is an example of a flow network modeling our situation.

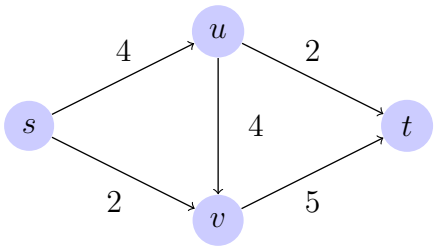

Figure 1.5: A simple flow network with a capacity assigned to each edge.

Using this graph, we want to find out how much cargo can be shipped from s to t. This now becomes a maximum flow problem. A straightforward approach is to keep finding paths from s to t where we can send cargo or flow through, send as much cargo as possible along each path, and then update the graph to account for the flow or space that has already been used. Figure 1.6 shows a random selection of a path from Figure 1.5. The first number

on the edge is the flow, and the second number is the capacity.

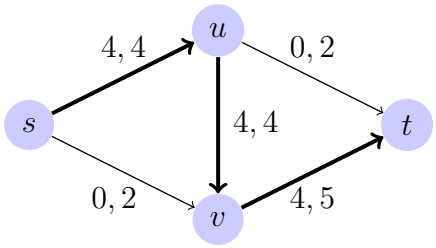

Figure 1.6: A path from s to t with positive flow.

In Figure 1.6 , we chose the path $s \to u \to v \to t$. The capacities along the path are 4, 4, 5 respectively. Since the flow cannot exceed the value of the capacity assigned to each edge, we can send at most 4 units along this path. Now we must update the graph. In order to do this, we must take into consideration the capacity that has already been used along each edge. We have used 4 available spaces along each edge, so we must decrease the capacity of each edge used by 4. By doing this, we see that the paths from $s \to u$ and $u \to v$ are both saturated, which here means that the maximum capacity has been reached for those edges. Therefore, the only other path from s to t is $s \to v \to t$. Note that since the graph has been updated, the capacity of the edge (v, t) is now 1. Thus, the capacities along the second path are now 2 and 1. Following the same procedure as we did before, we update the graph to obtain Figure 1.7.

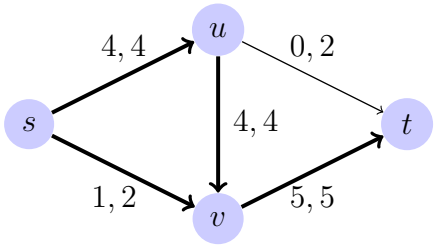

Figure 1.7: The current flow.

We can no longer find a path from s to t with only forward unsaturated edges.We were able to send a total of 5 units. However, there is a better solution to our problem that

will allow us to distribute more cargo from s to t. Before, we chose to send the maximum capacity allowed on each edge in the path we chose first. Now lets choose to send 3 units along the path $s \to u \to v \to t$. This allows us to send 1 unit on the path $s \to u \to t$ and 2 units along the path $s \to v \to t$. Following this modification, we increased our total flow to 6 units, which is the maximum flow possible. The optimal solution is shown in Figure 1.8.

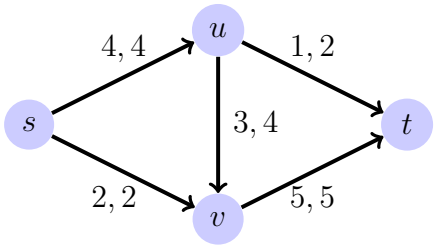

Figure 1.8: The optimal solution.

1.5 Max flow-Min Cut Algorithm

Our goal is to find the maximum flow from the source node to the sink node that satisfies the capacity restriction in the flow network.

Essentially, we have the function

$$f(x) = \sum_{(i,j)\in E(s)} x_{ij} \tag{1.1}$$

and want to maximize (1.1) under the following conditions:

- $$\sum_{\{j:(i,j)\in E\}} x_{ij} - \sum_{\{j:(j,i)\in E\}} x_{ji} = 0 \qquad \forall i \in V\backslash\{s,t\};$$
- $$0 \le x_{ij} \le u_{ij} \qquad \forall (i,j) \in E.$$

In what follows we mainly follow the notations and terminologies in [3].

Definition 3. *A cut in a network G is a set of edges C such that every path from the source node s to the sink node t uses exactly one edge from C, and the removal of all edges in C disconnects G into two parts.*

The capacity of a cut C, denoted $c(C)$,

$$c(C) = \sum_{e \in C} c(e), \tag{1.2}$$

is the sum with potential negative terms of the capacities of each edge in the cut set. The minimum cut is the one with minimum capacity.

Theorem 1.5.1. *Suppose in a network all edge capacities are integers. Then the value of the maximum flow is equal to the capacity of the minimum cut. Moreover, there is a maximum flow f for which all $f(e)$ are integers.*

This theorem can be proved using the Ford-Fulkerson Algorithm, which is a greedy algorithm that computes the maximum flow in a flow network. The idea behind this algorithm is that as long as there is a path from the source node to the sink node, with available capacity on all edges along this path, we send flow along one of the paths. These paths are called *augmenting paths*, constructed by repeatedly finding a path with available capacity from the source node to the sink node. This continues until no more augmenting paths can be found.

We begin the algorithm by letting $G(V, E)$ be a graph such that $c(u, v)$ is the capacity and $f(u, v)$ is the flow. We want to find the maximum flow from the source s to the sink t. We follow each step below until we can no longer find an augmenting path.

