4-2016

Optimal monitoring and mitigation of systemic risk in lending networks

Zhang Li
Purdue University

Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Zhang Li

Entitled

OPTIMAL MONITORING AND MITIGATION OF SYSTEMIC RISK IN LENDING NETWORKS

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Ilya Pollak Co-chair	Ben Craig
Borja M. Peleato-Inarrea Co-chair	
Saul B. Gelfand	
Xiaojun Lin	

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Ilya Pollak, Borja M. Peleato-Inarrea

Approved by: Venkataramanan Balakrishnan 4/26/2016

Head of the Departmental Graduate Program Date

OPTIMAL MONITORING AND MITIGATION OF SYSTEMIC RISK IN LENDING NETWORKS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Zhang Li

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2016

Purdue University

West Lafayette, Indiana

ACKNOWLEDGMENTS

First of all, I would like to express the deepest gratitude to my major advisors, Prof. Ilya Pollak and Prof. Borja Peleato, not only for their help on my research, but also for their encouragements and guidance during my Ph.D program. During the past four years, I was sometimes doubted and depressed. It is Prof. Pollak and Prof. Peleato that gave me confidence and faith to go through those difficult periods. I will never forget the two-hour conversation with Prof. Pollak at the Starbucks near campus on June 27, 2014, when I almost gave up pursuing my Ph.D degree. I will never forget my first presentation to Prof. Peleato on July 25, 2014. They brought me back to the correct track. Without them, this thesis would never be possible.

I would like to thank other members in my committee, Prof. Xiaojun Lin, Prof. Ben Craig and Prof. Saul Gelfand, for their precious time and effort on my research. Prof. Lin gave me guidance in the first two years on networking. Prof. Craig helped me cover the gap between my model and the real financial world. Prof. Gelfand encouraged me to learn machine learning, which is really useful in job market.

My greatest appreciation is given to my parents. No matter what happens, they are always with me and give me strongest support. I learn perseverance from my father and optimism from my mother.

Special thanks are given to Dr. Xiaoyu Chu from University of Maryland, for her encouragements on my Ph.D program application, my preliminary exam and my final exam. When I was in darkness, her phone call reminded me my dream and lit up the way toward my brightest future.

Last but not least, I would like to thank all my family members and friends who helped me during my four-year Ph.D program.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Li, Zhang PhD, Purdue University, May 2016. Optimal Monitoring and Mitigation of Systemic Risk in Lending Networks. Major Professors: Ilya Pollak and Borja Peleato.

This thesis proposes optimal policies to manage systemic risk in financial networks. Given a one-period borrower-lender network in which all debts are due at the same time and have the same seniority, we address the problem of allocating a fixed amount of cash among the nodes to minimize the weighted sum of unpaid liabilities. Assuming all the loan amounts and cash flows are fixed and that there are no bankruptcy costs, we show that this problem is equivalent to a linear program. We develop a duality-based distributed algorithm to solve it which is useful for applications where it is desirable to avoid centralized data gathering and computation. Since some applications require forecasting and planning for a wide variety of different contingencies, we introduce a stochastic model for the institutions operating cash and consider the problem of minimizing the expectation of the weighted sum of unpaid liabilities. We show that this problem is a two-stage stochastic linear program and develop an online learning algorithm based on stochastic gradient descent to solve it. We consider a number of further extensions of our deterministic scenario by incorporating various additional features of real-world lending networks into our model. For example, we show that if the defaulting nodes do not pay anything, then the optimal cash injection allocation problem is a mixed-integer linear program. In addition, we develop and evaluate two heuristic algorithms to allocate the cash injection amount so as to minimize the number of nodes in default. Our results provide algorithmic tools to help financial institutions, banking supervisory authorities, regulatory agencies, and clearing houses in monitoring and mitigating systemic risk in financial networks.

1. OPTIMAL MONITORING AND MITIGATION OF SYSTEMIC RISK IN FINANCIAL NETWORKS UNDER THE DETERMINISTIC MODEL

1.1 Introduction

The events of the last several years revealed an acute need for tools to systematically model, analyze, monitor, and control large financial networks. Motivated by this need, we propose to address the problem of optimizing the amount and structure of liquidity assistance in a distressed financial network, under a variety of modeling assumptions and implementation scenarios.

Two broad applications motivate our work: day-to-day monitoring of financial systems and decision making during an imminent crisis. Examples of the latter include the decision in September 1998 by a group of financial institutions to rescue Long-Term Capital Management, and the decisions by the Treasury and the Fed in September 2008 to rescue AIG and to let Lehman Brothers fail. The deliberations leading to these and other similar actions have been extensively covered in the press. These reports suggest that the decision making processes could have benefited from quantitative methods for analyzing potential policies and their likely outcomes. In addition, such methods could help avoid systemic crises in the first place, by informing day-to-day actions of financial institutions, regulators, supervisory authorities, and legislative bodies.

Given a financial network model, we are interested in addressing the following problem.

> **Problem I:** Allocate a fixed amount of cash assistance among the nodes in a financial network in order to minimize the (possibly weighted) sum of unpaid liabilities in the system.

An alternative formulation of the same problem, is to both select the amount of injected cash and determine how to distribute it among the nodes in order to minimize the overall cost equal to a linear combination of the weighted sum of unpaid liabilities and the amount of injected cash.

We consider a static model with a single maturity date, and with a known network structure. We assume that we know both the amounts owed by every node in the network to every other node, and the net asset amounts available to every node from sources external to the network. Even for this relatively simple model, Problem I is far from straightforward, because of a nonlinear relationship between the cash injection amounts and the loan repayment amounts. Building upon the results from [1], we construct algorithms for computing exact solutions for Problem I and its alternative variant, by showing in Section 1.3 that both formulations are equivalent to linear programs under a proportional payment scheme, such as the one assumed in [1].

We consider a number of extensions of our model by adding to it various features that characterize real-world lending networks. In Section 1.4, we allow the obligations in the network to have multiple seniorities, so that a node may only satisfy a liability once it fully repays all of its more senior liabilities. Within each seniority, we still assume the same proportional payment scheme as in [1]. We show that in this case, Problem I is an NP-hard mixed-integer linear program. However, we show through simulations that use optimization package CVX [2, 3] that this problem can be accurately solved in a few seconds on a personal computer for a network size comparable to the size of the US banking network.

In Section 1.5, we incorporate credit default swaps (CDSs) into our model: any node in the network can now sell a CDS to any other node that insures the latter against the default of one of its borrowers. In this case, we show that simultaneous bilateral clearing assumed in [1] does not necessarily guarantee the existence of a solution even for very simple networks with loops. We instead adopt a three-round clearing scheme: first, the payments on the underlying obligations are cleared; the second round consists of the payments from the CDSs triggered by the first-round

defaults; and in the third round additional payments can be made on the underlying obligations. We show that under this scheme, Problem I is also a mixed-integer linear program that can be efficiently solved for networks of relevant sizes.

We show in Section 1.8 that under the all-or-nothing payment scheme where the defaulting nodes do not pay at all, Problem I is also a mixed-integer linear program which can be accurately and efficiently solved.

We also consider another problem where the objective is to minimize the number of defaulting nodes rather than the weighted sum of unpaid liabilities:

> **Problem II:** Allocate a fixed amount of cash assistance among the nodes in a financial network in order to minimize the number of nodes in default.

For Problem II, we develop two heuristic algorithms in Section 1.6: a reweighted ℓ_1 minimization approach inspired by [4] and a greedy algorithm. We illustrate our algorithms using examples with synthetic data for which the optimal solution can be calculated exactly. We show through numerical simulations that the solutions calculated by the reweighted ℓ_1 algorithm are close to optimal, and that the performance of the greedy algorithm highly depends on the network topology. We also compare these two algorithms using three types of random networks for which the optimal solution is not available. In one of these three examples the performance of these two algorithms is statistically indistinguishable; in the second example the greedy algorithm outperforms reweighted ℓ_1 minimization; and in the third example the reweighted ℓ_1 minimization algorithm outperforms the greedy approach.

While Problem II is unlikely to be of direct practical importance (indeed, it is difficult to imagine a situation where a regulator would consider the failures of a small local bank and Citi to be equally bad), it serves as a stepping stone to a more practical and more difficult scenario where the optimization objective is a linear combination of the weighted unpaid liabilities (as in Problem I) and the sum of weights over the defaulted nodes (an extension of Problem II).

Problem III: Given a fixed amount of cash to be injected into the system, we consider an objective function which is a linear combination of the sum of weights over the defaulted nodes and the weighted sum of unpaid liabilities.

We show in Section 1.7 that this problem is equivalent to a mixed-integer linear program.

1.1.1 Related Literature

Contagion in financial networks has been frequently studied in the past, especially after the financial crisis in 2007-2008. Notable examples of network topology analysis based on real data are [5–8]. Real data informs the new approaches for assessing systemic financial stability of banking systems developed in [9–22].

Often, systemic failures are caused by an epidemic of defaults whereby a group of nodes unable to meet their obligations trigger the insolvency of their lenders, leading to the defaults of lenders' lenders, etc, until this spread of defaults infects a large part of the system. For this reason, many studies have been devoted to discovering network structures conducive to default contagion [23–29]. The relationships between the probability of a systemic failure and the average connectivity in the network are investigated in [23, 26, 29]. Other features, such as as the distribution of degrees and the structure of the subgraphs of contagious links, are examined in [27].

While potentially useful in policymaking, most of these references do not provide specific policy recipes. One strand of literature on quantitative models for optimizing policy decisions has focused on analyzing the efficacy of bailouts and understanding the behavior of firms in response to bailouts. To this end, game-theoretic models are proposed in [30] and [31] that have two agents: the government and a single private sector entity. The focus of another set of research efforts has been on the setting of capital and liquidity requirements [24, 32–34] in order to reduce systemic risk.

Our work contributes to the literature by taking a network-level view of optimal policies and proposing optimal cash injection strategies for networks in distress. This

chapter extends our earlier work reported in [35–37]. In addition to ours, several other papers have recently considered cash injection policies for lending networks [38–44], all based on the framework proposed in [1].

Under the proportional payment mechanism presented in [1], the problem of determining the clearing vector is formulated as a linear program in [38]. Two systemic risk measures are obtained by considering the associated dual problem of the linear program: *Contagion Risk Indicator* and *Funding Risk Indicator*. Furthermore [38] develops optimal bailout strategies for two objectives: minimizing the total amount of cash injection given a constraint on the weighted number of defaults and minimizing the weighted number of defaults given a budget of cash injection. It is shown in [38] that both these problems are mixed-integer linear programs. Our work on proportional payment scheme is also based on the linear program formulation extended from [1]. Our objective in Problem I is minimizing the weighted sum of unpaid liabilities, which is different from any objectives in [38]. Our problem II also aims to minimize the number of defaults. But instead of formulating it as a MILP, which is hard to solve for large networks, we propose two scalable heuristic algorithms. Beside the proportional payment scheme in [38], we also consider problem II with the all-or-nothing payment scheme. We adapt the model with multiple seniorities in [38] to our Problem I. For the model with CDS, there exists an issue in [38]: the one-period simultaneous clearing model in [1] cannot be simply extended to the one with CDS since simultaneous bilateral clearing does not necessarily guarantee the existence of a solution even for very simple networks with loops. In our work, we circumvent this issue with a three-round clearing scheme.

A cash injection targeting policy is developed in [39–41] for an infinitesimally small amount of injected cash. The basic idea of the policy is to inject the cash into the node with the largest "threat index" which is the same as the funding risk indicator from [38]. This targeting policy is optimal when the amount of the injected cash is small enough to keep the set of defaulting nodes unchanged. However, as pointed out in [41], the targeting policy is not monotone in the cash injection amount, and

therefore this algorithm cannot be easily extended to non-infinitesimal cash injection amounts.

In [42, 43], bankruptcy costs are incorporated into the model of [1]. The main contribution of that work is showing that because of the bankruptcy costs, it is sometimes beneficial for some solvent banks to form bailout consortia and rescue failing banks. However, it may happen that the solvent banks do not have enough means to effect a bailout, and in this case external intervention may still be needed.

A multi-period stochastic clearing framework based on [1] is proposed in [44], where a lender of last resort monitors the network and may provide liquidity assistance loans to failing nodes. The paper proposes several strategies that the lender of last resort might follow in making its decisions. One of these strategies, the so-called max-liquidity policy, aims to solve our Problem I during each period. However, [44] does not describe an algorithm for solving this problem.

Another related work is [45]. Based on the clearing payment framework in [1], the authors of [45] study the probability of contagion and amplification of losses due to network effects when the system suffers a random shock.

1.1.2 Outline of this chapter

This chapter is organized as follows. Section 1.2 describes the model of financial networks, the clearing payment mechanism, and the notation. Section 1.3 shows that if each defaulting node pays its creditors in proportion to the owed amounts, then Problem I and its alternative formulation are equivalent to linear programs. Section 1.4 investigates the model with multiple seniorities and Section 1.5 incorporates CDS in the model. Two heuristic algorithms are developed in Section 1.6 to solve Problem II under the proportional payment mechanism: a reweighted ℓ_1 minimization algorithm and a greedy algorithm. Problem III is considered in Section 1.7. Section 1.8 analyzes Problem I under the assumption that the defaulting nodes do not pay anything. We prove that it is then an NP-hard mixed-integer linear pro-

Table 1.1.
Notation for several vector quantities.

VECTOR	i-TH COMPONENT
$\mathbf{0}$	0
$\mathbf{1}$	1
$\mathbf{e} \geq \mathbf{0}$	net external assets at node i before cash injection
$\mathbf{c} \geq \mathbf{0}$	external cash injection to node i
$\bar{\mathbf{p}}$	the amount node i owes to all its creditors
$\mathbf{p} \leq \bar{\mathbf{p}}$	the total amount node i actually repays all its creditors on the due date of the loans
$\bar{\mathbf{p}} - \mathbf{p}$	node i's total unpaid liabilities
$\mathbf{r}$	remaining cash of node i after clearing payment
$\mathbf{w}$	the weight of \$1 of unpaid liability at node i
$\mathbf{v}$	the weight of node i's default
$\mathbf{d}$	indicator variable of whether node i defaults, i.e., $d_i = 1$ if node i defaults; $d_i = 0$ otherwise

gram and show that can be efficiently solved using modern optimization software for network sizes comparable to the size of the US banking system.

1.2 Model and Notation

Our network model is a directed graph with N nodes where a directed edge from node i to node j with weight $L_{ij} > 0$ signifies that i owes $\$L_{ij}$ to j. This is a one-period model with no dynamics—i.e., we assume that all the loans are due on the same date and all the payments occur on that date. We use the following notation:

- any inequality whose both sides are vectors is component-wise;

- $\mathbf{0}$, $\mathbf{1}$, $\mathbf{e}$, $\mathbf{c}$, $\bar{\mathbf{p}}$, $\mathbf{p}$, $\mathbf{r}$, $\mathbf{w}$, $\mathbf{v}$, and $\mathbf{d}$ are all vectors in $\mathbb{R}^N$ defined in Table 1.1;
- $W = \mathbf{w}^T(\bar{\mathbf{p}} - \mathbf{p})$ is the weighted sum of unpaid liabilities in the system;
- N_d is the number of nodes in default, i.e., the number of nodes i whose payments are below their liabilities, $p_i < \bar{p}_i$;
- Π_{ij} is what node i owes to node j, as a fraction of the total amount owed by node i,
$$\Pi_{ij} = \begin{cases} \frac{L_{ij}}{\bar{p}_i} & \text{if } \bar{p}_i \neq 0, \\ 0 & \text{otherwise;} \end{cases}$$
- Π and L are the matrices whose entries are Π_{ij} and L_{ij}, respectively.

Given the above financial system, we consider the proportional payment mechanism and the all-or-nothing payment mechanism. The latter can be alternatively interpreted as the proportional payment mechanism with 100% bankruptcy costs. As proposed in [1], the proportional payment mechanism without bankruptcy costs is defined as follows.

Proportional payment mechanism with no bankruptcy costs:

- If i's total funds are at least as large as its liabilities (i.e., $\sum_{j=1}^{N} \Pi_{ji}p_j + e_i + c_i \geq \bar{p}_i$), then all i's creditors get paid in full.
- If i's total funds are smaller than its liabilities, then i pays all its funds to its creditors.
- All i's debts have the same seniority. This means that, if i's liabilities exceed its total funds then each creditor gets paid in proportion to what it is owed. This guarantees that the amount actually received by node j from node i is always $\Pi_{ij}p_i$. Therefore, the total amount received by any node i from all its borrowers is $\sum_{j=1}^{N} \Pi_{ji}p_j$.

Under these assumptions, a node will pay all the available funds proportionally to its creditors, up to the amount of its liabilities. The payment vector can lie anywhere in the rectangle $[\mathbf{0}, \bar{\mathbf{p}}]$. Under the all-or-nothing payment scenario, the defaulting nodes do not pay at all, so each component i of the payment vector is either 0 or $\bar{p}_i$

All-or-nothing payment mechanism:

- If i's total funds are at least as large as its liabilities, then all i's creditors get paid in full.
- If i's total funds are smaller than its liabilities, then i pays nothing.

As defined in [1], a *clearing payment vector* $\mathbf{p}$ is a vector of borrower-to-lender payments that is consistent with the conditions of the payment mechanism.

In this chapter, we are mostly concerned with Problems I and II under the proportional payment scenario with no bankruptcy costs. We also prove that the all-or-nothing payment scenario makes Problem I NP-hard. In this case, Problem I can be formulated as a mixed-integer linear program that can be efficiently solved on a personal computer using modern optimization software for network sizes comparable to the size of the US banking system.

1.3 Optimal Solution

Consider a network with a known structure of liabilities L and a known vector $\mathbf{e}$ of net assets before cash injection. Using the notation established in the preceding section, Problem I seeks a cash injection allocation vector $\mathbf{c} \geq \mathbf{0}$ to minimize the following weighted sum of unpaid liabilities,

$$W = \mathbf{w}^T(\bar{\mathbf{p}} - \mathbf{p}),$$

subject to the constraint that the total amount of cash injection does not exceed some given number C:

$$\mathbf{1}^T\mathbf{c} \leq C.$$

In this section, we assume proportional payments with no bankruptcy costs. We first prove that, for any cash injection vector $\mathbf{c}$, there exists a unique clearing payment vector that minimizes the cost W.

Lemma 1 *Given a financial system* $(\Pi, \bar{\mathbf{p}}, \mathbf{e})$*, a cash injection vector* $\mathbf{c}$ *and a weight vector* $\mathbf{w} > \mathbf{0}$*, there exists a unique clearing payment vector* $\mathbf{p}$ *minimizing the weighted sum* $W = \mathbf{w}^T(\bar{\mathbf{p}} - \mathbf{p})$.

Proof First, note that since $\mathbf{w}$ and $\bar{\mathbf{p}}$ do not depend on $\mathbf{p}$ or $\mathbf{c}$, minimizing W is equivalent to maximizing $\mathbf{w}^T\mathbf{p}$. With a fixed cash injection vector $\mathbf{c}$, the financial system is equivalent to $(\Pi, \bar{\mathbf{p}}, \mathbf{e} + \mathbf{c})$. Since $\mathbf{w} > \mathbf{0}$, we have that $\mathbf{w}^T\mathbf{p}$ is a strictly increasing function of $\mathbf{p}$. By Lemma 4 in [1], the clearing payment vector $\mathbf{p}$ can be obtained by solving the following linear program:

$$\max_{\mathbf{p}} \mathbf{w}^T\mathbf{p} \tag{1.1}$$

subject to

$$\mathbf{0} \leq \mathbf{p} \leq \bar{\mathbf{p}}, \tag{1.2}$$

$$\mathbf{p} \leq \Pi^T\mathbf{p} + \mathbf{e} + \mathbf{c}. \tag{1.3}$$

From Theorem 1 in [1], there exists a greatest clearing payment vector $\mathbf{p}^*$. Since W is a strictly increasing function of $\mathbf{p}$, $\mathbf{p}^*$ is a solution of LP (1.1-1.3). For any other $\mathbf{p} \neq \mathbf{p}^*$, we have $p_i \leq p_i^*$ for $i = 1, 2, \cdots, N$ and at least one of these inequalities is strict. Thus, $\mathbf{w}^T\mathbf{p} < \mathbf{w}^T\mathbf{p}^*$. Therefore $\mathbf{p}^*$ is the unique solution of LP (1.1-1.3). This completes the Proof of Lemma 1. ■

We now establish the equivalence of Problem I and a linear programming problem.