- Initialize all flow to zero
- While there exists an augmenting path p from s to t in the graph G, such that $c_f(u, v) > 0$ for all edges $(u, v) \in p$:

- Find $c_f(p) = \min c_f(u, v) : (u, v) \in p$
- For each edge $(u, v) \in p$

  (1) $f(u, v) \leftarrow f(u, v) + c_f(p)$ (*send flow along the path*)

  (2) $f(u, v) \leftarrow f(u, v) - c_f(p)$ (*the backwards flow*)

By using this algorithm, we can compute the capacity of the minimum cut of a network flow. Recall from our example problem above, we found that the optimal solution had a maximum flow of 6 units. Thus, by Theorem 1.5.1, we know that the capacity of the minimum cut will also be 6.

1.6 Applications

In this thesis, we intend to survey the applications of flow networks. In particular, the various practical applications in finance that can be modeled through manipulating theoretical flow network tools. In chapter 2, we explore some known applications of flow networks and maximum flow and the potential use of each one in financial systems. In chapter 3, we use flow network to represent financial systems. This construction provides a visual representation of how funds are transferred throughout a financial system. We can use this model to analyze and solve possible problems that may occur in such a system i.e financial contagion. Lastly, we explore the design of flow network models that consider time.

CHAPTER 2

APPLICATIONS OF MAXIMUM FLOW

In this chapter we explore some known applications of flow networks and maximum flow. We include a brief justification for each application but omit some details. For a more detailed discussion one may see [1]. These applications include:

- Computing the maximum number of edge-disjoint paths using unit capacities and the Ford-Fulkerson algorithm;
- Computing the maximum number of vertex-disjoint paths by transforming a directed graph with both edge and vertex capacities into a traditional maximum flow network with only edge capacities;
- Finding the maximum number of independent edges (edges that do not share any common vertices) in a given bipartite graph by reducing it to a maximum flow problem;
- Finding the largest possible collection of pairs from two disjoint sets in a maximum-matching or the assignment problem in general; and
- Using Maximum Flow and Minimum-Cut in a a project selection problem, where our goal is to find a valid subset of the projects whose total profit would be as large as possible.

We then present the potential application of these theoretical observations in financial networks.

2.1 Edge-Disjoint Paths

One common application of maximum flows is computing the maximum number of edge-disjoint paths between two vertices s and t in a directed graph. Two paths in a graph H

are said to be *edge-disjoint* if they do not share an edge. Edge-disjoint paths may, however, pass through the same vertex.

This problem can be solved by assigning each edge in the graph H a capacity of 1. Then the maximum flow from s to t assigns a flow of 0 or 1 to each edge. Because every vertex in H lies in at most two saturated edges (one coming in, one going out, or no edges at all), the subgraph S of saturated edges is the union of edge-disjoints paths from H. Furthermore, the number of paths will be exactly equal to the value of the flow.

For example, in Figure 2.1 we are given a directed graph H with vertices s and t. We can find the maximum number of edge-disjoint paths using the Ford-Fulkerson Algorithm. We begin by assigning each edge in H a capacity of 1, and initialize the flow to be 0. Next, we find an augmenting path from s to t. We continue doing this until no more augmenting paths can be found.

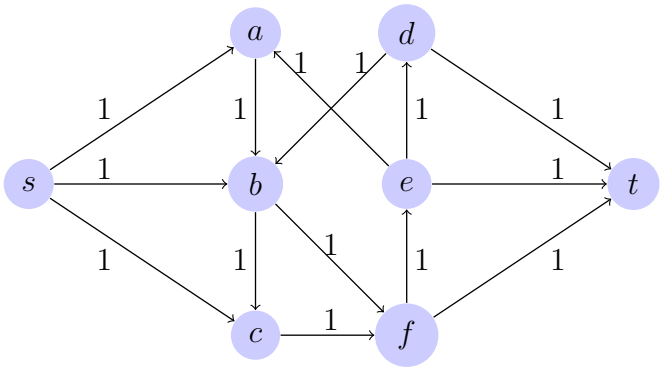

Figure 2.1: A directed graph H.

In Figure 2.2, we have chosen two edge-disjoint paths from the graph H. The first path highlighted in red is $s \to a \to b \to c \to f \to t$, and the second path highlighted in green is $s \to b \to f \to e \to t$. From this, we see that we cannot find anymore augmenting paths from s to t. Thus, the maximum flow from s to t is 2, and hence the maximum number of edge-disjoint paths is 2.

This same algorithm can be used in undirected graphs as well by simply replacing every

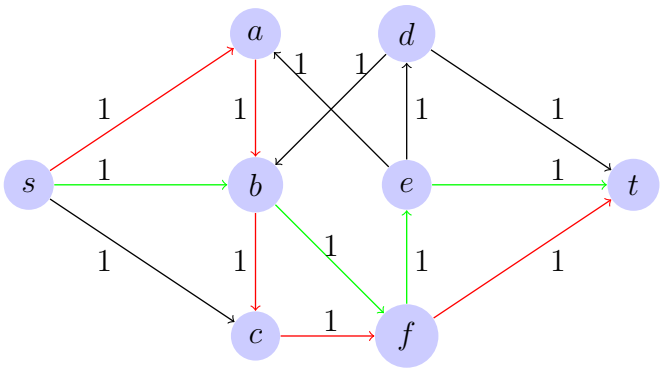

Figure 2.2: A directed graph H with maximum number of edge disjoint paths highlighted.

undirected edge in the graph with a pair of directed edges, assigning each edge a capacity of 1, and computing the maximum flow from s to t. If the maximum flow saturates both edges $u \to v$ and $v \to u$ in the newly formed directed graph, we can remove both edges from the flow without changing the value. Thus, the maximum flow assigns a direction to every saturated edge, and we are then able to remove edge-disjoint paths by looking for saturated directed edges in the graph.

2.2 Vertex Capacities and Vertex Disjoint Paths

Another application of maximum flows is computing the maximum number of vertex-disjoint paths in a directed graph. A set of paths is *vertex-disjoint* if each vertex in the directed graph appears in at most one of the paths. In Figure 2.3, we have a directed graph in which three vertex-disjoint paths have been highlighted. This turns out to be the maximum number of vertex-disjoint paths in this graph.

Now suppose vertices as well as edges have capacities. In addition to the constraints we have on flow networks, the total flow for any vertex v must be at most a non-negative value. Using the new constraints, we want to compute the maximum flow which will be equal to the maximum number of vertex-disjoint paths.

The most practical way of solving this problem is to transform the input into a tradi-

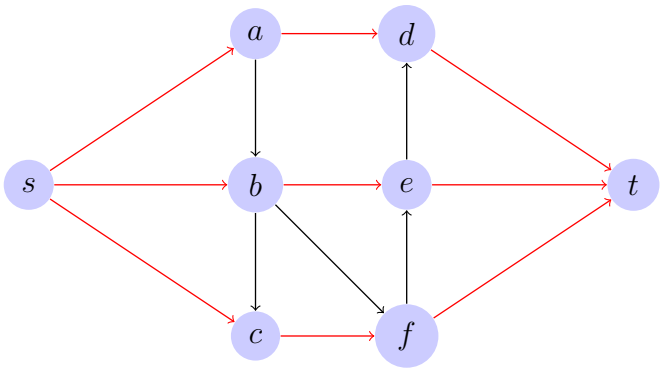

Figure 2.3: A directed graph with 3 Vertex-Disjoint paths.

tional max flow network that has only edge capacities. The idea is as follows:

(1) Split each vertex $v \in G$ into two vertices called v_{in} and v_{out}.

(2) For each vertex v, add an edge of capacity 1 from v_{in} to v_{out}.

(3) Replace every directed edge (u, v) with an edge of capacity 1 from u_{out} to v_{in}.

(4) Compute the maximum flow from s_{out} to t_{in} in the newly constructed network.

It is now easy to compute the maximum flow which is equal to the maximum number of vertex-disjoint paths from s to t in any directed graph. We simply assign each vertex a capacity of 1, and compute the maximum flow.

2.3 Maximum Matchings in Bipartite Graphs

Another application of maximum flows is finding large matchings in bipartite graphs. A matching is a subgraph in which no two edges share the same vertex. The goal is to find a matching with the maximum number of edges in a given bipartite graph.

We solve this by transforming the problem into a maximum flow problem. We start with a bipartite graph G with vertex set $U \cup W$ where every edge in G joins a vertex from U to a vertex in W. In Figure 2.4, we have the graph G, which shows a maximum matching.

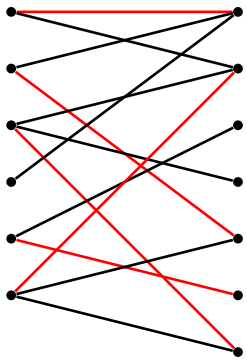

Figure 2.4: A Maximum Matching in a bipartite graph G.

Next, we create a directed graph G' by adding a source node s and sink node t; adding edges from s to every vertex in U and edges from every vertex in W to t (shown in Figure 2.5). Finally, we assign every edge in G' a capacity of 1.

Any matching in G can be transformed into a flow in G'. For each edge uw in G, one unit of flow is pushed along the path $s \to u \to w \to t$. Each path is vertex-disjoint from one another except at s and t. This satisfies the capacity constraint, and the value of the resulting flow is equal to the number of edges in the matching.

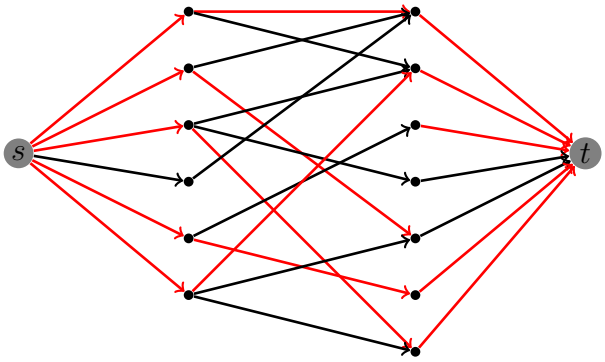

Figure 2.5: The corresponding maximum flow in G'.

In general, any flow in G' can be computed using the Ford-Fulkeron algorithm. Since each edge has a capacity of 1, the flow through it is either zero or one (i.e., every edge in G' is either saturated or avoided). Furthermore, since the flow can be at most 1 from s to

u or w to t, then the saturated edges from U to W form a maximum matching in G which will be exactly equal to the value of the maximum flow in G'.

2.4 Assignment Problem

Finding matchings with the maximum number of edges is a special case of *assignment problems*. An unweighted binary assignment problem includes two disjoint vertex sets A and B, which represent two different types of resources such as jobs and machines, hospitals and interns, columns and rows of a matrix, or customers and drink choices. Our goal is to find the maximum number of pairs (a, b) as possible, where $a \in A$ and $b \in B$, such that:

- Each element in A can appear in at most $c(a)$ pairs.
- Each element in B can appear in at most $c(b)$ pairs.
- Each pair $(a, b) \in A \times B$ can appear in at most $c(a, b)$.