Theorem 1 *Assume that the liabilities matrix* L*, the asset vector* $\mathbf{e}$*, the weight vector* $\mathbf{w}$*, and the total cash injection amount* C *are fixed and known. Assume that the system utilizes the proportional payment mechanism with no bankruptcy costs. Consider Problem I, i.e., the problem of calculating a cash injection allocation* $\mathbf{c} \geq \mathbf{0}$ *to*

minimize the weighted sum of unpaid liabilities $W = \mathbf{w}^T(\bar{\mathbf{p}} - \mathbf{p})$ *subject to the budget constraint* $\mathbf{1}^T\mathbf{c} \leq C$. *A solution to this problem can be obtained by solving the following linear program:*

$$\max_{\mathbf{p},\mathbf{c}} \mathbf{w}^T\mathbf{p} \tag{1.4}$$

subject to

$$\mathbf{1}^T\mathbf{c} \leq C, \tag{1.5}$$

$$\mathbf{c} \geq \mathbf{0}, \tag{1.6}$$

$$\mathbf{0} \leq \mathbf{p} \leq \bar{\mathbf{p}}, \tag{1.7}$$

$$\mathbf{p} \leq \Pi^T\mathbf{p} + \mathbf{e} + \mathbf{c}. \tag{1.8}$$

Proof Since the constraints on $\mathbf{c}$ and $\mathbf{p}$ in LP (1.4-1.8) form a closed and bounded set in $\mathbb{R}^{2N}$, a solution exists. Moreover, for any fixed $\mathbf{c}$, it follows from our Lemma 1 and Lemma 4 in [1] that the linear program has a unique solution for $\mathbf{p}$ which is the clearing payment vector for the system.

Let $(\mathbf{p}^*, \mathbf{c}^*)$ be a solution to (1.4-1.8). Suppose that there exists a cash injection allocation that leads to a smaller cost W than does $\mathbf{c}^*$. In other words, suppose that there exists $\mathbf{c}' > \mathbf{0}$, with $\mathbf{1}^T\mathbf{c}' \leq C$, such that the corresponding clearing payment vector $\mathbf{p}'$ satisfies $\mathbf{w}^T(\bar{\mathbf{p}} - \mathbf{p}') < \mathbf{w}^T(\bar{\mathbf{p}} - \mathbf{p}^*)$, or, equivalently,

$$\mathbf{w}^T\mathbf{p}^* < \mathbf{w}^T\mathbf{p}'. \tag{1.9}$$

Note that $\mathbf{c}'$ satisfies the first two constraints of (1.4-1.8). Moreover, since $\mathbf{p}'$ is the corresponding clearing payment vector, the last two constraints are satisfied as well. The pair $(\mathbf{p}', \mathbf{c}')$ is thus in the constraint set of our linear program. Therefore, Eq. (1.9) contradicts the assumption that $(\mathbf{p}^*, \mathbf{c}^*)$ is a solution to (1.4-1.8). This completes the Proof that $\mathbf{c}^*$ is the allocation of C that achieves the smallest possible cost W. ■

In an alternative formulation of Problem I, we are given a weight λ and must choose the total cash injection amount C and its allocation $\mathbf{c}$ to minimize $\lambda C + W$. This is equivalent to the following linear program:

$$\max_{C,\mathbf{c},\mathbf{p}} \mathbf{w}^T\mathbf{p} - \lambda C \tag{1.10}$$

subject to

$$\mathbf{1}^T\mathbf{c} = C,$$

$$\mathbf{c} \geq \mathbf{0},$$

$$\mathbf{0} \leq \mathbf{p} \leq \bar{\mathbf{p}},$$

$$\mathbf{p} \leq \Pi^T\mathbf{p} + \mathbf{e} + \mathbf{c}.$$

This equivalence follows from Theorem 1: denoting a solution to (1.10) by $(C^*, \mathbf{p}^*, \mathbf{c}^*)$, we see that the pair $(\mathbf{p}^*, \mathbf{c}^*)$ must be a solution to (1.4-1.8) for $C = C^*$. At the same time, the fact that C^* maximizes the objective function in (1.10) means that it minimizes $\lambda C + W = \lambda C + \mathbf{w}^T(\bar{\mathbf{p}} - \mathbf{p})$, since $\bar{\mathbf{p}}$ is a fixed constant.

1.4 Problem I with Multiple Seniorities

1.4.1 Assumptions and Notation

We now extend our model of Section 1.2 to the case of multiple seniorities. We assume that a node may only satisfy a liability once it fully repays all of its more senior liabilities. Within each seniority, we still assume the same proportional payment scheme as in Section 1.3, and we still assume that there are no bankruptcy costs, and that each node either pays all its liabilities in full or pays out all its available funds to its creditors. For all the variables that involve liabilities and payments, we augment our notation of Section II with superscript k to denote the seniority. For example, now L_{ij}^k is the amount that node i owes to node j at seniority k; d_i^k denotes whether node i's liabilities at seniority k are paid in full, etc. Larger numbers denote more junior obligations. This means that a nonzero payment $p_i^k > 0$ can only occur

if all node i's obligations more senior than k are satisfied in full, i.e., if $p_i^h = \bar{p}_i^h$ for all $h < k$. This also means that an incomplete payment $p_i^k < \bar{p}_i^k$ at any seniority k can only occur if i repays nothing for any of its more junior obligations than k, i.e., if $p_i^h = 0$ for all $h > k$. We denote the number of distinct seniorities among node i's obligations by K_i, and we let $K = \max_i K_i$. We denote the most senior obligation of each node by $k = 1$.

As in Section 1.3, we allow the unpaid liabilities in the objective function to have different weights for different nodes. In addition, we allow different weights for different seniorities. Thus, we seek a cash injection allocation vector $\mathbf{c} \geq \mathbf{0}$ to minimize

$$W = \sum_{k=1}^{K} \mathbf{w}^{kT}(\bar{\mathbf{p}}^k - \mathbf{p}^k),$$

subject to $\mathbf{1}^T\mathbf{c} \leq C$. We assume that all the weights are strictly positive: $w_i^k > 0$ for all nodes i and liabilities k.

1.4.2 Optimal Solution to Problem I with Multiple Seniorities

Theorem 2 *Assume that the system utilizes the proportional payment mechanism with multiple seniorities, as defined in Section 1.4.1. Then the optimal cash injection*

vector $\mathbf{c}^*$ *and its corresponding clearing payment vectors* $\mathbf{p}^{k*}$, $k = 1, 2, \ldots, K$, *are found from the following mixed integer linear program:*

$$\max_{\mathbf{p}^k, \mathbf{c}, \mathbf{d}^k} \sum_{k=1}^{K} \mathbf{w}^{kT} \mathbf{p}^k \tag{1.11}$$

subject to

$$\mathbf{1}^T \mathbf{c} \leq C, \tag{1.12}$$

$$\mathbf{c} \geq \mathbf{0}, \tag{1.13}$$

$$\sum_{k=1}^{K} \sum_{j=1}^{N} \Pi_{ji}^k p_j^k + e_i + c_i \geq \sum_{k=1}^{K} p_i^k, \text{ for } i = 1, 2, \ldots, N, \tag{1.14}$$

$$\mathbf{0} \leq \mathbf{p}^k \leq \bar{\mathbf{p}}^k, \text{ for } k = 1, 2, \ldots, K, \tag{1.15}$$

$$(1 - d_i^k)\bar{p}_i^k \leq p_i^k, \text{ for } i = 1, 2, \ldots, N \text{ and } k = 1, 2, \ldots, K, \tag{1.16}$$

$$p_i^{k+1} \leq (1 - d_i^k)\bar{p}_i^{k+1}, \text{ for } i = 1, 2, \ldots, N \text{ and } k = 1, 2, \ldots, K-1, \tag{1.17}$$

$$d_i^k \in \{0, 1\}, \text{ for } i = 1, 2, \ldots, N \text{ and } k = 1, 2, \ldots, K. \tag{1.18}$$

Proof Assume the solution to MILP (1.11) is ($\mathbf{c}^*$, $\mathbf{p}^{k*}$, $\mathbf{d}^{k*}$), $k = 1, 2, \ldots, K$. First, we prove that $\mathbf{p}^{k*}$, $k = 1, 2, \ldots, K$, are clearing payment vectors for cash injection vector $\mathbf{c}^*$. In a clearing payment, node i either fully pays its liabilities or pays all its available funds. In addition, if node i fails to fully pay its liabilities, it must pay off the more senior liabilities in full before it starts to pay off the more junior ones.

If $p_l^{k*} = \bar{p}_l^k$ for $k = 1, 2, \ldots, K$, then node l pays all its liabilities in full, which satisfies the requirements of the clearing payment vectors.

If node l does not pay its liabilities in full, in other words, if there exists a seniority h such that $p_l^{h*} < \bar{p}_l^h$, then we prove that node l repays no liabilities more junior than h and pays all its available funds. If $h = K$ or $\bar{p}_l^{h+1} = 0$, there are no liabilities junior to h for node l. If $h < K$ and $\bar{p}_l^{h+1} > 0$, we have $d_l^{h*} = 1$ due to constraints (1.16) and (1.18) and then as a consequence of constraint (1.17), $p_l^{h+1*} = 0$. Moreover, we have $p_l^{h+1*} < \bar{p}_l^{h+1}$. Thus, by induction, for all $k > h$, we have $p_l^{k*} = 0$, which proves that node l does not repay any liabilities more junior than h.

We furthermore prove that if there is a nonzero payment $p_l^h > 0$ at some seniority h, this means that node l repays in full all its obligations more senior than h. If $h = 1$ then there are no liabilities senior to h for node l. If $h > 1$, then constraints (1.17) and (1.18) imply that $d_l^{h-1} = 0$. But then constraints (1.15) and (1.16) imply that $p_l^{h-1} = \bar{p}_l^{h-1}$, and so l's obligations at seniority $h-1$ are paid in full. Applying this argument inductively for seniorities $h-1, h-2, \ldots, 1$ shows that all l's obligations more senior than h are paid in full.

Now we prove that node l pays out all its available funds, i.e., that for node l, we have:

$$\sum_{k=1}^{K}\sum_{j=1}^{N}\Pi_{jl}^{k}p_j^{k*} + e_l + c_l = \sum_{k=1}^{K}p_l^{k*}. \tag{1.19}$$

If this is not the case, then $\sum_{k=1}^{K}\sum_{j=1}^{N}\Pi_{jl}^{k}p_j^{k*} + e_l + c_l > \sum_{k=1}^{K}p_l^{k*}$. We construct a new solution $\mathbf{p}^{k\epsilon}$, $k = 1, 2, \ldots, K$, which is equal to $\mathbf{p}^{k*}$, $k = 1, 2, \ldots, K$, in all components except p_l^h. We set $p_l^{h\epsilon} = p_l^{h*} + \epsilon$, where $\epsilon > 0$ is small enough to ensure that $p_l^{h\epsilon} < \bar{p}_l^h$ and $\sum_{k=1}^{K}\sum_{j=1}^{N}\Pi_{jl}^{k}p_j^{k\epsilon} + e_l + c_l > \sum_{k=1}^{K}p_l^{k\epsilon}$. Since Π is a three-dimensional matrix with non-negative entries, for any node $i \neq l$, we have:

$$\sum_{k=1}^{K}\sum_{j=1}^{N}\Pi_{ji}^{k}p_j^{k\epsilon} + e_i + c_i \geq \sum_{k=1}^{K}\sum_{j=1}^{N}\Pi_{ji}^{k}p_j^{k*} + e_i + c_i \geq \sum_{k=1}^{K}p_i^{k*} = \sum_{k=1}^{K}p_i^{k\epsilon},$$

Thus, the new solution ($\mathbf{c}^*$, $\mathbf{p}^{k\epsilon}$, $\mathbf{d}^{k*}$), $k = 1, 2, \ldots, K$, is also in the feasible region of (1.12 - 1.18) and achieves a larger value of the objective function than ($\mathbf{c}^*$, $\mathbf{p}^{k*}$, $\mathbf{d}^{k*}$), $k = 1, 2, \ldots, K$. This contradicts the fact that ($\mathbf{c}^*$, $\mathbf{p}^{k*}$, $\mathbf{d}^{k*}$), $k = 1, 2, \ldots, K$, is a solution to (1.11). Hence, Eq. (1.19) holds and $\mathbf{p}^{k*}$, $k = 1, 2, \ldots, K$, are clearing payment vectors.

Second, we prove by contradiction that $\mathbf{c}^*$ is the optimal cash injection allocation. Assume $\mathbf{c}^+ \neq \mathbf{c}^*$ leads to a strictly smaller value of the weighted sum of unpaid liabilities than does $\mathbf{c}^*$. In other words, suppose that $\mathbf{c}^+$ satisfies the constraints

(1.12) and (1.13) and that the corresponding clearing payment vectors $\mathbf{p}^{k+}$ satisfy $\sum_{k=1}^{K} \mathbf{w}^{kT}(\bar{\mathbf{p}}^k - \mathbf{p}^{k+}) < \sum_{k=1}^{K} \mathbf{w}^{kT}(\bar{\mathbf{p}}^k - \mathbf{p}^{k*})$, which is equivalent to:

$$\sum_{k=1}^{K} \mathbf{w}^{kT}\mathbf{p}^{k+} > \sum_{k=1}^{K} \mathbf{w}^{kT}\mathbf{p}^{k*}.$$

Since $\mathbf{p}^{k+}$, $k = 1, 2, \ldots, K$, are the corresponding clearing payment vectors, constraints (1.14) and (1.15) are satisfied. Moreover, if we define d_i^{k+} as the binary variable indicating whether node i fully repays its liabilities with seniority k, then constraints (1.16), (1.17) and (1.18) are also satisfied. So ($\mathbf{c}^+$,$\mathbf{p}^{k+}$,$\mathbf{d}^{k+}$), $k = 1, 2, \ldots, K$, is in the feasible region of (1.11–1.18) and achieves a larger value of the objective function than ($\mathbf{c}^*$, $\mathbf{p}^{k*}$, $\mathbf{d}^{k*}$), $k = 1, 2, \ldots, K$, which contradicts the fact that ($\mathbf{c}^*$, $\mathbf{p}^{k*}$, $\mathbf{d}^{k*}$), $k = 1, 2, \ldots, K$, is the solution of (1.11–1.18). ■

In MILP (1.11), constraints (1.12)-(1.13) are on the cash injection vector. All the cash injections must be non-negative and the total amount must not exceed the overall cash injection budget. Constraints (1.14)-(1.15) ensure that the actual payment of a node cannot exceed its total liability or its total available funds. Constraints (1.16)-(1.18) enforce the requirements that any node i can only make an incomplete repayment at any seniority k if its repayments at the more junior seniorities are all zeros; and that it can only make a nonzero repayment at seniority k if it repays in full all the obligations senior to k.

1.4.3 Numerical Simulations

To solve MILP (1.11), we use CVX, a package for specifying and solving convex programs and also MILPs [2,3]. In CVX, we select *Mosek* to be the solver [46]. A variety of prior literature, e.g. [6], suggests that the US interbank network is well modeled as a core-periphery network that consists of a core of about 15 highly interconnected banks to which most other banks connect. Therefore, we test the running time on a modified core-periphery network with multiple seniorities, as shown in Fig. 1.1. It

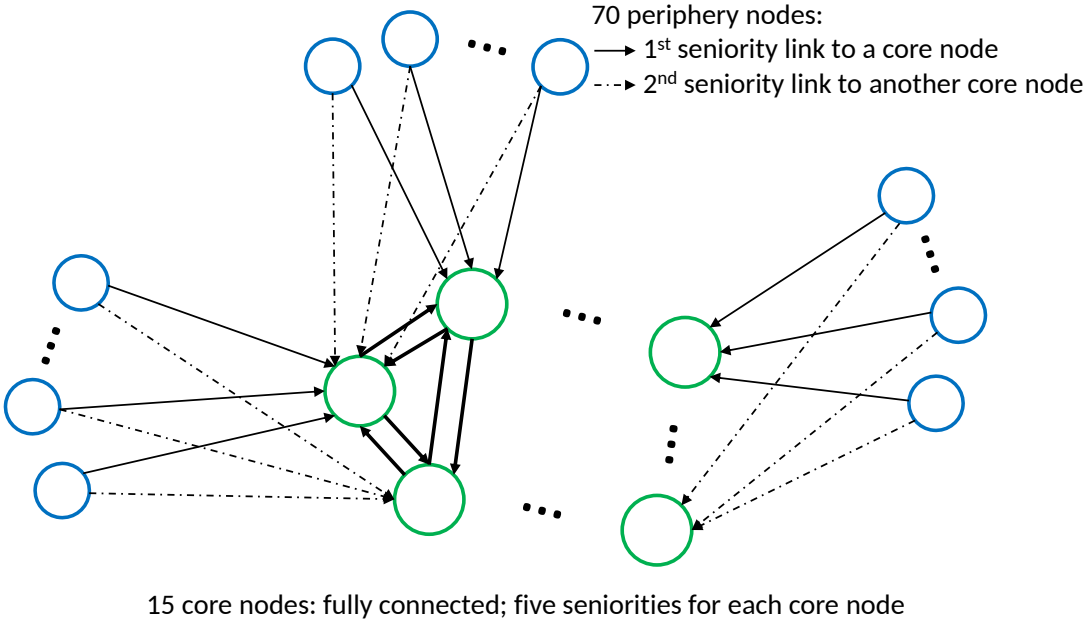


Fig. 1.1. A core-periphery network with liabilities with multiple seniorities.

contains 15 fully connected core nodes, denoted by core node 1 to core node 15. For each pair of core nodes i and j, there are five links with different seniorities. The amount of liability L_{ij}^k $(k = 1, 2, \ldots, 5)$ is uniformly distributed in $[0, 10]$. Each core node i has 70 periphery nodes. Each periphery node has a single link with seniority 1 pointing to the corresponding core node. In addition, each periphery node of core node i has another link with seniority 2 pointing to a random core node (uniformly selected among core nodes 1 to 15). For a core node i and its periphery node l, the obligation amount L_{li}^k $(k = 1, 2)$ is uniformly distributed in $[0, 1]$. All the obligation amounts are independent. Every node has zero external assets: $\mathbf{e} = \mathbf{0}$. For a core node i, we set the weight $w_i^k = 10$ for $k = 1, 2, \ldots, 5$; for a periphery node l, we set the weight $w_l^k = 1$ for $k = 1, 2$. The regulator has \$300 to be injected into the network. For this modified core-periphery network, we generate 100 samples. We run the CVX code on a personal computer with a 2.66GHz Intel Core2 Duo Processor P8800. The average running time is $9.85s$ and the sample standard deviation is $0.15s$. The relative gap between the objective of the solution and the optimal objective is less than 10^{-4}. (This bound is obtained by calculating the optimal value of the objective for the corresponding linear program, which is an upper bound for the optimal objective value of the MILP.) We can see that for the core-periphery network, MILP (1.11) can be solved by CVX efficiently and accurately. The CVX code is given in Appendix B.1.

1.5 Problem I with Credit Default Swaps

1.5.1 Clearing with CDSs

In this section, we incorporate credit default swaps (CDSs) into our framework. For simplification, we just consider the system with only one seniority. As defined in [38], a CDS is a contract whereby the seller node i insures the buyer node j against the default of node l on its underlying liabilities. In other words, a liability from node i to node j is created if node l does not pay its liability in full.

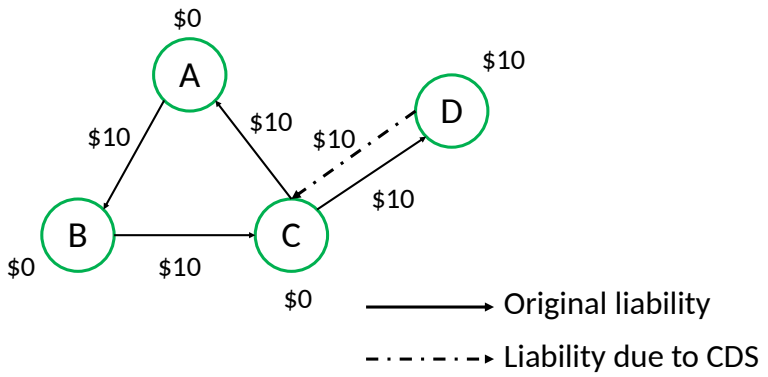


Fig. 1.2. Example showing a simultaneous clearing payment vector may not exist in a system with CDSs.

With CDSs in the system, a simultaneous clearing payment vector does not necessarily exist. For example, in Fig. 1.2, node A owes \$10 to node B; node B owes \$10 to node C and node C owes \$10 to A and \$10 to D. Node D sells a CDS to node C such that if node B defaults, node D will pay \$10 to node C as a compensation. Initially, there is no cash among A, B and C. Node D has \$10 on hand. Without CDSs, no nodes are able to make their payments so that A, B and C default. Then node D pays node C \$10 according to the CDS contract. With this \$10 on C's hand, all the liabilities in the system are cleared so that all nodes are rescued including B, which makes C ineligible for D's payment. Thus, there is no simultaneous clearing payment vector in this network.

Instead of the simultaneous clearing scheme, we clear the system with CDSs in three stages:

1. The system clears its original liabilities without considering CDSs.

2. New liabilities due to CDSs are created. The system clears these liabilities due to CDSs.

3. The system clears the remaining original liabilities.

In each stage above, the system is cleared according to the simultaneous clearing method in [1]. Thus, there exists a unique clearing payment vector for each stage.

As defined in Section 1.2, the cash owned by node i, the external cash injection into node i, and the original liability from node i to node j are denoted by e_i, c_i, and L_{ij}, respectively. We let d_i be the stage 1 default indicator for node i. The CDS-induced obligation from node i to node j triggered by the default of node l is denoted by D_{ij}^l. We define x_i and z_i to be the fractions of node i's total underlying liability $\sum_{j=1}^{N} L_{ij}$ that node i repays during stages 1 and 3, respectively. Thus, node i's total payments during stages 1 and 3 are, respectively, $\left(\sum_{j=1}^{N} L_{ij}\right) x_i$ and $\left(\sum_{j=1}^{N} L_{ij}\right) z_i$, and node i's total unpaid underlying liability is $\left(\sum_{j=1}^{N} L_{ij}\right) (1 - x_i - z_i)$.