Essentially, we create a pair in our output by assigning each element in A an element in B. The maximum-matching problem is a special case of a maximum flow problem, where the capacity of each vertex in $A \cup B$ is one, and the capacity of each of the pairs (a, b) is 0 if the pair does not define an edge in the graph or 1 if it does.

For example, suppose a school decides to organize a speed dating event. Every pair of students (boy and girl) that wants to participate must sign up for it in advance. The speed dating committee limits each pair to at most two dates together and limits individual students to 5 dates each overall. The committee's goal is to maximize the number of dates. This is a binary assignment problem for the set A of girls and B of boys, such that for each girl a and boy b, $c(a)$=$c(b)$=5. Furthermore, the capacity of the pairs (a, b)=2 (if a and b signed up) or 0 if the students did not register.

All binary assignment problems can be reduced to a maximum flow problem. First, let $G = (V, E)$ with vertices $A \cup B \cup (s, t)$ and the following edges:

- An edge from $s \to a$ with integer capacity $c(a)$ for every $a \in A$.
- An edge from $b \to t$ with integer capacity $c(b)$ for every $b \in B$.
- An edge from $a \to b$ with integer capacity $c(a, b)$ for every $a \in A$ and $b \in B$.

Because all edges have integer capacities, we can use the Ford-Fulkerson algorithm to find an integer maximum flow along s to t. This can be decomposed into the sum of the flows from each path of the form $s \to a \to b \to t$ for some $a \in A$ and $b \in B$. For each such path, we report the pair (a, b) and the corresponding value of the flow.

2.5 Project Selection

Finally, we look at a set of n projects that can be performed, such that $i = 1, 2, ..., n$ is an integer used to identify each project. In this particular example, certain projects can not be done until a project before it has been completed. This type of network can be illustrated using a *directed acyclic graph*: a directed graph with no directed cycles. For instance, a directed edge $i \to j$ means that project i depends on project j. Along with this, each project has an associated profit p_i that is given upon completion of each project. However, there are some projects that have associated negative profit, which we will refer to as positive costs. We can choose any subset X of the projects that include all projects that depend on each other. Our goal is to find a subset of projects that will return the largest profit possible. Consequently, if all jobs produce a negative profit, we do nothing.

Our job is to partition the projects into two subsets S and T, where S represents the projects we select and T represents the projects we do not select. Thus, we would like to model our problem as a minimum cut problem, and use this to find the best way to maximize profit.

We start with letting G be a graph with a source node s and a sink node t. For every profitable project j, we add an edge from $s \to j$, and for every project i with positive costs,

we add an edge from $i \to t$. We can think of s as a new job with profit/cost being 0 that we must perform last. We assign edge capacities as follows:

- $c(s \to j) = p_j$ for every profitable job j;
- $c(i \to t) = -p_j$ for every costly job i;
- $c(i \to j) = \infty$ for every edge $i \to j$.

Since all edge capacities are positive, it is legal to make edge capacities equal to ∞. Now consider an (S, T)-cut in G. If the capacity of the cut, denoted $||S, T||$ is finite, then projects i and j are on the same side of the cut, which means that the set of projects we selected, S, is a feasible solution. Furthermore, we claim that the jobs we selected to complete will earn us a total profit of $C - ||S, T||$, where C is the sum of all positive profits. So this implies that by finding the minimum cut, we will maximize our total profit.

For any subset A of projects, we define three functions:

- $cost(A) := \sum_{i \in A: p_i < 0} -p_i = \sum_{i \in A} c(i \to t)$
- $benefit(A) := \sum_{j \in A: p_i > 0} -p_j = \sum_{j \in A} c(s \to j)$
- $profit(A) := \sum_{i \in A} p_j = benefit(A) - cost(A)$.

In general, $C = benefit(S) + benefit(T)$, meaning the sum of all positive profits is equal to the sum of the profitable jobs of S and profitable jobs of T. Recall that the cut (S, T) can only have finite capacity. So only edges from $s \to j$ and $i \to t$ can cross the cut. Since we constructed our graph so that every edge from $s \to j$ represents a profitable job and from $i \to t$ represents a costly job, we know that the $||S, T|| = cost(S) + benefit(T)$.

Hence,

$$\begin{aligned} C - ||S,T|| &= (benefit(S) + benefit(T)) - (cost(S) + benefit(T)) \\ &= benefit(S) - cost(S) \\ &= profit(S), \end{aligned}$$

as we claimed.

2.6 Practical applications in finance

Before we end this section, we briefly discuss how the aforementioned applications can be applied to practical problems. In particular, how we may use them to deal with situations related to financial flows.

- A collection of edge disjoint paths in a financial flow network is equivalent to a number of different venues from a designated entity to another, such that no other entities are used more than once.
- Vertex capacities in a financial flow network are equivalent to limiting the handle competence of an individual financial entity.
- A maximum matching in a bipartite graph corresponds to finding the maximum number of ways to allocate money without sharing funds with any other financial institution.
- An assignment problem in a financial flow network corresponds to, for instance, banks offering loans to couples or individuals and wanting to maximize the amount of loans given.