We moreover let y_i be the fraction of node i's total CDS-induced liability that node i repays during stage 2. Furthermore, for any two nodes l and i, we define

$$y_i^l = \begin{cases} y_i & \text{if node } l \text{ defaults during stage 1,} \\ 0 & \text{if node } l \text{ does not default during stage 1.} \end{cases}$$

Then node i's stage 2 payment that is determined by node l's stage 1 default status is $\left(\sum_{j=1}^{N} D_{ij}^l\right) y_i^l$. Note that node i's stage 2 liability related to node l's stage 1 default status is $\left(\sum_{j=1}^{N} D_{ij}^l\right) d_l$ where d_l is node l's stage 1 default indicator. In other words, if l defaults in stage 1, i.e., if $d_l = 1$, then i owes $\sum_{j=1}^{N} D_{ij}^l$ in CDS-induced liabilities related to l; and if l does not default, i.e., if $d_l = 0$, then i does not have any CDS-induced liabilities related to l. Therefore, node i's total unpaid CDS-induced liabilities related to node l's stage 1 default status are $\left(\sum_{j=1}^{N} D_{ij}^l\right)(d_l - y_i^l)$. Thus, the total unpaid liability for the system over the three stages is:

$$\sum_{i=1}^{N}\left(\sum_{j=1}^{N} L_{ij}\right)(1 - x_i - z_i) + \sum_{l=1}^{N}\sum_{i=1}^{N}\left(\sum_{j=1}^{N} D_{ij}^l\right)(d_l - y_i^l).$$

This is the objective function we would like to minimize. As previously, the framework we develop in this section is applicable to weighted sums of liabilities; however, we omit the weights in order to simplify notation.

1.5.2 Minimizing the weighted sum of unpaid liabilities with CDSs

Assume the system clears the liabilities as described in Section 1.5.1. We investigate the problem of minimizing the sum of unpaid liabilities under the model with CDSs.

In [1], it is proved that the clearing payment is unique when the system is regular, which is always the case if all nodes have some cash on hand. In this section, we assume the system is regular. Otherwise, we just give each node \$1 to make it regular.

The following constraints for all i combined with constraints (1.5) and (1.6) will guarantee a clearing payment for the system in stage 1:

$$0 \leq x_i \leq 1 \tag{1.20}$$

$$\left(\sum_{j=1}^{N} L_{ij}\right) x_i \leq \sum_{j=1}^{N} L_{ji} x_j + e_i + c_i, \tag{1.21}$$

$$\sum_{j=1}^{N} L_{ji} x_j + e_i + c_i - \left(\sum_{j=1}^{N} L_{ij}\right) x_i \leq \frac{1}{\epsilon}(1 - d_i), \tag{1.22}$$

$$(1 - d_i) \leq x_i, \tag{1.23}$$

$$d_i \in \{0, 1\}, \tag{1.24}$$

$$1 - x_i \geq \epsilon d_i. \tag{1.25}$$

where ϵ is a small positive constant.

To show that these constraints are consistent with our definitions of $\mathbf{x}$ as stage 1 clearing payment vector rescaled to $[\mathbf{0}, \mathbf{1}]$ and $\mathbf{d}$ as stage 1 default indicator vector, first note that, because of constraints (1.20) and (1.23), having $d_i = 0$ would imply that $x_i = 1$, i.e., that node i fully repays its obligations during stage 1. Furthermore, if $d_i = 1$, then constraints (1.21) and (1.22) force node i's outgoing payments $\left(\sum_{j=1}^{N} L_{ij}\right) x_i$ to be equal to its available funds $\sum_{j=1}^{N} L_{ji} x_j + e_i + c_i$. Thus, any feasible $\mathbf{x}$ in the region defined by constraints (1.20) - (1.25) is a valid clearing payment vector. Conversely, note that if $x_i = 1$ (i.e., if node i fully repays its liabilities during stage 1), then constraints (1.24) and (1.25) imply $d_i = 0$. If $x_i < 1$ (i.e., if node i does not repay its entire liability during stage 1), then constraints (1.23) and (1.24) imply $d_i = 1$. Thus, d_i is a valid default indicator for node i. Therefore, for any feasible point in the region defined by constraints (1.20) - (1.25), $\mathbf{x}$ is a valid clearing payment vector rescaled to $[\mathbf{0}, \mathbf{1}]$, and $\mathbf{d}$ is a valid indicator showing whether node i is in default after stage 1.

Note that the LP from Theorem 1 which is formulated in Eqs. (1.4 - 1.8), is somewhat different from the optimization problem of maximizing the objective of

Eq. (1.4) subject to the constraints of Eqs. (1.5,1.6, 1.20 - 1.25), in the sense that there are some cases when the latter is infeasible, and there are also cases when the solutions to the two problems slightly differ. To explain this, we denote the latter optimization problem by P_1. When the total available funds of node i are smaller than but very close to its total liability, x_i might be less than 1 and greater than $1-\epsilon$. Then from constraint (1.23), we have $d_i = 0$ and from constraint (1.25), we have $d_i = 1$, which is a contradiction. Then, the feasible region of P_1 would be empty. Such cases are atypical since the problematic region for clearing vectors is very small when ϵ is small. Algorithmically, such cases can be resolved by slightly increasing the total cash amount C to let node i fully repay its liabilities, i.e., to let $x_i = 1$. Now suppose that the optimal solution to LP (1.4 - 1.8) has $1-\epsilon < x_i < 1$ and that P_1 is feasible. Then the optimal solution to P_1 would use a small amount of cash injection to ensure $x_i = 1$. In this case, the solution to P_1 will be slightly different from the solution to LP (1.4 - 1.8), but the difference between the values of x_i in the two solutions will be smaller than ϵ.

Now we consider stage 2: clearing the liabilities due to CDSs. After stage 1, the cash on hand at node i is $f_i = \sum_{j=1}^{N} L_{ji}x_j + e_i + c_i - \left(\sum_{j=1}^{N} L_{ij}\right) x_i$. Note that during stage 2, there may be further defaults among the sellers of the CDSs. The

clearing payment for the liabilities due to CDSs is located in the region defined by the following constraints:

$$0 \leq y_i^l \leq d_l, \tag{1.26}$$

$$y_i - y_i^l \leq 1 - d_l, \tag{1.27}$$

$$y_i^l - y_i \leq 1 - d_l, \tag{1.28}$$

$$\sum_{l=1}^{N}\sum_{j=1}^{N} D_{ij}^l y_i^l \leq \sum_{l=1}^{N}\sum_{j=1}^{N} D_{ji}^l y_j^l + f_i, \tag{1.29}$$

$$\sum_{l=1}^{N}\sum_{j=1}^{N} D_{ji}^l y_j^l + f_i - \sum_{l=1}^{N}\sum_{j=1}^{N} D_{ij}^l y_i^l \leq \frac{1}{\epsilon}(1 - d_i^{CDS}), \tag{1.30}$$

$$1 - d_i^{CDS} \leq y_i, \tag{1.31}$$

$$d_i^{CDS} \in \{0, 1\}. \tag{1.32}$$

Constraint (1.26) ensures that if node l is not in default, then $y_i^l = 0$ so that there are no payments associated with any CDSs written against l's default. Constraints (1.27) and (1.28) guarantee that if both node l and node h are in default, the liabilities due to the defaults of l and h are paid proportionally, i.e., $y_i^l = y_i^h = y_i$. Binary variable d_i^{CDS} is used to enforce the condition that any node whose stage 2 funds do not exceed the total stage 2 liability must pay out all its funds during stage 2. In most cases, d_i^{CDS} is the stage 2 default indicator variable for node i. However, in a degenerate case when node i's stage 2 funds are exactly equal to its stage 2 total liabilities, d_i^{CDS} may be either 0 or 1.

In stage 3, we clear the remaining original liabilities which are not fully paid in stage 1. The cash on hand at node i after stage 2 is $r_i = \sum_{l=1}^{N}\sum_{j=1}^{N} D_{ji}^l y_j^l + f_i - \sum_{l=1}^{N}\sum_{j=1}^{N} D_{ij}^l y_i^l$. Then the constraints to obtain the clearing payment in stage 3 are:

$$0 \leq z_i \leq 1 - x_i \tag{1.33}$$

$$\left(\sum_{j=1}^{N} L_{ij}\right) z_i \leq \sum_{j=1}^{N} L_{ji} z_j + r_i, \tag{1.34}$$

where x_i is the fraction of node i's underlying liabilities paid off in stage 1. Here we do not need the binary indicator any more since it is the last round so that the clearing payment is achieved if the objective function is strictly increasing with respect to z_i.

We therefore have the following result.

Theorem 3 *Optimal cash allocation to minimize the sum of the unpaid liabilities in the model with CDSs can be obtained via the following mixed integer linear program:*

$$\max_{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{d},\mathbf{d}^{CDS},\mathbf{c}} \sum_{i=1}^{N}\left(\sum_{j=1}^{N} L_{ij}\right)(x_i+z_i) + \sum_{l=1}^{N}\sum_{i=1}^{N}\left(\sum_{j=1}^{N} D_{ij}^{l}\right)(y_i^l - d_l) \tag{1.35}$$

subject to

$$\sum_{i=1}^{N} c_i \le C, \tag{1.36}$$

$$c_i \ge 0, \tag{1.37}$$

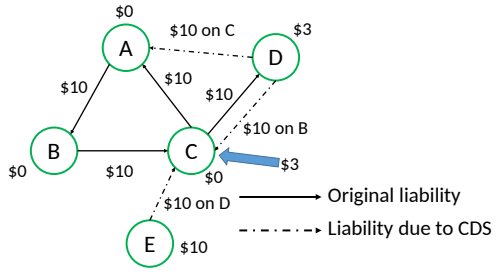
$(1.20) - (1.34)$.

1.5.3 Numerical Simulations

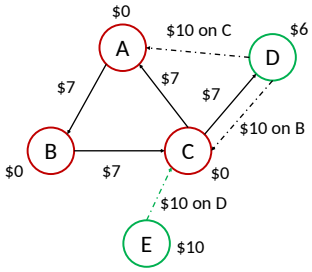
To solve MILP (1.35), we use CVX, a package for specifying and solving convex programs and also MILPs [2, 3]. In CVX, we select *Gurobi* to be the solver [47].

1.5.3.1 Example 1: A Five-Node Network

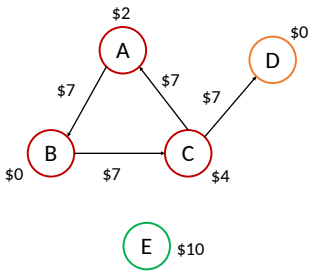
In this section, we illustrate and verify Theorem 3 on the five-node network shown in Fig. 1.3(a). Node A owes \$10 to B, B owes \$10 to C, C owes \$10 to A and \$10 to D respectively. If node C defaults in stage 1, node D has a \$5 CDS-induced liability to node A. If node B defaults in stage 1, node D has a \$10 CDS-induced liability to node C. If node D defaults in stage 1, node E has a \$10 CDS-induced liability to node C. Node A, B and C have no cash on hand. Node C has \$3. Nodes D and E each have \$10 on hand. The regulator has \$3 to be injected into the system.



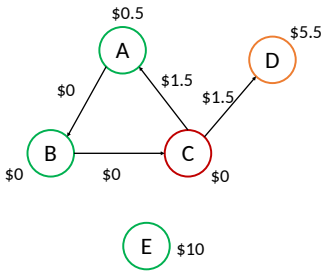
(a) Initial network.



(b) After stage 1.



(c) After stage 2.



(d) After stage 3.

Fig. 1.3. A five-node network.

We solve MILP (1.35) for this network via CVX and obtain cash injection vector $\mathbf{c} = [0, 0, 3, 0, 0]^T$, which directs the regulator to inject \$3 into node C. With \$3 cash injection to C, the original liabilities are cleared in stage 1. After the clearance in stage 1, the network is reduced to Fig. 1.3(b). The scaled payment vector is $\mathbf{x} = [0.3, 0.3, 0.3, 0, 0]^T$, which means that nodes A, B and C pay 30% of their liabilities. The stage 1 default indicator vector is $\mathbf{d} = [1, 1, 1, 0, 0]^T$, so nodes A, B and C are in default in stage 1, while D and E are safe.

Since nodes B and C default in stage 1, the CDS-induced liabilities related to B and C are created in stage 2, i.e., node D owes \$5 to A and \$10 to C in stage 2. The CDS-induced liability from E to C is not triggered since D does not default in stage 1. In the solution of MILP (1.35) for this network, we have $y_D^A = y_D^C = y_D = 0.4$ so that in stage 2, node D pays 40% of its CDS-induced liabilities. In addition, the

scaled payment from E to C is $y_E^C = 0$ due to constraint (1.26). The network after stage 2 is shown in Fig. 1.3(c).

In stage 3, the remaining original liabilities are cleared. The final network after stage 3 is shown in Fig. 1.3(d). The scaled clearing payment vector is $\mathbf{z} = [0.7, 0.7, 0.55, 0, 0]^T$.

To sum up, nodes A and B pay 30% of their original liabilities in stage 1 and 70% of their original liabilities in stage 3. Technically they are both in default, since in our model the inability to fully repay the underlying obligations in stage 1 is defined as default. Note, however, that because of the CDS-related payments cleared in stage 2, both of them are able to fully repay their remaining underlying obligations in stage 3. Thus, even though A and B both default, their defaults have a 100% recovery rate, i.e., their creditors are fully repaid after stage 3. Node C defaults and has \$3 unpaid liabilities after three-stage clearing. Node D does not have any original liabilities, but it defaults in stage 2 since it fails to fully pay its CDS-induced liabilities. Node E stays safe since it does not have any original liabilities and the CDS-induced liability is not triggered.

1.5.3.2 Example 2: A Core-Periphery Network

In this section we examine the practicality of the mixed-integer linear program. We test the running time on a core-periphery network with CDSs shown in Fig. 1.17. It contains 15 fully connected core nodes. Each core node has 70 periphery nodes. Each periphery node has a single link pointing to the corresponding core node. The asset of each node is uniformly distributed in $[0, 0.01]$ so that the system is regular. All the obligation amounts $L_{i,j}$ are independent uniform random variables. For each pair of core nodes i and j the obligation amount L_{ij} is uniformly distributed in $[0, 10]$. For a core node i and its periphery node k, the obligation amount L_{ki} is uniformly distributed in $[0, 1]$. We assume only core nodes buy and sell CDSs. For any pair of core nodes i, j, and the underlying, node l, node i sells a CDS with underlying l to

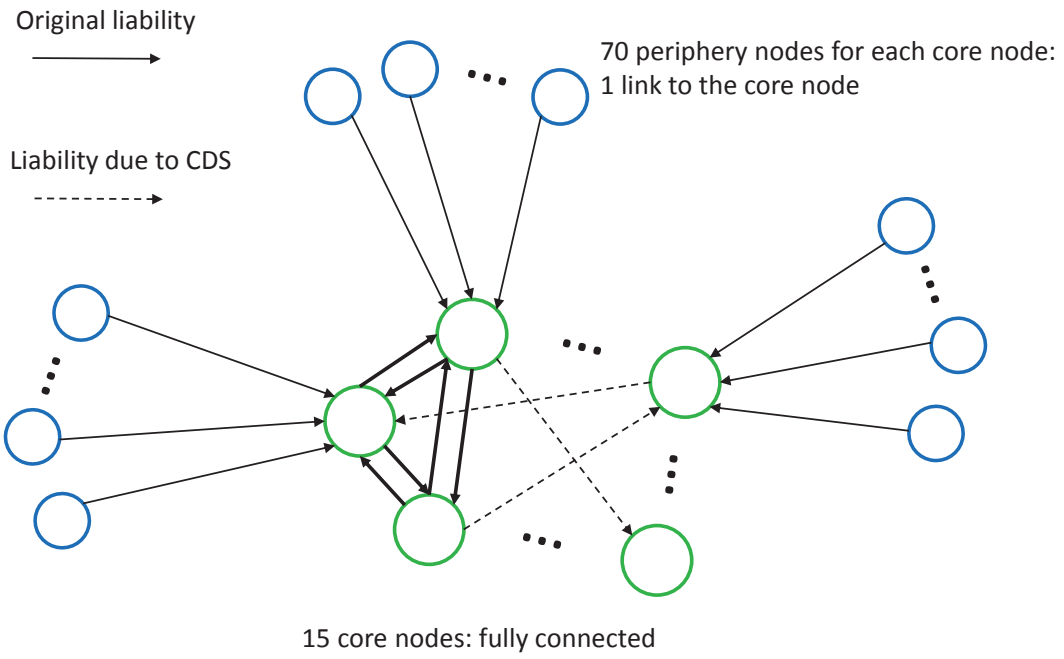


Fig. 1.4. A core-periphery network with CDSs.

node j with probability 0.01. The amount of the CDS-induced liability is uniform in $[0, 1]$. In other words, D^l_{ij} is uniformly distributed in $[0, 1]$ with probability 0.01, and zero with probability 0.99. The weights of original liabilities and the weights of CDS-induces liabilities are all the same , i.e., $\mathbf{w} = \mathbf{1}$. The regulator has \$100 to be injected into the network. For this core-periphery network, we generate 100 samples. We run the CVX code on a personal computer with a 2.66GHz Intel Core2 Duo Processor P8800. The average running time is $7.0s$ and the sample standard deviation is $3.6s$. The relative gap between the objective of the solution and the optimal objective is less than 10^{-4}. (This bound is obtained by calculating the optimal value of the objective for the corresponding linear program, which is an upper bound for the optimal objective value of the MILP.) We can see that for the core-periphery network, MILP (1.35) can be solved by CVX efficiently and accurately. The CVX code is given in Appendix B.2.

1.6 Heuristic Algorithms for Problem II under the Proportional Payment Mechanism

Given that the total amount of cash injection is C, Problem II seeks to find a cash injection allocation vector $\mathbf{c}$ to minimize the number of defaults N_d, i.e., the number of nonzero entries in vector $\bar{\mathbf{p}} - \mathbf{p}$. In other words, in Problem II, we would like to make vector $\bar{\mathbf{p}} - \mathbf{p}$ as sparse as possible.

In this section, we propose two heuristic algorithms to solve Problem II approximately. First, we adapt the reweighted ℓ_1 minimization strategy approach from Section 2.2 of [4]. Our algorithm solves a sequence of weighted versions of the linear program (1.4-1.8), with the weights designed to encourage sparsity of $\bar{\mathbf{p}} - \mathbf{p}$. In the following pseudo code of our algorithm, $\mathbf{w}^{(m)}$ is the weight vector during the m-th iteration.

Reweighted ℓ_1 minimization algorithm:

1. $m \leftarrow 0$.

2. Select $\mathbf{w}^0$ (e.g., $\mathbf{w}^0 \leftarrow \mathbf{1}$).

3. Solve linear program (1.4-1.8) with objective function replaced by $\mathbf{w}^{(m)T}\mathbf{p}$.

4. Update the weights: for each $i = 1, \cdots, N$,

$$w_i^{(m+1)} \leftarrow \frac{1}{\exp\left(\bar{p}_i - p_i^{*(m)}\right) - 1 + \epsilon},$$

   where $\epsilon > 0$ is constant, and $\mathbf{p}^{*(m)}$ is the clearing payment vector obtained in Step 3.

5. If $\|\mathbf{w}^{(m+1)} - \mathbf{w}^{(m)}\|_1 < \delta$, where $\delta > 0$ is a constant, stop; else, increment m and go to Step 3.

Note that nodes for which $\bar{p}_i - p_i^{*(m)}$ is very small require very little additional resources to avoid default. This is why Step 4 is designed to give more weight to such nodes, thereby encouraging larger cash injections into them. On the other hand, nodes for which $\bar{p}_i - p_i^{*(m)}$ is very large require a lot of cash to become solvent. The algorithm essentially "gives up" on such nodes by assigning them small weights.

The second heuristic algorithm we develop is a greedy algorithm. At each iteration of the greedy algorithm, we calculate the clearing payment vector and select the node with the smallest unpaid liability among all the defaulting nodes. We inject cash into that node to rescue it so that during each iteration, we save the one node that requires the smallest cash expenditure. In this procedure, we inject the cash sequentially, bailing out some nodes completely before they fully receive the payments from their borrowers. These nodes may subsequently receive some more cash from their borrowers if their borrowers are rescued several steps later. Because of this, a rescued node may end up with a surplus. If this happens, the node would use its surplus to repay its cash injection. Such repayments can then be used to assist other nodes. The algorithm terminates either when there are no defaults in the system or when the injected cash reaches the total amount C and no rescued node has a surplus.

Greedy algorithm:

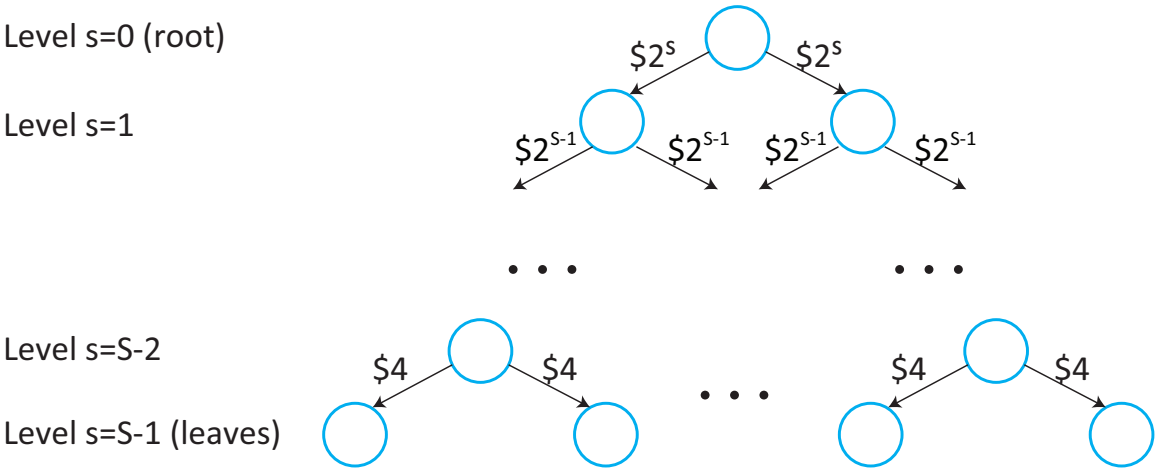


Fig. 1.5. Binary tree network.

1. $C_r \leftarrow C$, $\mathbf{c} \leftarrow \mathbf{0}$, $\mathbf{w} \leftarrow \mathbf{1}$.

2. Solve linear program (1.1-1.3) to obtain the clearing payment vector $\mathbf{p}$.

3. Calculate the surplus of each node after clearing: $\mathbf{r} \leftarrow \Pi^T \mathbf{p} + \mathbf{e} + \mathbf{c} - \mathbf{p}$.

4. Update the remaining cash to be injected into the system after the rescued nodes repay their cash injections: $C_r \leftarrow C_r + \sum_{i=1}^{N} \min\{r_i, c_i\}$, $c_i \leftarrow c_i - \min\{c_i, r_i\}$ for $i = 1, 2, \cdots, N$.

5. If $C_r = 0$ or there are no defaults in the system, stop.

6. Find node k with the minimum unpaid liability $\bar{p}_k - p_k$ among all defaulting nodes.

7. $c_k \leftarrow \min\{C_r, \bar{p}_k - p_k\}$, $C_r \leftarrow C_r - c_k$, go to Step 2.

We now illustrate our two heuristic algorithms using three simple synthetic networks for which the optimal solutions can be calculated exactly.

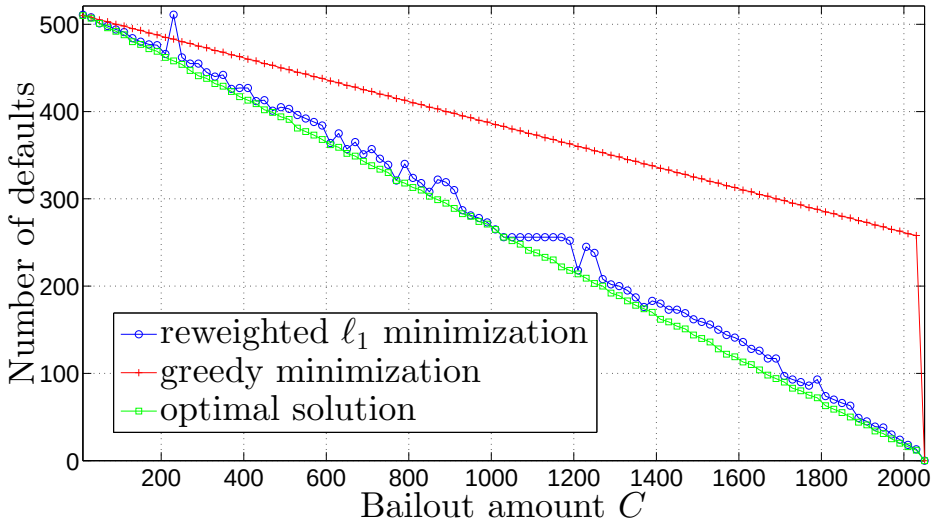


Fig. 1.6. Our algorithms for minimizing the number of defaults vs the optimal solution calculated in Section 1.6.1, for the binary tree network of Fig. 1.5.

1.6.1 Example: A Binary Tree Network

First, we use a full binary tree with S levels and $N = 2^S - 1$ nodes. As shown in Fig. 1.5, levels 0 and $S-1$ correspond to the root and the leaves, respectively. Every node at level $s < S-1$ owes $\$2^{S-s}$ to each of its two creditors (children). We set $\mathbf{e} = \mathbf{0}$.

If $C < 8$, then all $2^{S-1} - 1$ non-leaf nodes are in default, and the 2^{S-1} leaves are not in default. In aggregate, the nodes at any level $s < S-1$ owe $\$2^{S+1}$ the nodes at level $s+1$. Therefore, if $C \geq 2^{S+1}$, then $N_d = 0$ can be achieved by allocating the entire amount to the root node.

For $8 \leq C < 2^{S+1}$, we first observe that if $C = 2^{S+1-s}$ for some integer s, then the optimal solution is to allocate the entire amount to a node at level s. This would prevent the defaults of this node and all its $2^{S-s-1} - 2$ non-leaf descendants, leading to $2^{S-1} - 2^{S-s-1}$ defaults. If C is not a power of two, we can represent it as a sum of

powers of two and apply the same argument recursively, to yield the following optimal number of defaults:

$$N_d = T(S) - \sum_{u=4}^{U} b(u) \cdot T(u-2),$$

where $T(x) = 2^{x-1} - 1$ is the number of non-leaf nodes in an x-level complete binary tree, $b(u)$ is the u-th bit in the binary representation of C (right to left) and U is the number of bits. To summarize, the smallest number of defaults N_d, as a function of the cash injection amount C, is:

$$N_d(C) = \begin{cases} T(S) & \text{if } C < 8, \\ T(S) - \sum_{u=4}^{U} b(u) T(u-2) & \text{if } 8 \le C < 2^{S+1}, \\ 0 & \text{if } C \ge 2^{S+1}. \end{cases} \tag{1.38}$$

In our test, we set $S = 10$. The green line in Fig. 1.6 represents the minimum number of defaults as a function of C, as shown in Eq. (1.38). The blue line is the solution calculated by the reweighted ℓ_1 minimization algorithm with $\epsilon = 0.001$ and $\delta = 10^{-6}$. The algorithm was run using six different initializations: five random ones and $\mathbf{w^{(0)}} = \mathbf{1}$. Among the six solutions, the one with the smallest number of defaults was selected. The red line is the solution provided by the greedy algorithm. Fig. 1.6 shows that the results of the reweighted ℓ_1 minimization algorithm are very close to the optimal for the entire range of C, while the performance of the greedy algorithm is poor. The greedy algorithm always injects cash into the nodes at level $S-2$ which have the smallest unpaid liabilities. For $C \ge 16$, this strategy is inefficient since spending \$16 on a node at level $S-3$ rescues both that node and its two children, whereas spending \$16 on two nodes at level $S-2$ only rescues those two nodes.

To investigate the sensitivity of the reweighted ℓ_1 minimization algorithm to our random initialization strategy, we conduct the following experiment with the binary tree topology. For each of the 11 amounts of cash injection $C = 0, 200, 400, \ldots, 2000$, we repeat the algorithm 100 times. The five random initial weights in all these runs are chosen independently. The results are shown in Fig. 1.7. The green squares mark the numbers of defaults for the optimal solutions. The blue crosses show, for each

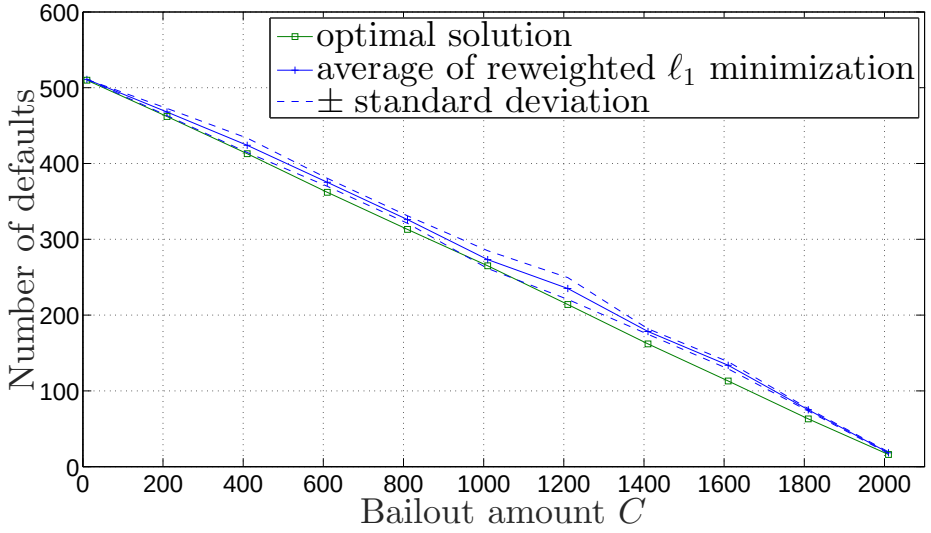


Fig. 1.7. Reweighted ℓ_1 minimization algorithm with different initializations for the binary tree network of Fig. 1.5.

of the 11 values of C, the average number of defaults over 100 repetitions of the reweighted ℓ_1 minimization algorithm. The dashed blue lines show the band of ± 1 standard deviations around the mean. Note that the means are close to the optimal solutions, and that the standard deviations are small, indicating that the algorithm is robust to the initial weights.

1.6.2 Example: A Network with Cycles

Second, we test our algorithms on the network with cycles shown in Fig. 1.8. The network contains M cycles with six nodes each. The nodes in the k-th cycle are denoted $n_{k1}, n_{k2}, \cdots, n_{k6}$. Node n_{k1} owes $\$2a$ to n_{k2}. Node n_{k6} owes $\$a$ to n_{k1}. For $i = 2, \cdots, 5$, n_{ki} owes $\$a$ to $n_{k(i+1)}$. The root node, denoted as n_R, owes $\$a$ to n_{k1}, for every $k = 1, 2, \cdots, M$. We set $\mathbf{e} = \mathbf{0}$.

If $C < a$, then the root node and all M nodes connected to the root, $n_{k1} (k = 1, 2, \ldots, M)$, are in default. The remaining $5M$ nodes are not in default.

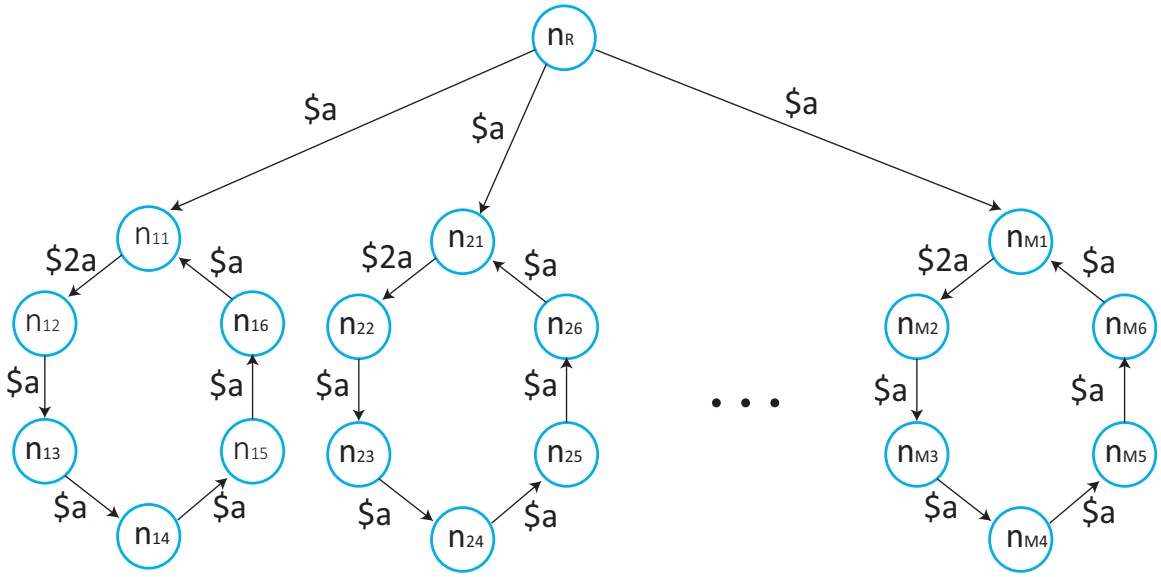


Fig. 1.8. Network topology with cycles.

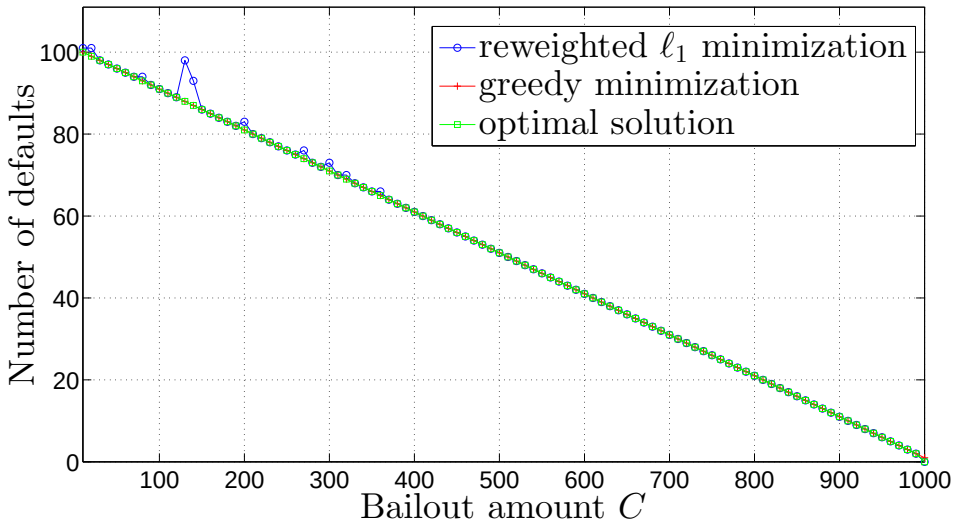


Fig. 1.9. Our algorithms for minimizing the number of defaults vs the optimal solution calculated in Section 1.6.2, for the network of Fig. 1.8.

If $C \geq aM$, then allocating the entire amount C to the root yields zero defaults.

If $a \leq C < aM$, then giving $\$a$ to node n_{k1} will prevent it from defaulting. Thus, the total number of defaults in this case is $M + 1 - [C/a]$.

Summarizing, for this network structure, the smallest number of defaults N_d, as a function of the cash injection amount C, is:

$$N_d(C) = \begin{cases} M+1 & \text{if } C < a, \\ M+1-[C/a] & \text{if } a \leq C < aM, \\ 0 & \text{if } C \geq aM. \end{cases} \tag{1.39}$$

In our test, we set $a = 10$ and $M = 100$. In Fig. 1.9, the green line is a plot of the minimum number of defaults as a function of C. The blue line is the solution calculated by the reweighted ℓ_1 minimization algorithm with $\epsilon = 0.001$ and $\delta = 10^{-6}$. The algorithm was run using six different initializations: five random ones and $\mathbf{w^{(0)}} = \mathbf{1}$. Among the six solutions, the one with the smallest number of defaults was selected. The red line is the solution calculated by the greedy algorithm. As evident from Fig. 1.9, the results produced by both algorithms are very close to the optimal ones. The greedy algorithm achieves the optimal solution for the entire range of C except the point $C = 1000$. When $C = 1000$, the optimal strategy is to inject \$1000 into the root node whereas the greedy algorithm injects \$10 into n_{k1} for $k = 1, 2, \cdots, 100$.

1.6.3 Example: A Core-Periphery Network

Third, we test our algorithm on a simple core-periphery network, since core-periphery models are widely used to model banking systems [7,8,48,49]. In Fig. 1.10, i, ii, and iii are the three core nodes. Node i owes \$100 each to nodes ii and iii, and node ii owes \$100 to iii. Ten periphery nodes are attached to each core node, and each periphery node owes \$20 to its core node. There are no external assets in the system, so in the absence of an external injection of cash, all the nodes are in default except node iii.

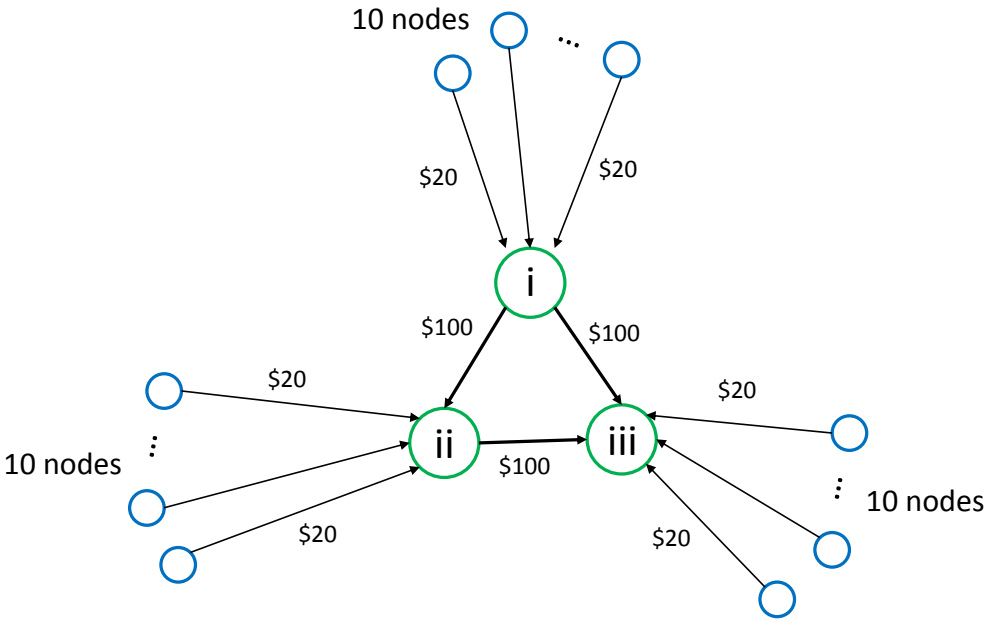


Fig. 1.10. Core-periphery network topology.

If the cash injection amount is $C < 100$, the optimal solution is to select any $\lceil C/20 \rceil$ periphery nodes and give \$20 to each of them. This reduces the number of defaults by $\lceil C/20 \rceil$.

If $100 \leq C < 200$, we first select any five periphery nodes of core node ii and give \$20 to each of them, because this saves both node ii and these five periphery nodes. Then we select any other $\lfloor (C-100)/20 \rfloor$ periphery nodes and give \$20 to each. This decreases the number of defaults by $\lfloor C/20 \rfloor + 1$.

If $200 \leq C < 600$, we first use \$200 to rescue all 10 periphery nodes of core node i, saving i, ii, and these 10 periphery nodes. Then we select any other $\lfloor (C-200)/20 \rfloor$ periphery nodes and give \$20 to each. This decreases the number of defaults by $\lfloor C/20 \rfloor + 2$.

If $C \geq 600$, then all the nodes can be rescued by giving \$20 to each periphery node.

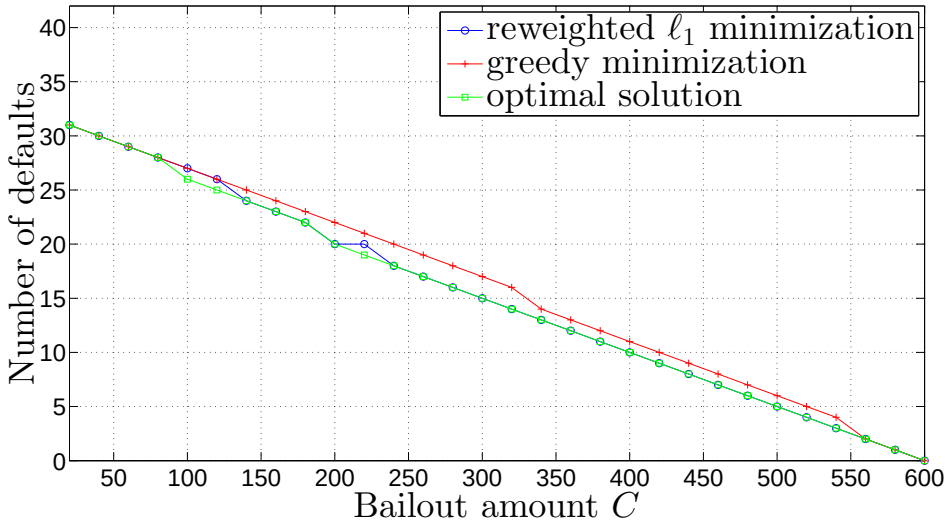


Fig. 1.11. Our algorithms for minimizing the number of defaults vs the optimal solution calculated in Section 1.6.3, for the network of Fig. 1.10.