CHAPTER 3

FINANCIAL FLOW NETWORKS

In the final chapter, we discuss more direct applications of flow networks to financial systems. We begin by introducing basic terminologies in finance that will be used throughout the chapter. Next, we construct a basic financial system using flow networks to show how funds will be transferred. We will use this same model to discuss possible financial contagion, in which case the direction of the flows are reversed for convenience. Last but not least, we present a novel construction of flow networks that analyzes financial systems over time.

3.1 Basic Finance Terminologies

We begin with basic finance terminologies that will be used throughout this chapter.

Definition 4. *A "financial intermediary" is an entity that acts as a middleman between two parties in a financial transaction, such as a commercial bank, investment banks, mutual funds, and pension funds.*

Through the process of financial intermediation, certain assets or liabilities are turned into different assets or liabilities. From this, financial intermediaries channel funds from people with excess capital (providers) to those who do not have enough money (borrowers) to carry out desired activities. For example, banks connect borrowers with lenders by providing capital from other financial institutions. Insurance companies collect premiums for policies and provide policy benefits. A pension fund collects funds for the members and then distributes payments to the people receiving the pension.

As mentioned before, financial intermediaries turn assets or liabilities into different assets and liabilities. This provides liquidity and maturity transformations.

Definition 5. *"Assets" are resources held by a person or company with the expectation that*

it will provide an economic benefit in the future. Debt is the amount of money borrowed by one party from another.

The most common forms of assets are cash, savings accounts, shares, bonds, land, loans, etc. The most common forms of debt are loans, which is what we will focus on the most in this section. Under the terms of a loan, a borrower is required to repay the balance of the loan by a certain date. Usually the loan will accrue interest over these years, which become assets for the lenders.

Definition 6. *A "liability" is a debt owed by a company to a supplier, bank, lender, or other provider of good and services.*

To settle a liability, a business must sell or hand over an economic benefit. This may include cash, other company assets, or the fulfillment of a service.

Definition 7. *"Maturity transformations" is the process of obtaining short term funds to invest in long term assets. Liquidity transformations are similar to maturity transformations, except they use cash-like liabilities, such a deposits, to buy assets such as loans.*

3.2 Introduction to Financial Flow Networks

Financial systems intermediate the supply and demands of funds expressed by providers and users of the funds. The providers of the funds could be the government, manufacturing companies, households' mortgages, and banks; the users of the funds are borrowers-students, home buyers, business, etc. Flow networks are a natural modeling tool for the representation of these types of financial systems. For our purposes, we represent the source nodes as the providers of the funds and the sink nodes as the users of the funds. In between, there are financial intermediaries: the nodes in the network that are neither sink nor source nodes, which intermediate between providers and users of funds by providing liquidity and maturity transformations.

A financial system is composed of a set of operators $\Omega = \{\omega_i | i = 1, ..., n\}$, such as banks and other financial intermediaries which are directly or indirectly connected to one another by some financial obligation. Let d_{ij} be the value of the liability issued by agent i and held by agent j. Then the balance sheet of a member Ω is $a_i + c_i = e_i + d_i + h_i$ where:

(i) $a_i \in R^+$ is the value of the *external assets* held by agent *i*

(ii) $c_i = \sum_j d_{ji}$ is the sum of the *internal assets* held by agent *i*

(iii) $d_i = \sum_j d_{ij}$ is the sum of the *internal debt* of agent *i*

(iv) h_i is the external debt of agent *i*

(v) e_i value of the equity of agent *i*

From this, we can say that the assets are equal to the debt plus the equity of an agent i, or from the budget identity, equity is equal to your total amount of assets minus debt of an agent i.

A financial flow network is a multi-source flow network $N = \{\Omega, L, A, T, H, \Gamma\}$ such that N is a directed, weighted and connected graph, with two sources and some sink nodes, where:

1. $\Omega = \{\omega_i\}$ is a set of n nodes that represent the financial intermediaries.
2. $A = \{a^k\}$ is the sink node that represents the external assets held by members in Ω.
3. H is a source node that represents households who hold debt claims (deposits and bonds) against agents in Ω.
4. T is another source node that represents the shareholders who own equity of the agents in Ω.

5. L is a the set of directed edges from the source nodes to the intermediary nodes, and from the intermediary nodes to the sink nodes.

6. $\Gamma : L \to R^+$ is a map, called *capacity function*, associates each edge in the network with its appropriate value.

In Figure 3.1, we have visual representation of the financial network described above with an added initial source node s and sink node t.

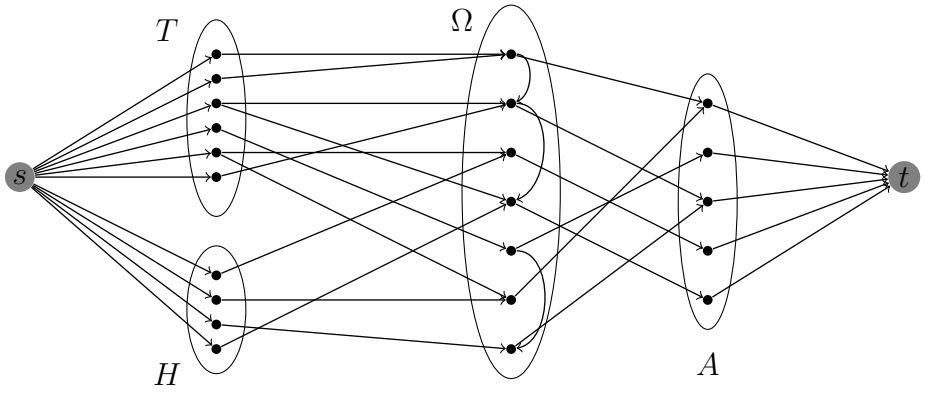

Figure 3.1: Example of a Financial Flow Network

3.3 Direct Financial Contagion

Sometimes in financial networks, financial contagion (also known as the *domino effect* or *systemic risk*) occurs. Financial contagion is the flow of losses across a network from borrowers to lenders due to outside shock that causes the initial default of some agents. In this section, we will analyze the possible effects of direct financial contagion using a model similar to the financial flow network we created previously. For details of the discussion here one can see [2].

Since we are analyzing the transmission of losses from the borrowers to the lenders, it makes sense to reverse the flow of our original flow network. The financial system will still

be composed of a set of operators Ω, but A will now be a set of source nodes that represent the external assets held by the members of Ω. H and T are sinks nodes representing the shareholders and households who own equity or debt claims of agents in Ω. We represent such a network, N', in Figure 3.2.

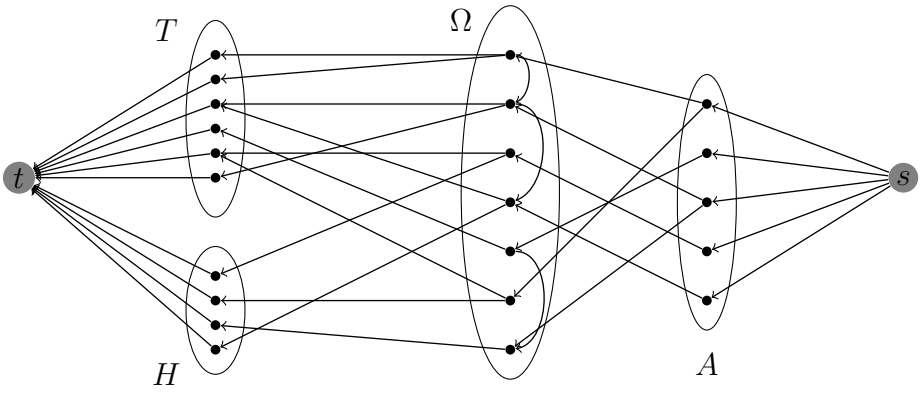

Figure 3.2: Another example of a Financial Flow Network.

A transmission of losses crosses N' when it is upset by a loss of value of some assets in A. The direct financial contagion in a network N' is a diffusion process governed by laws of limited liability, debt priority, and pro-rata reimbursements of creditors. In other words, these laws are put into place to help diffuse the effect contagion may have on the lenders of the network. These laws are described in the model by the following *absorption* and *loss-given default* functions.

The *absorption function* represents the loss that some nodes in Ω suffer from when shock occurs. This loss is first offset by the equity of the nodes and endured by the shareholders. This function measures the share of net worth lost by a node in Ω

$$\beta_i(\lambda_i) = min(\frac{\lambda_i}{e_i}, 1) \tag{3.1}$$

where λ_i is the total loss endured by the i-th node in Ω.

If a node ω_i experiences a positive flow of losses, it sends to its shareholders (T) a fraction of its remaining equity equal to $\beta_i e_i$ where $\beta_i \in [0, 1]$. If $\lambda_i > e_i$, then this node is

insolvent, and the losses that were not absorbed by this node's equity is sent to its creditors. For each node in Ω, let

$$b_i(\lambda_i) = \begin{cases} 0 & \text{if } \lambda_i < e_i \\ \frac{\lambda_i - e_i}{d_i + h_i} & \text{if } \lambda_i \geq e_i \end{cases}$$

be the *loss-given default function*. If a node of Ω is solvent, then b_i is null. If a member defaults, then $b_i \in (0, 1]$. In this case, the assets of ω_i are liquidated and its creditors get a *pro-rata* refund. This a refund in which the shareholders receive a percentage of the member's final net worth. For instance, if a member holds 200 outstanding shares, valued at two dollars per share, the total amount of shares they must pay to their share holder is 400 dollars. Thus, if share holder own 75 shares, the amount they would receive is 75/200 x 400 = 150 dollars. Households receive a loss equal to $b_i h_i$, while creditors receive a loss of $b_i d_{ij}$.

The loss endured by a financial intermediary in Ω is the sum of its internal and external assets. That is

$$\lambda_i = \sum_{k \in A} b_k a_i^k + \sum_{j \in P(\omega_i)} b_j d_{ji} \tag{3.2}$$

where b_k is the portion of the k-th asset that was lost in the initial shock that occurred. The term $P(\omega_i) = \{\omega_j | \exists l_{ji} \in L\}$ here is known as the *parent node*. As shock occurs in a network N', the absorption and loss-given default functions assign a positive real value to the edges in L, known as the *propagation function*.

Later, we will look at two properties of flow networks that are useful in analyzing the vulnerability of different financial networks: the modularity of flow propagations and the constancy of a flow across all cuts of a network.

3.3.1 Financial Contagion Example

In 2008, the United States of America suffered a housing market crash that effected over half of the U.S. When the stock market crashed in the early 2000's, investors looked for

low-risk high return markets to invest their money in. They turned to the housing market. Investors believed that this was a solid investment, since it seemed highly unlikely that the housing market would crash. We will use this crisis to demonstrate a real-world example of financial contagion.