To sum up, for this core-periphery network structure, the smallest number of defaults N_d, as a function of the cash injection amount C, is:

$$N_d(C) = \begin{cases} 32 - \lfloor C/20 \rfloor & \text{if } C < 100, \\ 31 - \lfloor C/20 \rfloor & \text{if } 100 \leq C < 200, \\ 30 - \lfloor C/20 \rfloor & \text{if } 200 \leq C < 600, \\ 0 & \text{if } C \geq 600. \end{cases} \tag{1.40}$$

In Fig. 1.11, the green line is a plot of this minimum number of defaults as a function of C. The blue line is the solution calculated by our reweighted ℓ_1 minimization algorithm with $\epsilon = 0.001$ and $\delta = 10^{-6}$. The algorithm was run using six different initializations: five random ones and $\mathbf{w}^{(\mathbf{0})} = \mathbf{1}$. Among the six solutions, the one with the smallest number of defaults was selected. The red line is the solution calculated by the greedy algorithm. As evident from Fig. 1.11, the results produced by the reweighted ℓ_1 algorithm are very close to the optimal ones for the entire range of C. Note that for the greedy algorithm, the performance depends on the order of

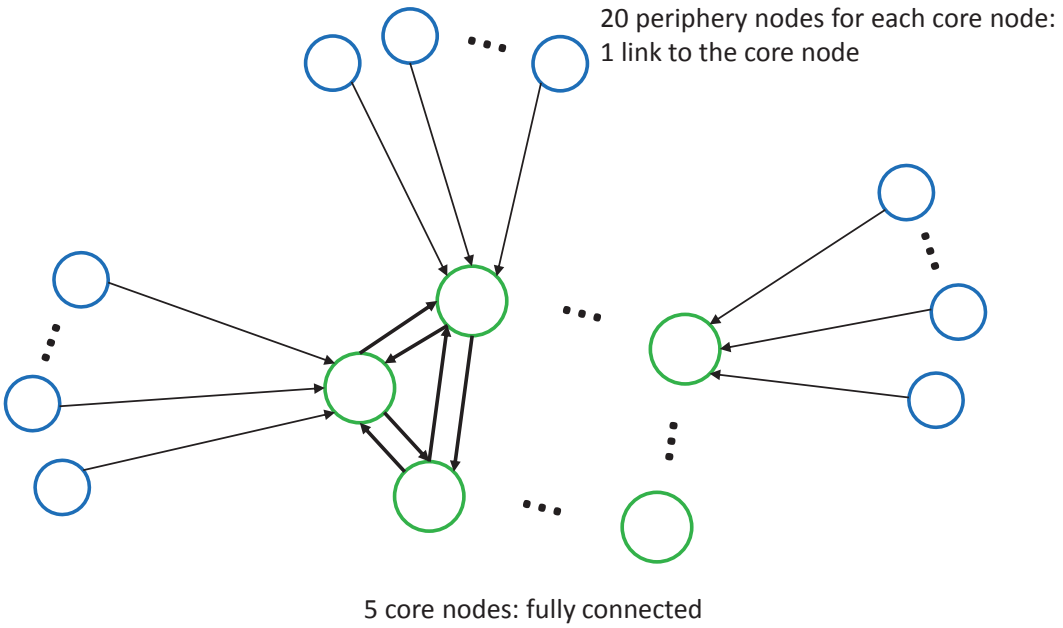


Fig. 1.12. Random core-periphery network to compare the reweighted ℓ_1 algorithm and the greedy algorithm.

rescuing nodes with the same unpaid liability amounts. For example, if the greedy algorithms rescue the periphery nodes of core node iii first, the performance would be poor.

1.6.4 Example: Three Random Networks

We now compare the reweighted ℓ_1 minimization algorithm to the greedy algorithm using more complex network topologies in which the optimal solution is difficult to calculate directly.

We construct three types of random networks, all having external asset vector $\mathbf{e} = \mathbf{0}$. The first one is a random graph with 30 nodes. For any pair of nodes i and j,

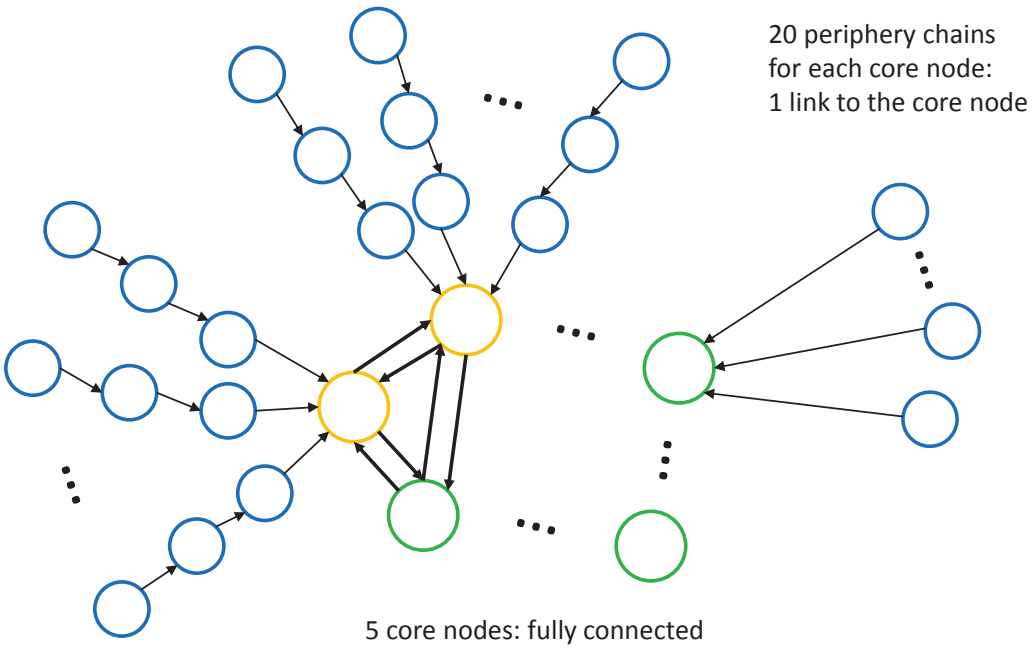


Fig. 1.13. Random core-periphery network with long chains to compare the reweighted ℓ_1 algorithm and the greedy algorithm.

L_{ij} is zero with probability 0.8 and is uniformly distributed in $[0, 2]$ with probability 0.2.

The second one is a random core-periphery network which is illustrated in Fig. 1.12. The core contains five nodes which are fully connected. The liability from one core node to every other core node is uniformly distributed in $[0, 20]$. Each core node has 20 periphery nodes. Each periphery node owes money only to its core node. This amount of money is uniformly distributed in $[0, 1]$.

The third one is a random core-periphery network with chains of periphery nodes. As shown in Fig. 1.13, the core contains five nodes which are fully connected. The liability from one core node to every other core node is uniformly distributed in [0,20]. Each core node has 20 periphery chains connected to it, each chain consisting of either

a single periphery node (short chains) or 3 periphery nodes (long chains). Each core node has either only short periphery chains connected to it or only long periphery chains connected to it. There are two core nodes with long periphery chains. The liability amounts along each long chain are the same, and are uniformly distributed in [0,1]. The liability amounts along each short chain are also uniformly distributed in [0,1].

For each of these three random networks, we generate 100 samples from the distribution and run both the reweighted ℓ_1 minimization algorithm and the greedy algorithm on each sample network. In the reweighted ℓ_1 minimization algorithm, we set $\epsilon = 0.001$, $\delta = 10^{-6}$. We run the algorithm using six different initializations: five random ones and $\mathbf{w}^{(\mathbf{0})} = \mathbf{1}$. Among the six solutions, the one with the smallest number of defaults is selected.

The results are shown in Figs. 1.14, 1.15, and 1.16. The blue and red solid lines represent the average numbers of defaulting nodes after the cash injection allocated by the two algorithms: blue for the reweighted ℓ_1 minimization and red for the greedy algorithm. The dashed lines show the error bars for the estimates of the average. Each error bar is $\pm$two standard errors.

From Fig. 1.14, we see the performance of the reweighted ℓ_1 algorithm is close to the greedy algorithm on the random networks. From Fig. 1.15 and Fig. 1.16, we see that on random core-periphery networks, the greedy algorithm performs better than the reweighted ℓ_1 algorithm, while on random core-periphery networks with chains, the reweighted ℓ_1 algorithm is better.

1.7 Problem III under the Proportional Payment Mechanism

We now investigate Problem III which is a combination of Problem I and Problem II. Instead of just minimizing the weighted sum of unpaid liabilities or the number of defaulting nodes, we consider an objective function which is a linear combination

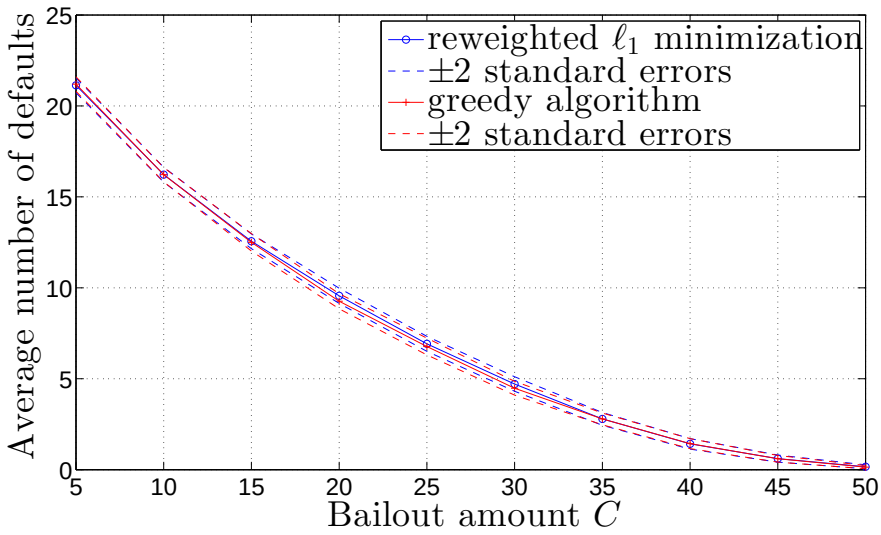


Fig. 1.14. Two heuristic algorithms for minimizing the number of defaults: evaluation on random networks.

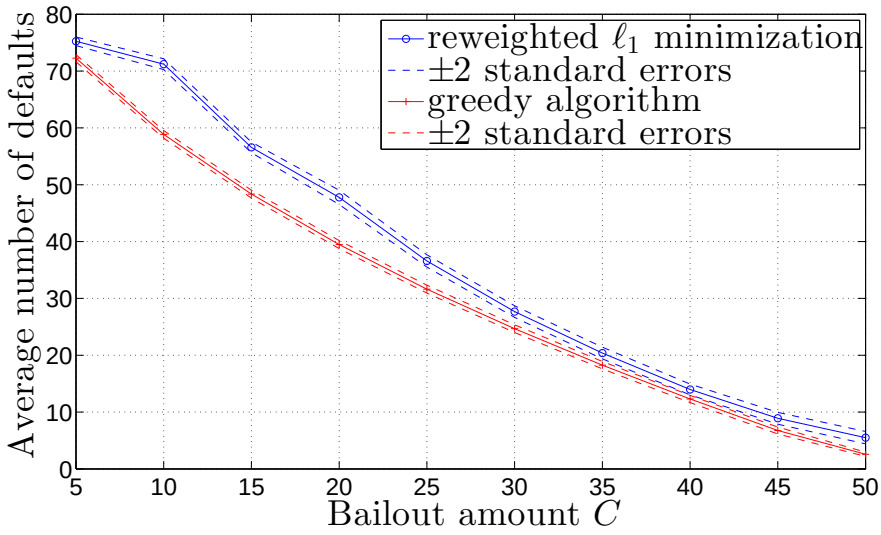


Fig. 1.15. Two heuristic algorithms for minimizing the number of defaults: evaluation on random core-periphery networks.

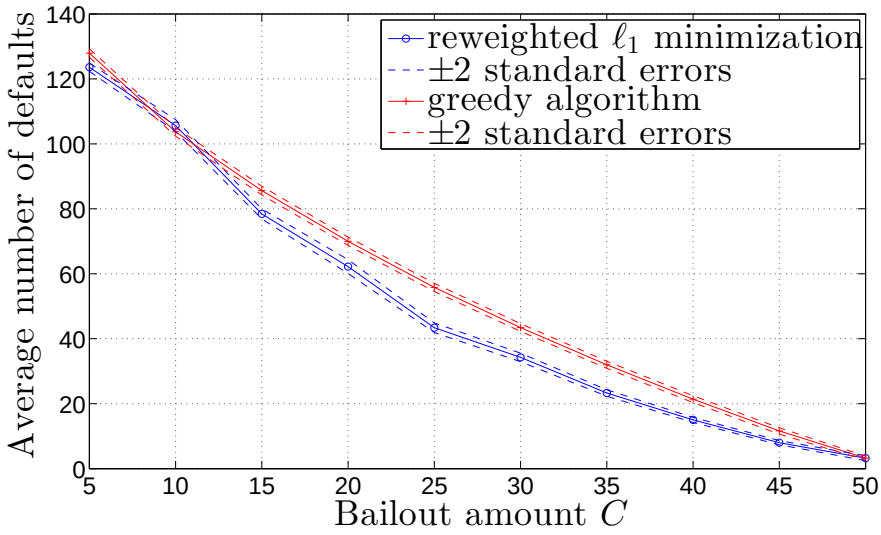


Fig. 1.16. Two heuristic algorithms for minimizing the number of defaults: evaluation on random core-periphery networks with long chains.

of the sum of weights over the defaulting nodes and the weighted sum of unpaid liabilities:

$$D = \mathbf{w}^T(\bar{\mathbf{p}} - \mathbf{p}) + \mathbf{v}^T\mathbf{d}.$$

As defined in Table 1.1, d_i is a binary variable indicating whether node i defaults: $d_i = 1$ if $\bar{p}_i - p_i > 0$ and $d_i = 0$ if $\bar{p}_i - p_i = 0$. v_i is the weight of node i's default.

Since D is strictly decreasing with respect to $\mathbf{p}$, Lemma 4 in [1] implies that minimizing D will yield a clearing payment vector. In light of this fact, we prove that minimizing D subject to a fixed injected cash amount C is equivalent to a mixed-integer linear program.

Theorem 4 *Assume that the liabilities matrix L, the external asset vector $\mathbf{e}$, the weight vectors $\mathbf{w} > \mathbf{0}$ and $\mathbf{v} > \mathbf{0}$ and the total cash injection amount C are fixed and known. Assume that the system utilizes the proportional payment mechanism with no bankruptcy costs. Define $\mathbf{d}$ as in Table 1.1. Then the optimal cash allocation policy to minimize the cost function $D = \mathbf{w}^T(\bar{\mathbf{p}} - \mathbf{p}) + \mathbf{v}^T\mathbf{d}$ can be obtained by solving the following mixed-integer linear program:*

$$\max_{\mathbf{p},\mathbf{c},\mathbf{d}} \mathbf{w}^T\mathbf{p} - \mathbf{v}^T\mathbf{d} \tag{1.41}$$

subject to

$$\mathbf{1}^T\mathbf{c} \le C, \tag{1.42}$$

$$\mathbf{c} \ge \mathbf{0}, \tag{1.43}$$

$$\mathbf{0} \le \mathbf{p} \le \bar{\mathbf{p}}, \tag{1.44}$$

$$\mathbf{p} \le \Pi^T\mathbf{p} + \mathbf{e} + \mathbf{c}, \tag{1.45}$$

$$\bar{p}_i - p_i \le \bar{p}_i d_i, \text{ for } i = 1, 2, \cdots, N, \tag{1.46}$$

$$d_i \in \{0, 1\}, \text{ for } i = 1, 2, \cdots, N. \tag{1.47}$$

Proof Let ($\mathbf{p}^*$, $\mathbf{c}^*$, $\mathbf{d}^*$) be a solution of the mixed-integer linear program (1.41–1.47). We first show that $\mathbf{p}^*$ is a clearing payment vector, i.e., that for each i, we have $p_i^* = \bar{p}_i$ or $p_i^* = \sum_{j=1}^{N} \Pi_{ji} p_j^* + e_i + c_i$. Assume that this is not the case for some

node k, i.e., that $p_k^* < \bar{p}_k$ and $p_k^* < \sum_{j=1}^{N} \Pi_{jk} p_j^* + e_k + c_k$. We construct a vector $\mathbf{p}^\xi$ which is equal to $\mathbf{p}^*$ in all components except the k-th component. We set the k-th component of $\mathbf{p}^\xi$ to be $p_k^\xi = p_k^* + \xi$, where $\xi > 0$ is small enough to ensure that $p_k^\xi < \bar{p}_k$ and $p_k^\xi < \sum_{j=1}^{N} \Pi_{jk} p_j^\xi + e_k + c_k$. Since Π is a matrix with non-negative entries, for any $i \neq k$, we have:

$$p_i^\xi = p_i^* < \sum_{j=1}^{N} \Pi_{ji} p_j^* + e_i + c_i < \sum_{j=1}^{N} \Pi_{ji} p_j^\xi + e_i + c_i.$$

In addition, $\bar{p}_k - p_k^\xi < \bar{p}_k - p_k^* \leq \bar{p}_k d_k$. Thus, ($\mathbf{p}^\xi$, $\mathbf{c}^*$, $\mathbf{d}^*$) is also in the feasible region of (1.41–1.47) and achieves a larger value of the objective function than ($\mathbf{p}^*$, $\mathbf{c}^*$, $\mathbf{d}^*$). This contradicts the fact that ($\mathbf{p}^*$, $\mathbf{c}^*$, $\mathbf{d}^*$) is a solution of (1.41–1.47). Hence, $\mathbf{p}^*$ is a clearing payment vector.

Second, we show that d_i^* indicates whether node i defaults, i.e., $d_i^* = \mathbb{I}_{\bar{p}_i - p_i^* > 0}$. If $\bar{p}_i - p_i^* > 0$, then $d_i^* = 1$ due to constraints (1.46) and (1.47). If $\bar{p}_i - p_i^* = 0$, then constraint (1.46) is always true. In this case the fact that $v_i > 0$ implies that, in order to maximize the objective function, d_i^* must be zero. Thus, $d_i^* = \mathbb{I}_{\bar{p}_i - p_i^* > 0}$.

So far, we have proved that $\mathbf{p}^*$ and $\mathbf{d}^*$ are the clearing payment vector and default indicator vector, respectively, for cash injection vector $\mathbf{c}^*$. We now prove by contradiction that $\mathbf{c}^*$ is the optimal cash injection allocation. Assume $\mathbf{c}' \neq \mathbf{c}^*$ leads to a strictly smaller value of the cost function D than does $\mathbf{c}^*$. In other words, suppose that $\mathbf{c}'$ satisfies the constraints (1.42) and (1.43), and that the corresponding clearing payment vector $\mathbf{p}'$ and default indicator vector $\mathbf{d}'$ satisfy $\mathbf{w}^T(\bar{\mathbf{p}} - \mathbf{p}') + \mathbf{v}^T\mathbf{d}' < \mathbf{w}^T(\bar{\mathbf{p}} - \mathbf{p}^*) + \mathbf{v}^T\mathbf{d}^*$, which is equivalent to:

$$\mathbf{w}^T\mathbf{p}' - \mathbf{v}^T\mathbf{d}' > \mathbf{w}^T\mathbf{p}^* - \mathbf{v}^T\mathbf{d}^*.$$

Since $\mathbf{p}'$ is the corresponding clearing payment vector, constraint (1.44) and (1.45) are satisfied. Moreover, $\mathbf{d}'$ is the corresponding default indicator vector satisfying constraint (1.46) and (1.47) for $\mathbf{c}'$. So ($\mathbf{p}'$,$\mathbf{c}'$,$\mathbf{d}'$) is in the feasible region of (1.41–1.47) and achieves a larger objective function than ($\mathbf{p}^*$, $\mathbf{c}^*$, $\mathbf{d}^*$), which contradicts the fact that ($\mathbf{p}^*$, $\mathbf{c}^*$, $\mathbf{d}^*$) is the solution of (1.41–1.47). ■

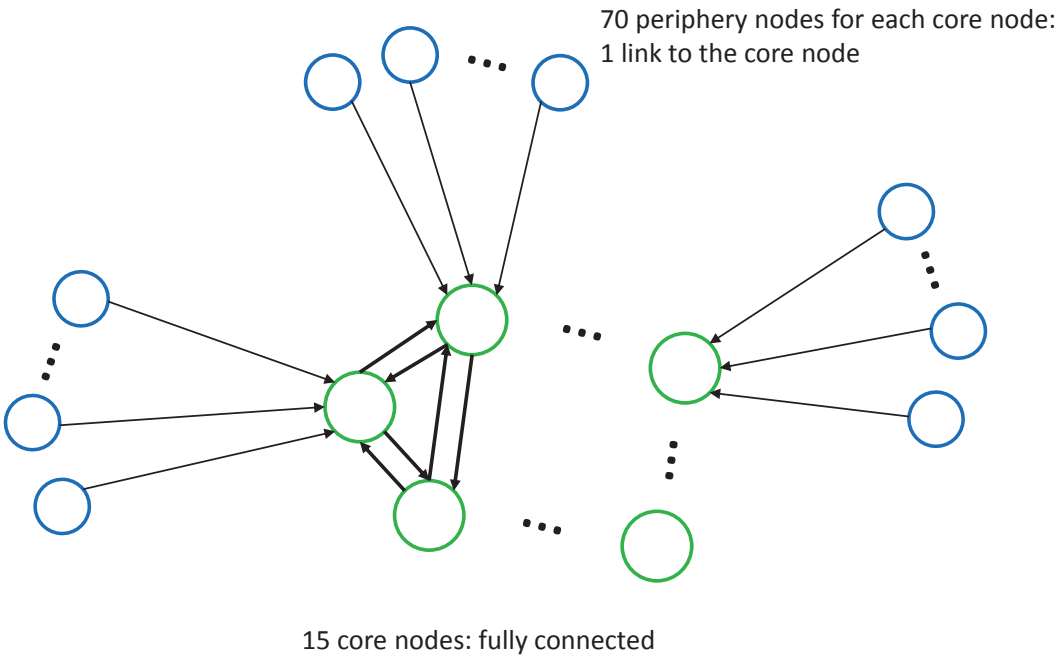


Fig. 1.17. A core-periphery network.

1.7.1 Numerical Simulations

We use CVX with solver *Gurobi* [47] to solve MILP (1.41). We test the running time on the core-periphery network shown in Fig. 1.17, which contains 15 fully connected core nodes and 70 periphery nodes for each of them. Each periphery node has a single link pointing to the corresponding core node. Every node has zero external assets: $\mathbf{e} = \mathbf{0}$. All the obligation amounts $L_{i,j}$ are independent uniform random variables. For each pair of core nodes i and j the obligation amount L_{ij} is uniformly distributed in $[0, 10]$. For a core node i and its periphery node k, the obligation amount L_{ki} is uniformly distributed in $[0, 1]$. The weights of defaults are 1 for all core nodes, and are 0.1 for all periphery nodes. The weights of unpaid liabilities are uniform in $[0, 0.1]$ for all nodes. The regulator has \$300 to be injected into the network.

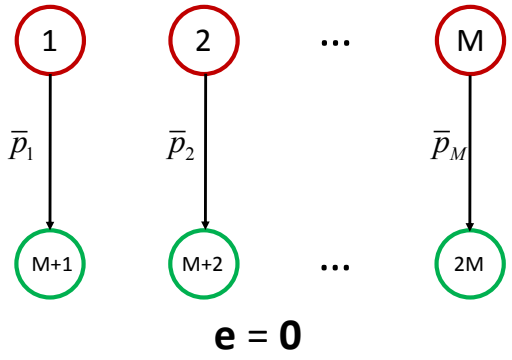


Fig. 1.18. Financial network used in the Proof of Theorem 5. For this network, Problem I under the all-or-nothing payment mechanism is a knapsack problem.

For this core-periphery network, we generate 100 samples. We run the CVX code on a personal computer with a 2.66GHz Intel Core2 Duo Processor P8800. The average running time is $0.90s$ and the sample standard deviation is $0.55s$. The relative gap between the objective of the solution and the optimal objective is less than 5×10^{-3}. (This bound is obtained by calculating the optimal value of the objective for the corresponding linear program, which is an upper bound for the optimal objective value of the MILP.) MILP (1.41) can be solved by CVX efficiently and accurately. The CVX code is given in Appendix B.3.

1.8 All-or-Nothing Payment Mechanism

We now show that under the all-or-nothing payment mechanism, Problem I is NP-hard. Despite this fact, we show through simulations that for network sizes comparable to the size of the US banking system, this problem can be solved in a few seconds on a personal computer using modern optimization software.

Theorem 5 *With the all-or-nothing payment mechanism, Problem I is NP-hard since the knapsack problem is reducible to Problem I.*

Proof Consider the network depicted in Fig. 1.18. The network has $N = 2M$ nodes where M is a positive integer. We let $L_{i,M+i} = \bar{p}_i$ for $i = 1, 2, \cdots, M$; for all other pairs (i, j), we set $L_{ij} = 0$. We set the external asset vector to zero: $\mathbf{e} = \mathbf{0}$. We set all the weights to 1: $\mathbf{w} = \mathbf{1}$. We let x_i be the rescue indicator variable for node i, i.e., $x_i = 0$ if i is in default and $x_i = 1$ if i is fully rescued, for $i = 1, \cdots, M$.

Note that under the all-or-nothing payment mechanism, fully rescuing node i for any $i = 1, \cdots, M$ in Fig. 1.18 means injecting $c_i = \bar{p}_i$. On the other hand, injecting any other nonzero amount $c_i < \bar{p}_i$ is wasteful, as it does not reduce the total amount of unpaid liabilities in the system. Therefore, for each defaulting node i we have $x_i = 0$, $c_i = 0$, and $p_i = 0$, and for each rescued node i we have $x_i = 1$, $c_i = \bar{p}_i$, and $p_i = \bar{p}_i$. The reduction in the total amount of unpaid obligations due to the cash injection is

$$\sum_{i=1}^{M} x_i \bar{p}_i.$$

We must select $\mathbf{x}$ to maximize this amount, subject to the budget constraint $\sum_{i=1}^{M} x_i \bar{p}_i \leq C$ that says that the total amount of cash injection spent on fully rescued nodes must not exceed C:

$$\begin{aligned} &\max_{\mathbf{x}} \sum_{i=1}^{M} x_i \bar{p}_i \qquad (1.48)\\ &\text{subject to}\\ &\sum_{i=1}^{M} x_i \bar{p}_i \leq C,\\ &x_i \in \{0, 1\}, \text{ for } i = 1, 2, \cdots, M. \end{aligned}$$

If any cash remains, it can be arbitrarily allocated among the remaining nodes or not spent at all, because partially rescuing a node does not lead to any improvement

of the objective function. Program (1.48) is a *knapsack problem*, a well-known NP-hard problem. Thus, Problem I under the all-or-nothing payment mechanism is an NP-hard problem. ■

We now establish a mixed-integer linear program to solve Problem I with the all-or-nothing payment mechanism.

Theorem 6 *Assume that the liabilities matrix L, the external asset vector $\mathbf{e}$, the weight vector $\mathbf{w} > \mathbf{0}$ and the total cash injection amount C are fixed and known. Assume the all-or-nothing payment mechanism. Then Problem I is equivalent to the following mixed-integer linear program:*

$$\max_{\mathbf{p},\mathbf{c},\mathbf{d}} \mathbf{w}^T\mathbf{p} \tag{1.49}$$

subject to

$$\mathbf{1}^T\mathbf{c} \leq C, \tag{1.50}$$

$$\mathbf{c} \geq \mathbf{0}, \tag{1.51}$$

$$p_i = \bar{p}_i(1 - d_i), \text{ for } i = 1, 2, \cdots, N, \tag{1.52}$$

$$\bar{p}_i - \sum_{j=1}^{N} \Pi_{ji} p_j - e_i - c_i \leq \bar{p}_i d_i, \text{ for } i = 1, 2, \cdots, N, \tag{1.53}$$

$$d_i \in \{0, 1\}, \text{ for } i = 1, 2, \cdots, N. \tag{1.54}$$

Proof Let $(\mathbf{p}^*, \mathbf{c}^*, \mathbf{d}^*)$ be a solution of the mixed-integer linear program (1.49–1.54). We first show that $\mathbf{p}^*$ is the clearing payment vector corresponding to $\mathbf{c}^*$. For node i, if $\bar{p}_i > \sum_{j=1}^{N} \Pi_{ji} p_j^* + e_i + c_i$, then from constraints (1.53) and (1.54) it follows that $d_i^* = 1$ so that $p_i^* = 0$. If $\bar{p}_i \leq \sum_{j=1}^{N} \Pi_{ji} p_j^* + e_i + c_i$, then constraint (1.53) is satisfied for both $d_i = 0$ and $d_i = 1$. In this case, in order to maximize the objective function, it must be that $d^* = 0$ and $p_i^* = \bar{p}_i$. This completes the Proof that $\mathbf{p}^*$ is the clearing payment vector corresponding to $\mathbf{c}^*$ under the all-or-nothing payment mechanism.

Second, we prove by contradiction that $\mathbf{c}^*$ is the optimal allocation. Assume that $\mathbf{c}'$ leads to a smaller weighted sum of unpaid liabilities, or equivalently, a larger value

of $\mathbf{w}^T\mathbf{p}'$, where $\mathbf{p}'$ is the clearing payment vector corresponding to $\mathbf{c}'$. Since $\mathbf{p}'$ is a clearing payment vector, we have that if $\bar{p}_i > \sum_{j=1}^{N} \Pi_{ji} p'_j + e_i + c'_i$ then $p'_i = 0$; and if $\bar{p}_i \leq \sum_{j=1}^{N} \Pi_{ji} p'_j + e_i + c'_i$ then $p'_i = \bar{p}_i$. We define vector $\mathbf{d}'$ as $d'_i = 0$ for $p'_i = \bar{p}_i$ and $d'_i = 1$ otherwise. Then $(\mathbf{p}', \mathbf{c}', \mathbf{d}')$ is located in the feasible region of MILP (1.49–1.54) but leads to a larger value of the objective function than $(\mathbf{p}^*, \mathbf{c}^*, \mathbf{d}^*)$. This contradicts the fact that $(\mathbf{p}^*, \mathbf{c}^*, \mathbf{d}^*)$ is a solution of (1.49–1.54). ■

Under the all-or-nothing payment mechanism, Problem I with multiple seniorities is equivalent to the one with single seniority because an institution either pays off all the liabilities at different seniorities or none of them.

Now we consider Problem II and Problem III under the all-or-nothing payment mechanism. The objective that we aim to maximize in Problem II is the number of non-defaulting nodes. If node i defaults, it repays zero liabilities and contributes zero to the objective; if node i does not default, it pays off $\bar{p}_i$ and contributes one to the objective, which means that the weight for node i in the objective is $1/\bar{p}_i$. Therefore, under the all-or-nothing payment mechanism, Problem II is a special case for Problem I when we set the weight $w_i = 1/\bar{p}_i$ for nonzero $\bar{p}_i$ and $w_i = 0$ for zero $\bar{p}_i$ in MILP (1.49). Similarly, Problem III can also be transformed to Problem I by setting $w_i = w'_i + v_i/\bar{p}_i$ for nonzero $\bar{p}_i$ and $w_i = w'_i$ for zero $\bar{p}_i$ in MILP (1.49), where w'_i is the original weight for node i's unpaid liabilities in Problem III and v_i is the weight of node i's default.

1.8.1 Numerical Simulations

To solve MILP (1.49), we use CVX, a package for specifying and solving convex programs and also MILPs [2, 3]. In CVX, we select *Mosek* to be the solver [46]. We test the running time on the core-periphery network shown in Fig. 1.17, which contains 15 fully connected core nodes and 70 periphery nodes for each of them. Each periphery node has a single link pointing to the corresponding core node. Every node

has zero external assets: $\mathbf{e} = \mathbf{0}$. All the obligation amounts $L_{i,j}$ are independent uniform random variables. For each pair of core nodes i and j the obligation amount L_{ij} is uniformly distributed in $[0, 10]$. For a core node i and its periphery node k, the obligation amount L_{ki} is uniformly distributed in $[0, 1]$. For a core node i, we set the weight $w_i = 10$; for a periphery node k, we set the weight $w_k = 1$. The regulator has \$300 to be injected into the network. For this core-periphery network, we generate 100 samples. We run the CVX code on a personal computer with a 2.66GHz Intel Core2 Duo Processor P8800. The average running time is $1.9s$ and the sample standard deviation is $2.0s$. The relative gap between the objective of the solution and the optimal objective is less than 10^{-4}. (This bound is obtained by calculating the optimal value of the objective for the corresponding linear program, which is an upper bound for the optimal objective value of the MILP.) If the optimal value of the objective is in the millions of dollars, a relative error of 10^{-4} means that we are within a few hundred dollars away from that optimum, which is a reasonable precision. MILP (1.49) can be solved by CVX efficiently and accurately. The CVX code is given in Appendix B.4.

1.9 Conclusions

In this chapter, we have developed a linear program to obtain the optimal cash injection policy to minimize the weighted sum of unpaid liabilities in a basic model with one period and a single seniority. We also incorporate multiple seniorities and CDS into the basic model so that it could be adapted to more realistic financial systems. With such extensions, the linear program turns into mixed-integer linear programs. We have further proposed a reweighted ℓ_1 minimization algorithm based on this linear program and a greedy algorithm to find the cash injection allocation strategy which minimizes the number of defaults in the system. By constructing three topologies in which the optimal solution can be calculated directly, we have tested both algorithms and shown through simulation that the results of the reweighted

ℓ_1 minimization algorithm are close to optimal, and that the performance of the greedy algorithm highly depends on the network topology. We also compare these two algorithms using three types of random networks for which the optimal solution is not available. In addition, we have shown that the introduction of the all-or-nothing payment mechanism turns the optimal cash injection allocation problem into an NP-hard mixed-integer linear program. We have shown through simulations that this problem can be accurately solved in a few seconds for a network size comparable to the size of the US banking network. Our results provide algorithmic tools to help financial institutions, banking supervisory authorities, regulatory agencies, and clearing houses in monitoring and mitigating systemic risk in financial networks.

2. A DISTRIBUTED ALGORITHM FOR SYSTEMIC RISK MITIGATION IN FINANCIAL SYSTEMS

2.1 Introduction

In Chapter 1, we showed in Theorem 1 that Problem I without bankruptcy costs is equivalent to a linear program, and therefore can be solved exactly, for any network topology, using standard LP solvers. In some scenarios, however, this approach may be impractical or undesirable, as it requires the solver to know the entire network structure, namely, the net external assets of every institution, as well as the amounts owed by each institution to each other institution. To collect and effectively use all this information in a centralized fashion would impose a prohibitive regulatory burden both on the financial institutions and on the regulators themselves.

In this chapter, we adapt our framework to applications where it is necessary to avoid centralized data gathering and computation. We propose a distributed algorithm to solve our linear program. The algorithm is iterative and is based on message passing between each node and its neighbors. During each iteration of the algorithm, each node only needs to receive a small amount of data from its neighbors, perform simple calculations, and transmit a small amount of data to its neighbors. During the message passing, no node will reveal to any other node any proprietary information on its asset values, the amounts owed to other nodes, or the amounts owed by other nodes.

We verify the convergence of the distributed algorithm on a four-node network and estimate the practical time to converge. While the algorithm is slower than standard centralized LP solvers, simulations suggest its practicality for the US banking system which we model as a core-periphery network with 15 core nodes and 1050 periphery nodes.

Our algorithm can be used both to monitor financial networks and to simulate stress-testing scenarios. The integrity of the process can be enforced by the supervisory authorities through auditing.

This chapter is organized as follows. A duality-based distributed algorithm for Problem I under the proportional payment mechanism is proposed in Section 2.2. In Section 2.3, we further extend the distributed algorithm for the alternative formulation of Problem I. We illustrate the practicality of the distributed algorithm for the network with the same size as the US banking system by simulations in Section 2.4 and conclude in Section 2.5.

2.2 Problem I under the Proportional Payment Mechanism

2.2.1 A Distributed Algorithm

In this chapter, we follow the same notations defined in Section 1.2 of Chapter 1. To develop a distributed algorithm for LP (1.4-1.8), we formulate its dual problem and solve it via gradient descent. We solve the dual problem because unlike the primal, it has simpler constraints which are easily decomposable. It turns out that every iteration of the gradient descent involves only local computations, which enables a distributed implementation.

In order to apply the gradient descent method to the dual problem, we need the objective function in (1.4) to be strictly concave, which would guarantee that the dual problem is differentiable at any point [50]. However, the objective function of LP (1.4) is not strictly concave so we apply the Proximal Optimization Algorithm [51,52]. The basic idea is to make the objective function strictly concave by adding quadratic terms that converge to zero at the optimal, so as to avoid changing the solution. A popular choice for these quadratic terms are the residuals from equality constraints.

We introduce two $N \times 1$ vectors $\mathbf{y}$ and $\mathbf{z}$ and add two quadratic terms

$$\|\mathbf{p}-\mathbf{y}\|^2 = \sum_{i=1}^{N}(p_i - y_i)^2, \qquad \|\mathbf{c}-\mathbf{z}\|^2 = \sum_{i=1}^{N}(c_i - z_i)^2$$

to (1.4). Then we proceed as follows.

Algorithm $\mathcal{P}$:

At the t-th iteration,

- **P1)** Fix $\mathbf{y} = \mathbf{y}(t)$ and $\mathbf{z} = \mathbf{z}(t)$ and maximize the objective function with respect to $\mathbf{p}$ and $\mathbf{c}$:

$$\max_{\mathbf{p},\mathbf{c}} \mathbf{w}^T\mathbf{p} - \|\mathbf{p}-\mathbf{y}\|^2 - \|\mathbf{c}-\mathbf{z}\|^2 \tag{2.1}$$

subject to

$$\mathbf{1}^T\mathbf{c} \le C, \tag{2.2}$$

$$\mathbf{c} \ge \mathbf{0},$$

$$\mathbf{0} \le \mathbf{p} \le \bar{\mathbf{p}},$$

$$\mathbf{p} \le \Pi^T\mathbf{p} + \mathbf{e} + \mathbf{c}. \tag{2.3}$$

  Note that since the objective function is strictly concave, a unique solution exists. Denote it as $\mathbf{p}^*$ and $\mathbf{c}^*$.

- **P2)** Set $\mathbf{y}(t+1) = \mathbf{p}^*$, $\mathbf{z}(t+1) = \mathbf{c}^*$.

It is proved in Proposition 4.1 in [52] that algorithm $\mathcal{P}$ will converge to the optimal solution of LP (1.4-1.8).

2.2.2 Implementation of Algorithm $\mathcal{P}$

In Step **P1**, for fixed $\mathbf{y}$ and $\mathbf{z}$, the objective function of (2.1) is strictly concave so that the dual problem is differentiable at any point [50]. Hence, we can solve the dual problem using gradient descent.

Let a scalar λ and an $N \times 1$ vector $\mathbf{q}$ be Lagrange multipliers for constraints (2.2) and (2.3), respectively. We define the Lagrangian as follows:

$$L(\mathbf{p},\mathbf{c},\lambda,\mathbf{q},\mathbf{y},\mathbf{z}) = \mathbf{w}^T\mathbf{p} - \lambda(\mathbf{1}^T\mathbf{c} - C) - \mathbf{q}^T((I_N - \Pi^T)\mathbf{p} - \mathbf{e} - \mathbf{c}) - \|\mathbf{p}-\mathbf{y}\|^2 - \|\mathbf{c}-\mathbf{z}\|^2, \tag{2.4}$$

where λ and $\mathbf{q}$ are non-negative and I_N is an $N \times N$ identity matrix. We further expand (2.4):

$$\begin{aligned} L(\mathbf{p},\mathbf{c},\lambda,\mathbf{q},\mathbf{y},\mathbf{z}) &= \sum_{i=1}^{N} w_i p_i - \lambda \sum_{i=1}^{N} c_i + \lambda C - \sum_{i=1}^{N} q_i (p_i - \sum_{j=1}^{N} \Pi_{ji} p_j - e_i - c_i) \\ &\quad - \sum_{i=1}^{N} (p_i - y_i)^2 - \sum_{i=1}^{N} (c_i - z_i)^2 \qquad (2.5) \\ &= -\sum_{i=1}^{N} \left[p_i^2 - (w_i - q_i + 2y_i + \sum_{j=1}^{N} q_j \Pi_{ij}) p_i \right] \\ &\quad - \sum_{i=1}^{N} \left[c_i^2 - (q_i - \lambda + 2z_i) c_i \right] + \sum_{i=1}^{N} \left[q_i e_i - y_i^2 - z_i^2 \right] + \lambda C. \qquad (2.6) \end{aligned}$$

To obtain Eq. (2.6) from Eq. (2.5), we use the following equation: $\sum_{i=1}^{N}\sum_{j=1}^{N} q_i \Pi_{ji} p_j = \sum_{i=1}^{N}\sum_{j=1}^{N} q_j \Pi_{ij} p_i$. Then the objective function of the dual problem is:

$$D(\lambda,\mathbf{q},\mathbf{y},\mathbf{z}) = \max_{\mathbf{0}\le\mathbf{p}\le\bar{\mathbf{p}},\mathbf{c}\ge\mathbf{0}} L(\mathbf{p},\mathbf{c},\lambda,\mathbf{q},\mathbf{y},\mathbf{z}). \qquad (2.7)$$

In Eq. (2.6), the term $q_j \Pi_{ij}$ is 0 if node i is not a borrower of j. Thus, if node i receives all the q_j from its lenders, it can determine p_i and c_i to achieve the maximum of the Lagrangian $L(\mathbf{p},\mathbf{c},\lambda,\mathbf{q},\mathbf{y},\mathbf{z})$.