Let s and t be the source and sink nodes of a financial network H. In this network we have a set of nodes, B, that represent the banks or creditors. In the housing business, banks lend money to people who want to buy a house who do not have the money to buy immediately. We call these people *borrowers*. They will be represented by the set of nodes A. The money that is borrowed is called a *mortgage*. A mortgage is a legal agreement in which banks or creditors lend money to help borrowers finance a home. Under this agreement, the borrower must pay the money back within a certain amount of time along with interest. If the borrowers cannot pay the mortgage back in a timely fashion, then the borrower defaults on their loan, and the bank gains ownership of their property.

In order for banks to lend money, they must use short term debts to finance long term assets. This is called a liquidity transformation. In this case, the short term debt used by the banks is the money given to them by investors, which will be represented by a set of nodes I in our network. We represent such a network in Figure 3.3. The links from the nodes in I to the nodes in B represent shareholders who have invested in the mortgages issued by banks and creditors. The links from the nodes in B to the nodes in A represent the loans given to the borrowers.

Often times in the housing market, banks acquire hundreds of mortgages and sell them back to investors. During this era, it seemed as though the housing market was a lucrative, low-risk investment. If a borrower defaulted on their loan, the bank could easily take the title of the house away from the current owners, and put the house back on the market to be bought again. This logic led to an increase in demand from investors of shares, thus resulting in lenders having to create more shares for investors to buy. In order for lenders to

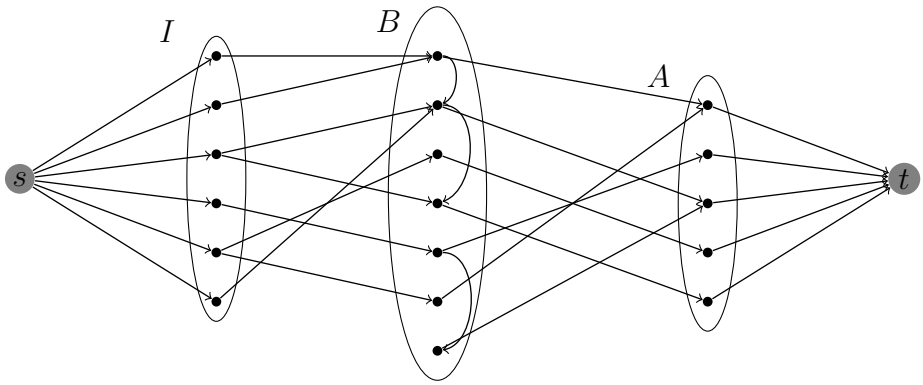

Figure 3.3: Housing Market Financial Network

create more loans, they had to lower their standards on the qualifications of being approved for said loan. For instance, borrowers with poor credit and low-income were being approved for loans. We show this in Figure 3.4, where the red links from B to A represent risky loans that were given to borrowers by banks and creditors.

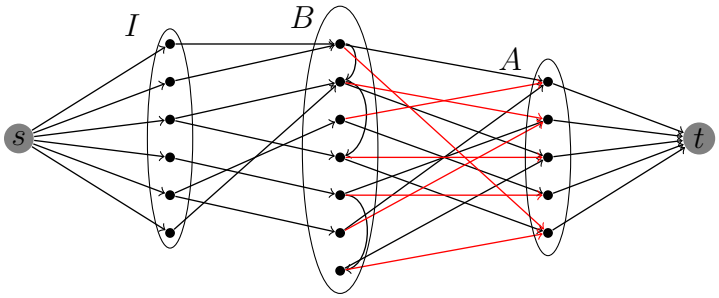

Figure 3.4: Housing Market Financial Network with risky loans.

As a result, housing prices began to rise because of the lax lending requirements and low interest rates. Consequently, borrowers were unable to pay mortgages, which caused them to default on their loans, leaving the banks and creditors with expensive houses that no one could afford. Now banks were unable to pay back the debts they owed to investors and shareholders, causing a transmission of losses from the borrowers to lenders to shareholders/ investors. Hence, financial contagion occurred. The debt of banks outweighed their assets, and many financial institutions were forced to declare bankruptcy.

3.3.2 Modularity

Essentially modularity claims that one may "cluster" a certain subset of nodes and links together and consider the induced subgraph as a single node. The links going into and coming out of this node are from the set of links that go into or come out of the original subgraph.

The advantage of establishing such clusters, or "strongly connected components" as termed in [2], is so that one may consider a simplified network with fewer nodes and links (where algorithms run faster exponentially). After identifying the crucial nodes or links, one may reconsider the corresponding original subgraphs and reexamine the flow network induced by them. In [2], modularity is used to cluster defaulting agents and analyze their behavior in isolation. By doing this, we are able to conclude they cannot determine the losses of that network if the agents in the cluster do not have debt towards each other. In other words, if there is no debt between agents in the cluster, then the losses that the network as a whole experiences cannot be determined.

3.3.3 Flow Constancy Across Cuts

As discussed in [2], "flow constancy across cuts" follows from the simple mathematical fact that, no matter what "cut" (partition of the nodes into two sets with the source in one part and the sink in the other) one considers in the flow network the amount of flow going from the source side to the sink side is a constant.

This means, in order to find the threshold of the network flow or the set of links whose capacities prevents the flow to become larger, one may consider any cut in the network and simply find the one with the minimum sum of capacities from the source side to the sink side. This is coherent with the max flow-min cut theorem. According to [2], these properties characterize the first and final thresholds of contagion. The first shows the value of the

smallest shock capable of causing financial contagion, and the final shows the smallest shock capable of causing all agents in a network to default.

3.4 Financial Flow Networks Over Time

One of the main drawbacks of the flow network model is the lack of consideration of the change in information over time. In this section we provide an easy but interesting way of dealing with this issue. In Figure 3.5, each Ω_i represents a particular time period, say a year. The links with in each Ω_i provides us the corresponding information of financial transactions between the financial intermediaries within that year. The transfer of funds from year to year will be represented by the links between Ω_i and Ω_{i+1} for $i = 1, 2, \ldots, n-1$.

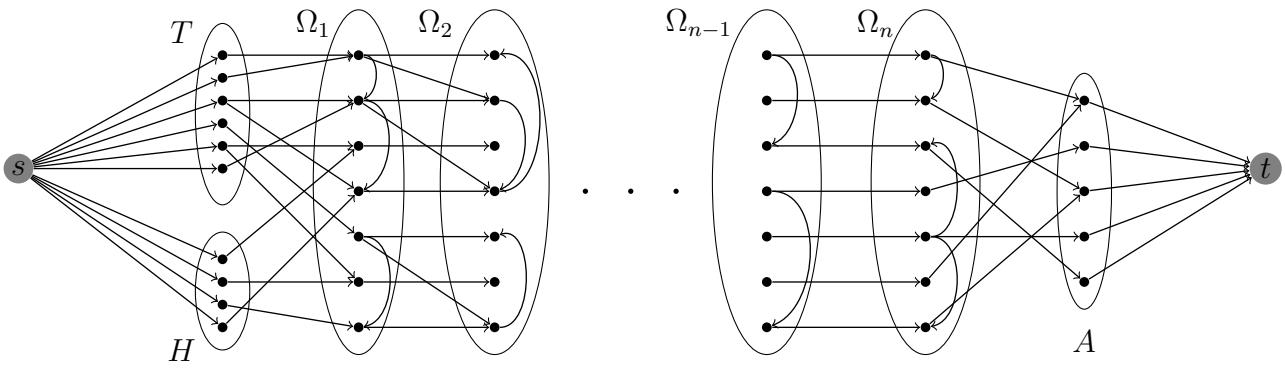

Figure 3.5: Flow network over a given time period

Next we further discuss some of the specific applications, presented in Chapter 2, in this model.

3.4.1 Edge disjoint paths

In the context of Figure 3.5, a collection of edge disjoint paths from the source to the sink represents a set of means to transfer fund from the source to the sink (or, in reality, a way for a collection of financial flows to happen during the course of n years) such that no two

paths share a common edge. In other words, how money was transferred over the years such that no transactions interact with each other.

For example, consider a marriage that is ending in divorce. The husband and the wife would represent the source nodes, H and W. Ω is still represented as the financial intermediaries, and the sink node would represent the total amount of assets the couple has acquired together over the years. This is shown in Figure 3.6. We could use edge-disjoint paths to

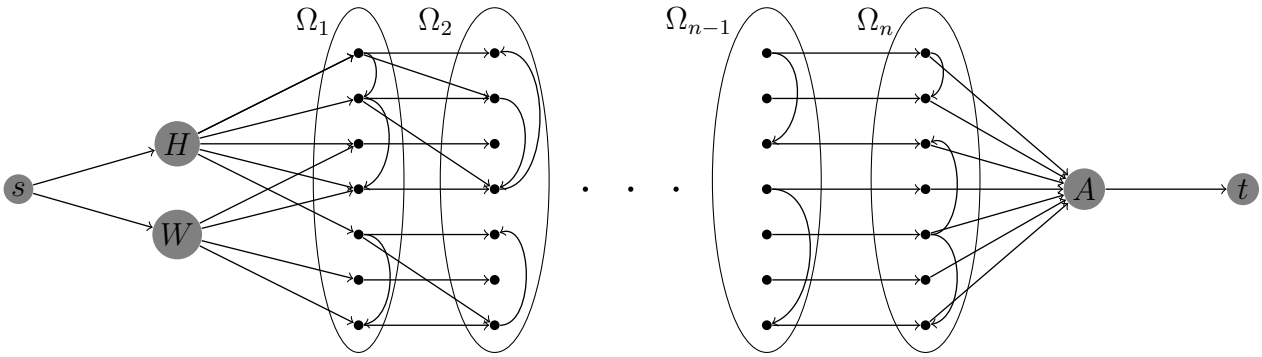

Figure 3.6: A Husband and Wife's Financial Network over time.

trace the total amount of money the wife and the husband invested separately to determine how the assets should be divided in the divorce.

3.4.2 Assignment problem

The assignment problem, when applied to the bipartite graph formed by Ω_i and Ω_{i+1}, limits the number of input or output links to nodes in Ω_{i+1} and from nodes in Ω_i. This corresponds to the constraints that each financial institution is only allowed to be involved in a limited number of transactions between years.

For example, in many cases of savings accounts, IRA's, and other forms of savings, banks/creditors often limit the account holder to the number of times they are able to withdraw from their account. This is because banks often use these deposits to finance their

own assets. If an account holder uses too much of the money in their account, then banks are unable to fulfill their obligations i.e. loan disbursement.

REFERENCES

[1] T. Cormen, C. Leiserson, "Graph Algorithms." *Introduction to Algorithms*, 3rd Edition. Place of Publication Not Identified: n.p., 2009. 709-32. Print.

[2] M. Eboli, Financial applications of Flow Network Theory, *Adv. Dyanamic Modeling of Economic and Social Systems*, SCI 448, 21–29.

[3] D. Guichard, *An introduction to Combinatorics and Graph Theory*, $https://www.whitman.edu/mathematics/cgt\_online/cgt.pdf$