Given $\mathbf{y}$ and $\mathbf{z}$, the dual problem of (2.1) is then minimizing (2.7) over Lagrange multipliers λ and $\mathbf{q}$:

$$\min_{\lambda\ge 0,\mathbf{q}\ge\mathbf{0}} D(\lambda,\mathbf{q},\mathbf{y},\mathbf{z}). \qquad (2.8)$$

The objective of the dual problem is differentiable at any point since the objective function of the primal is strictly concave [50]. Hence, gradient descent iterations can be applied to solve the dual.

Let $\lambda(u)$ and $\mathbf{q}(u)$ respectively denote the values of λ and $\mathbf{q}$ at iteration u. Then the gradients of D with respect to λ and $\mathbf{q}$ at this point are:

$$\frac{\partial D}{\partial \lambda} = C - \mathbf{1}^T \mathbf{c}(u),$$

$$\frac{\partial D}{\partial \mathbf{q}} = \mathbf{e} + \mathbf{c}(u) - (I_N - \Pi^T)\mathbf{p}(u),$$

where $\mathbf{p}(u)$ and $\mathbf{c}(u)$ solve (2.7) for $\lambda = \lambda(u)$ and $\mathbf{q} = \mathbf{q}(u)$:

$$p_i(u) = \begin{cases} 0 & \text{if } \tilde{p}_i(u) < 0 \\ \bar{p}_i & \text{if } \tilde{p}_i(u) > \bar{p}_i \\ \tilde{p}_i(u) & \text{otherwise} \end{cases} \tag{2.9}$$

where $\tilde{p}_i(u) = y_i(t) + \frac{1}{2}\left(w_i - q_i(u) + \sum_{j \in \mathcal{C}_i} q_j(u)\Pi_{ij}\right)$, and

$$c_i(u) = \left[z_i(t) + \frac{1}{2}\left(q_i(u) - \lambda(u)\right)\right]^+. \tag{2.10}$$

Therefore, taking into account the non-negativity of λ and $\mathbf{q}$, the gradient descent equations are:

$$\lambda(u+1) = \left[\lambda(u) - \alpha\left(C - \sum_{i=1}^{N} c_i(u)\right)\right]^+, \tag{2.11}$$

$$\mathbf{q}(u+1) = \left[\mathbf{q}(u) - \beta\left(\mathbf{e} + \mathbf{c}(u) - \mathbf{p}(u) + \Pi^T\mathbf{p}(u)\right)\right]^+. \tag{2.12}$$

where α and β are the step sizes, and $[x]^+ = \max\{0, x\}$. For fixed $\mathbf{y}$ and $\mathbf{z}$, the dual update will converge to the minimizer of D as $u \to \infty$, if the step size is small enough [52].

From (2.11), we notice that in order to update λ, c_i is required from all the N nodes. It means at each iteration u, each node should send $c_i(u)$ to a central node which updates λ and send it back to every node in the system.

If node j is not a borrower of node i, then $\Pi_{ji}p_j(u) = 0$; otherwise, $\Pi_{ji}p_j(u)$ represents the amount of money that node j pays to node i at u-th iteration. Hence, with the information of $\Pi_{ji}p_j(u)$ from all its borrowers, node i is able to update q_i based on (2.12).

2.2.3 A More Efficient Algorithm

As shown in Fig. 2.1, in the algorithm $\mathcal{P}$ first fixes $\mathbf{y}$ and $\mathbf{z}$ and solve (2.1) by updating λ, $\mathbf{q}$ and $\mathbf{p}$, $\mathbf{c}$ iteratively until they converge. Then we update $\mathbf{y}$ and $\mathbf{z}$. This

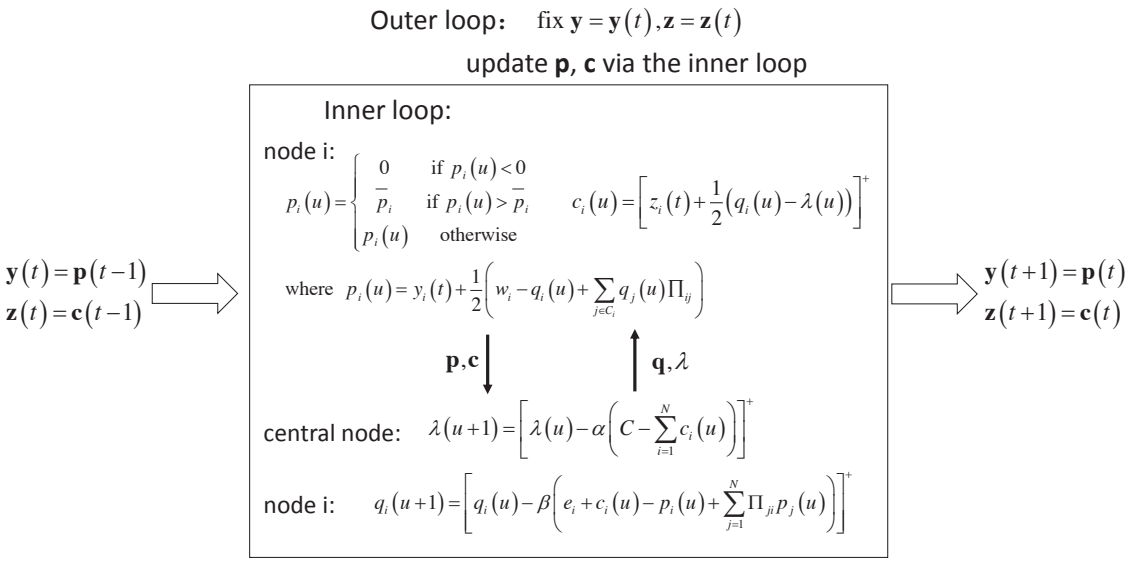


Fig. 2.1. Duality-based approach.

is a two-stage iteration, which is likely to slow down the convergence of the entire algorithm as too many dual updates are wasted for each fixed $\mathbf{y}$ and $\mathbf{z}$ [51]. To avoid the two-stage iteration structure, we consider the following algorithm.

Algorithm $\mathcal{A}$:

At the t-th iteration,

- **A1)** Fix $\mathbf{y} = \mathbf{y}(t)$ and $\mathbf{z} = \mathbf{z}(t)$, maximize L with respect to $\mathbf{p}$ and $\mathbf{c}$,

$$[\mathbf{p}(t), \mathbf{c}(t)] = \arg \max_{\mathbf{0} \le \mathbf{p} \le \bar{\mathbf{p}}, \mathbf{c} \ge \mathbf{0}} L(\mathbf{p}, \mathbf{c}, \lambda(t), \mathbf{q}(t), \mathbf{y}(t), \mathbf{z}(t)).$$

- **A2)** Update Lagrange multipliers $\lambda(t+1)$ and $\mathbf{q}(t+1)$ by

$$\lambda(t+1) = \left[\lambda(t) - \alpha \left(C - \sum_{i=1}^{N} c_i(t) \right) \right]^+, \tag{2.13}$$

$$\mathbf{q}(t+1) = \left[\mathbf{q}(t) - \beta(\mathbf{e} + \mathbf{c}(t) - \mathbf{p}(t) + \Pi^T \mathbf{p}(t)) \right]^+. \tag{2.14}$$

- **A3)** Update $\mathbf{y}$ and $\mathbf{z}$ with

$$[\mathbf{y}(t+1), \mathbf{z}(t+1)] = \arg \max_{\mathbf{0} \le \mathbf{p} \le \bar{\mathbf{p}}, \mathbf{c} \ge \mathbf{0}} L(\mathbf{p}, \mathbf{c}, \lambda(t+1), \mathbf{q}(t+1), \mathbf{y}(t), \mathbf{z}(t)).$$

In algorithm $\mathcal{A}$, instead of an infinite number of dual updates, we only update Lagrange multipliers λ and $\mathbf{q}$ once for each fixed $\mathbf{y}$ and $\mathbf{z}$. The following theorem guarantees the convergence of algorithm $\mathcal{A}$.

Theorem 7 *Algorithm $\mathcal{A}$ will converge to the optimal solution of LP (1.4-1.8) provided the step sizes α and β are sufficiently small.*

Theorem 7 is an extension of Proposition 4 in [53].

2.2.4 Implementation of Algorithm $\mathcal{A}$

Assume $\mathcal{B}_i$ and $\mathcal{C}_i$ are the sets of borrowers and creditors of node i respectively. Then the t-th iteration of algorithm $\mathcal{A}$ is as follows.

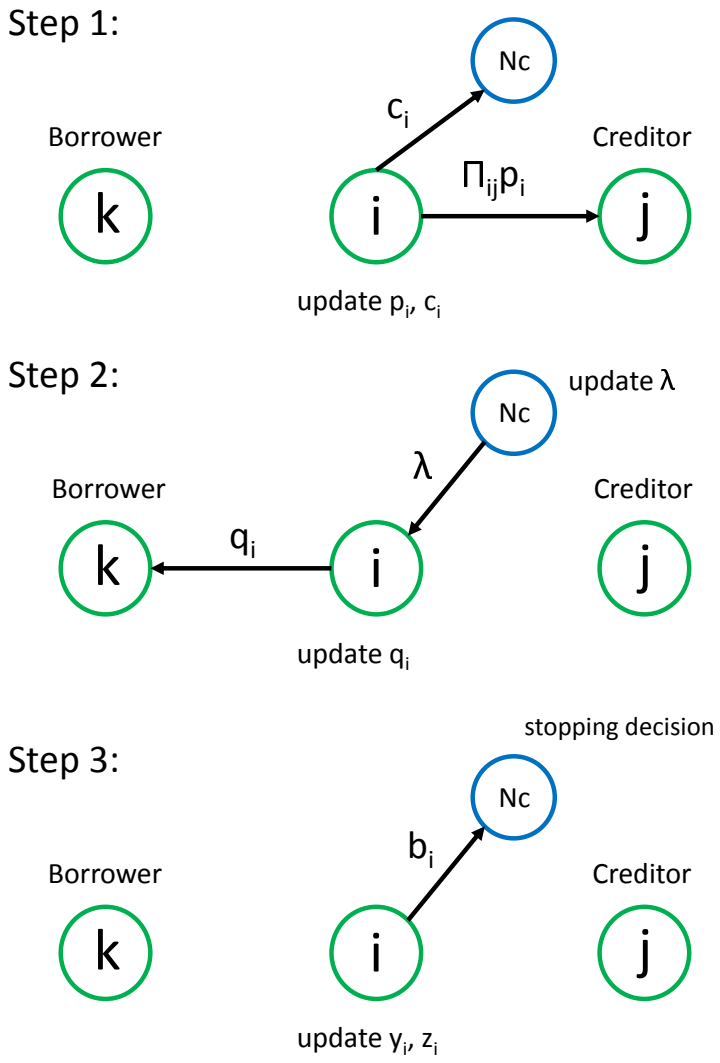


Fig. 2.2. The t-th iteration of the distributed algorithm $\mathcal{A}$ for a fixed maximum total amount of injected cash.

1. For each node i, fix $y_i = y_i(t)$, $z_i = z_i(t)$, $\lambda = \lambda(t)$ and $\mathbf{q} = \mathbf{q}(t)$, and calculate p_i and c_i:

$$p_i(t) = \begin{cases} 0 & \text{if } \tilde{p}_i(t) < 0 \\ \bar{p}_i & \text{if } \tilde{p}_i(t) > \bar{p}_i \\ \tilde{p}_i(t) & \text{otherwise,} \end{cases}$$

where $\tilde{p}_i(t) = y_i(t) + \frac{1}{2}\left(w_i - q_i(t) + \sum_{j \in \mathcal{C}_i} q_j(t)\Pi_{ij}\right)$, and

$$c_i(t) = \left[z_i(t) + \frac{1}{2}(q_i(t) - \lambda(t))\right]^+.$$

Then send $\Pi_{ij}p_i(t)$ to every node $j \in \mathcal{C}_i$, and send the updated $c_i(t)$ to node N_c.

2. Each node i receives $\Pi_{ki}p_k(t)$ from every $k \in \mathcal{B}_i$ and updates q_i:

$$q_i(t+1) = \left[q_i(t) + \beta(p_i(t) - e_i - c_i(t) - \sum_{k \in \mathcal{B}_i} \Pi_{ki}p_k(t))\right]^+.$$

Then each node i sends the updated $q_i(t+1)$ to every node $k \in \mathcal{B}_i$.

Node N_c receives c_i from all nodes i and updates λ:

$$\lambda(t+1) = \left[\lambda(t) + \alpha\left(\sum_{i=1}^{N} c_i(t) - C\right)\right]^+.$$

Then node N_c send the updated $\lambda(t+1)$ to every node i.

3. Every node i receives $q_j(t+1)$ from each $j \in \mathcal{C}_i$ and receives $\lambda(t+1)$ from node N_c, then updates y_i and z_i:

$$y_i(t+1) = \begin{cases} 0 & \text{if } \tilde{y}_i(t+1) < 0 \\ \bar{p}_i & \text{if } \tilde{y}_i(t+1) > \bar{p}_i \\ \tilde{y}_i(t+1) & \text{otherwise} \end{cases}$$

where $\tilde{y}_i(t+1) = y_i(t) + \frac{1}{2}(w_i - q_i(t+1) + \sum_{j \in \mathcal{C}_i} q_j(t+1)\Pi_{ij})$, and

$$z_i(t+1) = [\tilde{z}_i(t+1)]^+,$$

where $\tilde{z}_i(t+1) = z_i(t) + \frac{1}{2}(q_i(t+1) - \lambda(t+1))$.

Every node i then checks the conditions $|\tilde{y}_i(t+1) - \tilde{y}_i(t)| < \delta_1$ and $|\tilde{z}_i(t+1) - \tilde{z}_i(t)| < \delta_2$. If both conditions hold, node i sets $b_i = 1$; otherwise it sets $b_i = 0$. It then sends b_i to the central node N_c. If $b_i = 1$ for all i, then N_c directs all nodes to terminate the algorithm.

These steps are illustrated in Fig. 2.2.

In Step 3, δ_1 and δ_2 are the stopping tolerances, which are usually set as small positive numbers according to the accuracy requirement. We utilize $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{z}}$ rather than their projections $\mathbf{y}$ and $\mathbf{z}$ in the stopping criterion because the convergence of $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{z}}$ implies the convergence of the Lagrange multipliers $\mathbf{q}$ and λ, whereas the convergence of $\mathbf{y}$ and $\mathbf{z}$ does not.

In the implementation of algorithm $\mathcal{A}$, we include a central node. At each iteration the central node has two functions. One is to sum the $c_i(t)$ and calculate $\lambda(t+1)$ in Step 2; the other is to test whether $b_i = 1$ for all nodes i in Step 3. For both functions, the central node only collects a small amount of data and performs simple calculations. We could entirely exclude the central node by calculating the sum of $c_i(t)$ and communicating the stopping sign in a distributed way, at the cost of added computational burden during each iteration.

2.3 The Alternative Formulation of Problem I

We now apply the duality-based distributed algorithm to LP (1.10), the alternative formulation of Problem I. Note that now λ represents the importance of the injected cash amount in the overall cost function. The algorithm is similar to Section 2.2

except for the fact that λ is not updated at each iteration because λ is fixed and given. Similar to (2.4), we define the Lagrangian as:

$$\begin{aligned} L(\mathbf{p},\mathbf{c},\mathbf{q},\mathbf{y},\mathbf{z}) &= \mathbf{w}^T\mathbf{p} - \lambda\mathbf{1}^T\mathbf{c} - \mathbf{q}^T((I_N - \Pi^T)\mathbf{p} - \mathbf{e} - \mathbf{c}) - \|\mathbf{p}-\mathbf{y}\|^2 - \|\mathbf{c}-\mathbf{z}\|^2 \\ &= -\sum_{i=1}^{N}\left[p_i^2 - (w_i - q_i + 2y_i + \sum_{j=1}^{N} q_j\Pi_{ij})p_i\right] \\ &\quad - \sum_{i=1}^{N}\left[c_i^2 - (q_i - \lambda + 2z_i)c_i\right] + \sum_{i=1}^{N}\left[q_ie_i - y_i^2 - z_i^2\right]. \end{aligned} \tag{2.15}$$

The objective function of the dual problem is:

$$D(\mathbf{q},\mathbf{y},\mathbf{z}) = \max_{\mathbf{0}\le\mathbf{p}\le\bar{\mathbf{p}},\mathbf{c}\ge\mathbf{0}} L(\mathbf{p},\mathbf{c},\mathbf{q},\mathbf{y},\mathbf{z})$$

Then the dual problem is:

$$\min_{\mathbf{q}\ge\mathbf{0}} D(\mathbf{q},\mathbf{y},\mathbf{z})$$

The Lagrange multipliers $\mathbf{q}$ are updated by (2.14), where $\mathbf{p}$ and $\mathbf{c}$ maximize Lagrangian (2.15).

2.3.1 Implementation of Algorithm $\mathcal{A}'$

Our algorithm for this problem is a simple modification of Algorithm $\mathcal{A}$. We call it Algorithm $\mathcal{A}'$. Its t-th iteration is as follows.

1. Each node i fixes $y_i = y_i(t)$, $z_i = z_i(t)$, and $\mathbf{q} = \mathbf{q}(t)$, and calculates p_i and c_i:

$$p_i(t) = \begin{cases} 0 & \text{if } \tilde{p}_i(t) < 0 \\ \bar{p}_i & \text{if } \tilde{p}_i(t) > \bar{p}_i \\ \tilde{p}_i(t) & \text{otherwise} \end{cases}$$

where $\tilde{p}_i(t) = y_i(t) + \frac{1}{2}(w_i - q_i(t) + \sum_{j\in\mathcal{C}_i} q_j(t)\Pi_{ij})$, and

$$c_i(t) = \left[z_i(t) + \frac{1}{2}(q_i(t) - \lambda)\right]^+.$$

Then each node i sends $\Pi_{ij}p_i(t)$ to every node $j \in \mathcal{C}_i$.

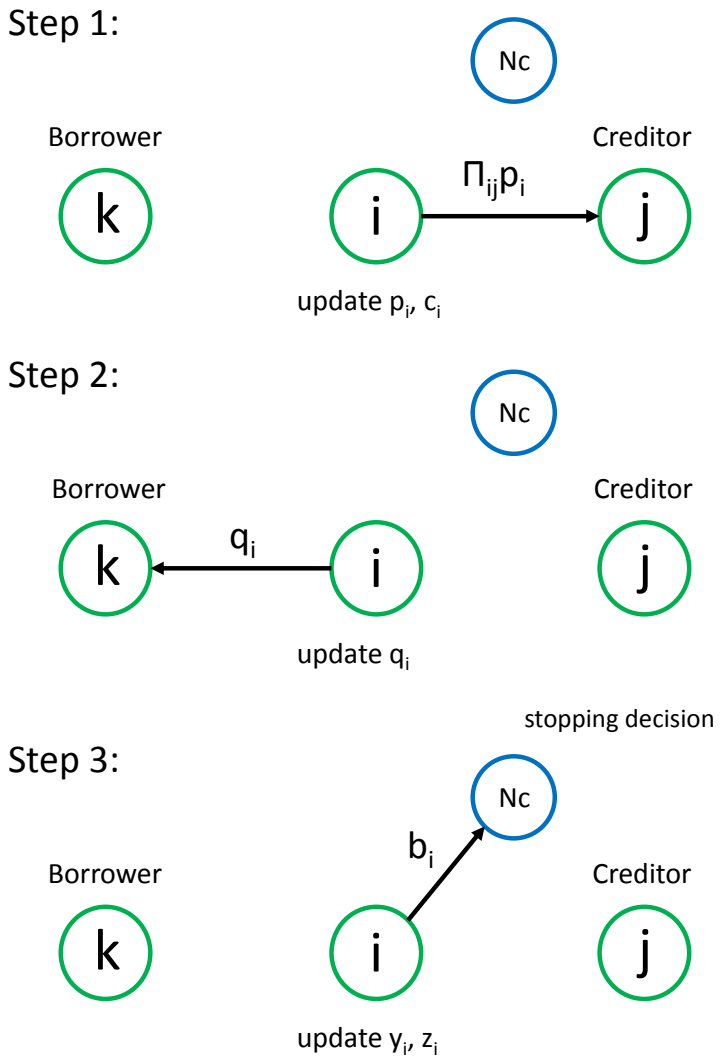


Fig. 2.3. The t-th iteration of the distributed algorithm $\mathcal{A}'$ that includes optimizing the total amount of injected cash.

2. Each node i receives $\Pi_{ki}p_k(t)$ from every $k \in \mathcal{B}_i$ and updates q_i:

$$q_i(t+1) = \left[q_i(t) + \beta(p_i(t) - e_i - c_i(t) - \sum_{k \in \mathcal{B}_i} \Pi_{ki} p_k(t)) \right]^+ .$$

Then each node i sends the updated $q_i(t+1)$ to every $k \in \mathcal{B}_i$.

3. Each node i receives $q_j(t+1)$ from every $j \in \mathcal{C}_i$ and updates y_i and z_i:

$$y_i(t+1) = \begin{cases} 0 & \text{if } \tilde{y}_i(t+1) < 0 \\ \bar{p}_i & \text{if } \tilde{y}_i(t+1) > \bar{p}_i \\ \tilde{y}_i(t+1) & \text{otherwise} \end{cases}$$

where $\tilde{y}_i(t+1) = y_i(t) + \frac{1}{2}(w_i - q_i(t+1) + \sum_{j \in \mathcal{C}_i} q_j(t+1)\Pi_{ij})$, and

$$z_i(t+1) = [\tilde{z}_i(t+1)]^+ ,$$

where $\tilde{z}_i(t+1) = z_i(t) + \frac{1}{2}(q_i(t+1) - \lambda)$.

Each node i checks the conditions $|\tilde{y}_i(t+1) - \tilde{y}_i(t)| < \delta_1$ and $|\tilde{z}_i(t+1) - \tilde{z}_i(t)| < \delta_2$. If both conditions hold, it sets $b_i = 1$; otherwise, it sets $b_i = 0$. It then sends b_i to the central node N_c. If $b_i = 1$ for all i then N_c asks all nodes to terminate the algorithm.

These steps are illustrated in Fig. 2.3. Unlike the one in Algorithm $\mathcal{A}$, the central node does not need to collect c_i at each iteration in Algorithm $\mathcal{A}'$.

2.4 Numerical Results

2.4.1 Example 1: A Four-Node Network

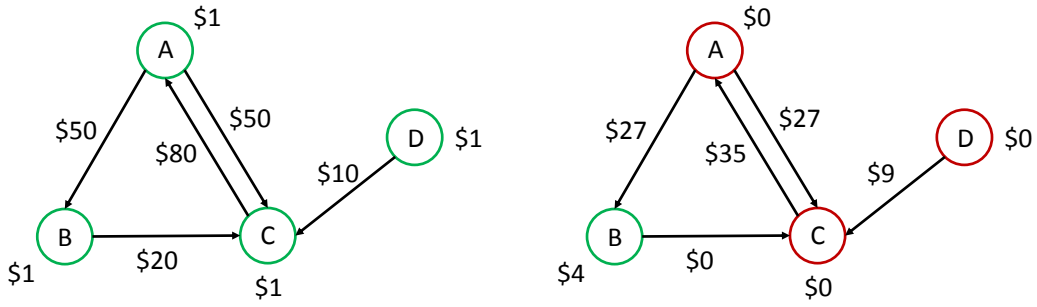
In this section, we illustrate the convergence of our distributed algorithm to the optimal solution. We use the four-node network shown in Fig. 2.4(a). Node A owes \$50 to B and C, node B owes \$20 to C, node C owes \$80 to A, and node D owes \$10 to C. Each node has \$1 on hand. After all the clearing payments, the borrower-lender

network reduces to Fig. 2.4(b). Without any external financial support, nodes A, C, and D are in default, and the total amount of unpaid liability is \$98. Assume that $w_i = 0.45$ for $i = 1, 2, \cdots, N$ in LP (1.4), i.e., that each dollar of unpaid liability contributes 0.45 to the cost. Without any external cash injection, the value of the cost function is $98 \times 0.45 = 44.1$.

We first study Problem I, the case with a fixed maximum total amount of injected cash. We assume that we can inject at most \$15 into the system. We run our algorithm with initial $\mathbf{y}(0) = \mathbf{z}(0) = \mathbf{q}(0) = \mathbf{0}$ and $\lambda = 0$. The step size is $\alpha = \beta = 0.1$, and the stopping tolerance is $\delta_1 = \delta_2 = 10^{-6}$. Figs. 2.6(a) and 2.6(b) illustrate the evolution of payment vector $\mathbf{p}$ and cash injection vector $\mathbf{c}$, respectively, as a function of the number of iterations of the proposed distributed algorithm. The payment vector converges to $[p_A, p_B, p_C, p_D] = [76, 20, 75, 10]$; the cash injection vector converges to $[c_A, c_B, c_C, c_D] = [0, 0, 6.0, 9.0]$. These are optimal, as verified by solving the LP (1.4-1.8) directly. With external cash injection, the borrower-lender network reduces to Fig. 2.5 after all the payments. Now the total unpaid liability is \$29. Thus the value of the cost due to unpaid liability after the optimal bailout is $29 \times 0.45 = 13.05$.

Second, we test our algorithm on the alternative formulation of Problem I. In this example, the initial settings are the same as in the previous example. In addition, we fix the weight $\lambda = 1$. As shown in Fig. 2.8(a) and Fig. 2.8(b), the payment vector converges to $[p_A, p_B, p_C, p_D] = [81, 20, 80, 10]$, and the cash injection vector converges to $[c_A, c_B, c_C, c_D] = [0, 0, 8.5, 9.0]$. These are optimal, as verified by solving the LP (1.10) directly. With external cash injection, the borrower-lender network reduces to Fig. 2.7 after all the payments. Now the total unpaid liability is \$19, and the cash injection amount is \$17.5. Thus the value of the cost function after the optimal bailout is $17.5 + 19 \times 0.45 = 26.05$. By injecting \$17.5, we reduce the total unpaid liability by \$35.55, and we reduce the total cost by $\$35.55 - \$17.5 = \$18.05$.

We see from Fig. 2.7 that although A is still in default, in the optimal bailout strategy we choose not to inject any cash in A. The reason is that if we inject some cash $\$x$ into A in Fig. 2.7, the total unpaid liability will decrease by $\$x$ so that the



(a) Initial network. (b) Clearing without any bailout.

Fig. 2.4. A four-node network.

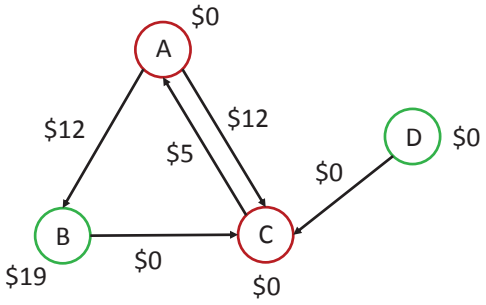
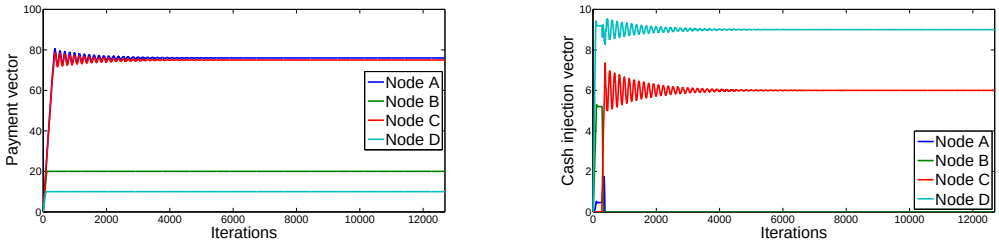


Fig. 2.5. Clearing in the network of Fig. 2.4 for the optimal allocation of a $15 cash injection.



(a) Payment vector. (b) Cash injection vector.

Fig. 2.6. Evolution of the node payments and cash injections through the iterations of the distributed algorithm for finding the optimal allocation of a $15 cash injection into the network of Fig. 2.4.

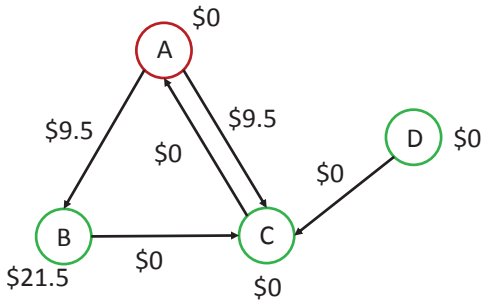
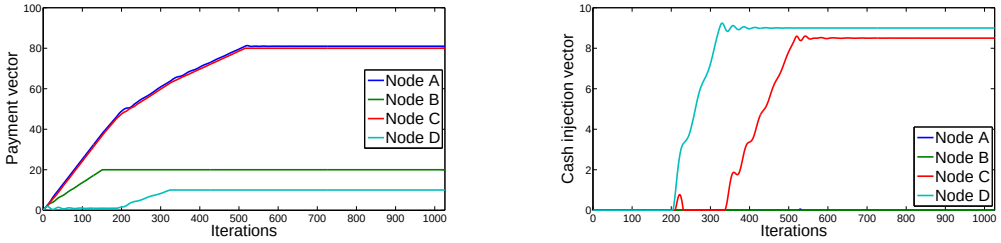


Fig. 2.7. Clearing with the optimal bailout amount and allocation.



(a) Payment vector.

(b) Cash injection vector.

Fig. 2.8. Evolution of the node payments and cash injections through the iterations of the distributed algorithm that optimizes both the amount and the allocation of the injected cash.

unpaid liability term of the cost function will be reduced by $0.45x$, i.e., the value of the overall cost function will actually increase by $x - 0.45x = 0.55x$.

2.4.2 Example 2: A Core-Periphery Network

In this section, we examine the practicality of our distributed algorithm. As in Section 1.8, we assume that the US interbank network is well modeled as a core-periphery network that consists of a core of 15 highly interconnected banks to which most other banks connect [6]. We test the distributed algorithm for LP (1.10) on a simulated core-periphery network illustrated in Fig. 1.17. The core network consists of 15 fully connected core nodes. Each core node has 70 corresponding periphery

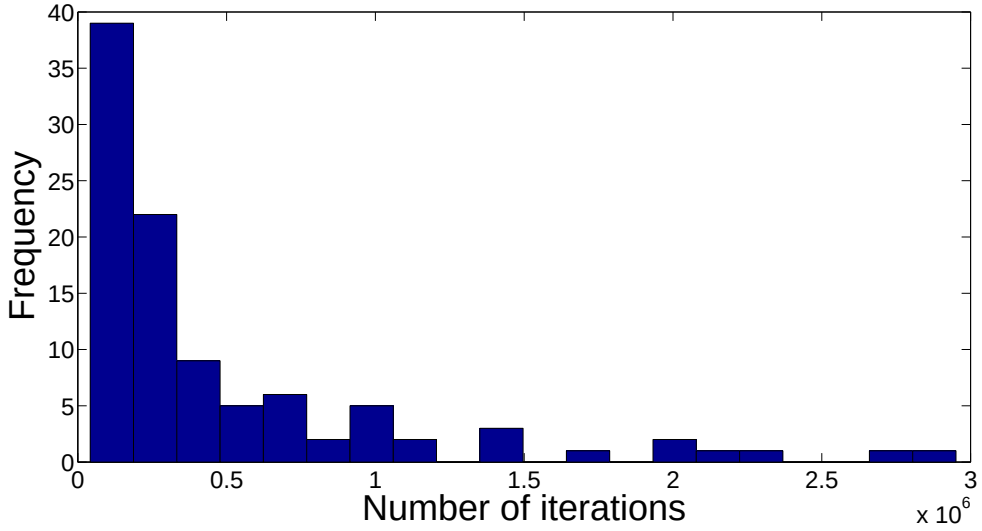


Fig. 2.9. Number of iterations for the core-periphery network with $\delta_1 = \delta_2 = 10^{-7}$.

nodes which owe money only to this core node. For each pair of two core nodes i and j, we set L_{ij} as a random number uniformly distributed in $[0, 10]$. For a core node i and its periphery node k, L_{ki} is set to be uniformly distributed in $[0, 1]$. All these obligation amounts are statistically independent. The asset vector is $\mathbf{e} = \mathbf{0}$. In addition, we assume $w_i = 0.3$ for $i = 1, 2, \cdots, N$, and $\lambda = 1$ in LP (1.10). We generate 100 independent samples of a core-periphery network drawn from this distribution. These samples thus all have the same topology but different amounts of liabilities. We run the distributed algorithm of Section 2.3.1 with initial conditions $\mathbf{y}(0) = \mathbf{z}(0) = \mathbf{q}(0) = \mathbf{0}$. The step size is $\beta = 0.01$.

The stopping criterion for the distributed algorithm is $\max\{\|\tilde{\mathbf{y}}(t+1) - \tilde{\mathbf{y}}(t)\|_\infty, \|\tilde{\mathbf{z}}(t+1) - \tilde{\mathbf{z}}(t)\|_\infty\} < 10^{-7}$. Let T_d be the value of the total cost function $W + \lambda C$ calculated by our distributed algorithm, and let T_l be the corresponding value obtained by solving the linear program directly, in a centralized fashion. Under this stopping criterion, the relative error, defined as $|T_d - T_l|/T_l$, is less than 10^{-6} for each sample in our simulations.

The number of iterations is shown in Fig. 2.9. The average number of iterations is 4.98×10^5. Moreover, from Fig. 2.9, we can see that for most cases, the algorithm terminates within 10^6 iterations.

The time spent on each iteration consists of two parts: the computing time and the time it takes to convey messages between the nodes. During each iteration, a node needs to transmit information to a set of neighbors twice: in Steps 1 and 2. Note that in Step 3, the stopping sign b_i is transmitted to the central node. However, it is not necessary for a node to wait for the response before next iteration. Therefore, we do not count it towards the communication delay during one iteration. It takes light 13.2ms to travel from LA to NYC, which is the longest possible distance between two financial institutions within the continental US. So the propagation delay in one iteration could be roughly estimated as $13.2ms \times 2 = 26.4ms$. Hence, for most cases, the algorithm would terminate within $26.4ms \times 10^6 = 7.3h$, and the average running time would be below $26.4ms \times 4.98 \times 10^5 = 3.65h$. These running times would be acceptable in applications where these computations are run overnight or during a weekend. Note that the computation time at each node is negligible compared to these communication times, and therefore we ignore it in these estimates.

Another possible set-up is that each institution provides a client-end computer and we colocate these computers in one room. Assuming that the longest network cable in this room is 100 meters, the propagation delay per iteration would be around $2 \times 100/(3 \times 10^8) = 6.67 \times 10^{-7}s$. For the computing time, we just analyze the core nodes because the periphery nodes have no borrowers and only one creditor so that the computing time for the periphery nodes is much smaller than for the core nodes. Usually, multiplications dominate the computing time. At each iteration, a core node calculates $q_j(t)\Pi_{ij}$, $\Pi_{ij}p_i(t)$, and $q_j(t+1)\Pi_{ij}$ for all its creditors j. Since the core network is a fully connected network with 15 core nodes, a core node has 14 creditors so that it does less than 50 multiplications per iteration. Assuming that each multiplication takes 500 cpu cycles and the cpu on the client-end computer is 3GHz, then the computing time per iteration is around $50 \times 500/(3 \times 10^9) = 8.33 \times 10^{-6}s$.

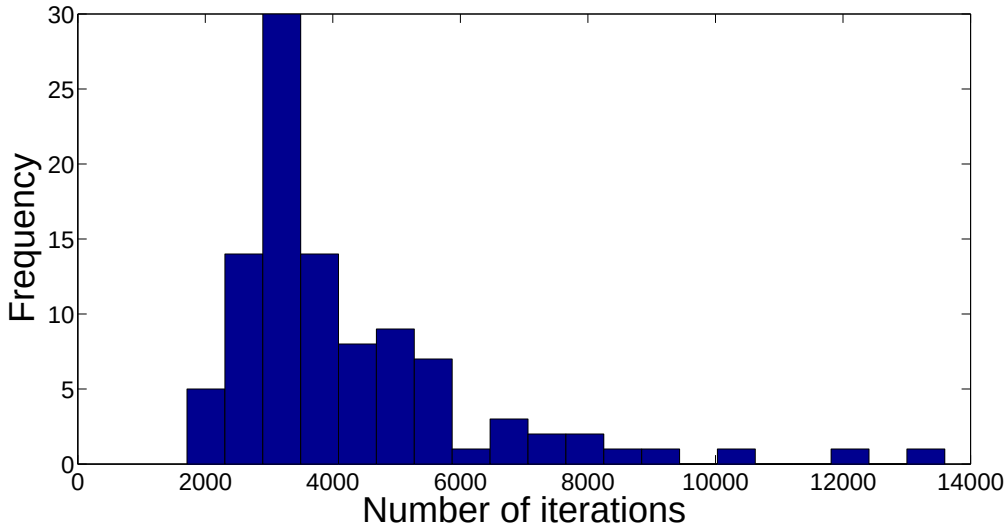


Fig. 2.10. Number of iterations for the core-periphery network with $\delta_1 = \delta_2 = 10^{-3}$.

Thus, for most cases, the algorithm terminates within $(8.33 \times 10^{-6} + 6.67 \times 10^{-7}) \times 10^6 \approx 10s$. By colocating the client-end computers of all the financial institutions in the system, we can significantly reduce the running time of our distributed algorithm so that it can be easily run many times during a day.

In a monitoring application, our aim might be to calculate the payments approximately rather than exactly. In this case, the running time can be reduced by relaxing the termination tolerance. We set the stopping criterion as $\max\{\|\tilde{\mathbf{y}}(t+1) - \tilde{\mathbf{y}}(t)\|_\infty, \|\tilde{\mathbf{z}}(t+1) - \tilde{\mathbf{z}}(t)\|_\infty\} < 10^{-3}$. Under this stopping criterion, the relative error, $|T_d - T_l|/T_l$, is around 1% for each sample in our simulations. Fig. 2.10 illustrates the number of iterations. The average number is 4260. The number of iterations is less than 10000 for most cases. By similar analysis, the average running time for the non-colocated scenario is $26.4ms \times 4260 \approx 2min$. For most cases, the algorithm will be terminated within $26.4ms \times 10^4 \approx 4.4min$. If we colocate the client-end computers of all the financial institutions, the algorithm will terminate within $(8.33 \times 10^{-6} + 6.67 \times 10^{-7}) \times 10^4 \approx 0.1s$.

The above running time analysis is for the alternative formulation of Problem I, in which λ is a constant. In Problem I, λ is a dual variable that also needs to converge. So the running time of the distributed algorithm for Problem I will be larger than the time for its alternative formulation. From Fig. 2.6 and Fig. 2.8, we observe that with the same stopping tolerance, the number of iterations of the distributed algorithm for Problem I is around 10 times the number of iterations for its alternative formulation. Therefore, for Problem I, to calculate the exact payment vector, the algorithm will terminate within around $70h$ for the non-colocated scenario and within $100s$ for the colocated scenario. To obtain the payments within 1% error, the algorithm will terminate within around $44min$ and $1s$ for the non-colocated and colocated scenarios.

2.5 Conclusions

In this chapter, we propose a duality-based distributed algorithm to solve Problem I and its alternative formulation. The distributed algorithm is iterative and is based on message passing between each node and its neighbors. No centralized gathering of large amounts of data is required, and each participating institution avoids revealing its proprietary book information to other institutions. The convergence and the practicality of the distributed algorithm are both supported by our simulations.

3. OPTIMAL MITIGATION OF SYSTEMIC RISK IN FINANCIAL NETWORKS UNDER THE RANDOM CAPITAL MODEL

3.1 Introduction

In the previous chapters, we assume that the external asset vector $\mathbf{e}$ is a deterministic vector known by the regulators. However, in some circumstances, the information of the external assets may not be fully available to the regulator. Moreover, some applications, such as stress testing, require forecasting and planning for a wide variety of different contingencies. Such applications call for the use of stochastic models for the nodes' external asset values.

In this chapter, we aim to solve a stochastic version of Problem I under the proportional payment mechanism: how to allocate a fixed amount of cash assistance among the nodes in a financial network in order to minimize the expectation of the (possibly weighted) sum of unpaid liabilities in the system.

We first prove that the optimal cash allocation strategy to minimize the expected weighted sum of unpaid liabilities could be obtained by solving a two-stage stochastic linear program. There exist efficient methods to obtain lower and upper bounds for this stochastic linear program, but there is no guarantee that these bounds are tight enough for practical guidelines. Thus, in the case that the bounds are loose, an approximate solution is needed. Even if we assume that we know the joint distribution of the external asset vector and are able to efficiently obtain independent samples of this vector, it is not trivial to obtain the accurate solution for this stochastic linear program due to the non-linear relationship between the weighted sum of unpaid liabilities and the asset vector. The basic idea of solving the stochastic linear program is to generate a large number of samples of the asset vector and utilize a large-scale

deterministic linear program to approximate the stochastic linear program. In this chapter, we propose two algorithms based on Monte Carlo sampling: the Benders decomposition algorithm and the projected stochastic gradient descent algorithm.

First being proposed in [54], the Benders decomposition decomposes the large-scale linear program into a group of small-scale linear programs and solves them iteratively. In this way, the computational complexity is highly reduced. The projected stochastic gradient descent algorithm is an extension of gradient descent algorithm developed in [55]. Instead of dealing with a huge number of samples at the same time, we only take one sample at each iteration. We calculate the gradient of the weighted sum of unpaid liabilities for that sample, with respect to the cash injection vector and move the cash injection vector along the direction of the negative gradient. As shown in [55], if the step size is selected properly, the procedure will converge to the optimal cash vector, which minimizes the expectation of the weighted sum of unpaid liabilities.

This chapter is organized as follows. Section 3.2 describes the random capital model and shows that it is equivalent to a stochastic linear program. The upper bounds and the lower bounds for the stochastic linear program are discussed in Section 3.3. Section 3.4 adapts the Benders decomposition to solve the linear program. In Section 3.5, we propose the projected stochastic gradient descent algorithm to solve the stochastic linear program. Moreover, we verify the convergence of the projected stochastic gradient descent algorithm on a five-node network for which the optimal solution can be calculated explicitly, and also test the performance of the projected stochastic gradient descent algorithm on a core-periphery network with the same size as the US banking system.

3.2 Model and Notation

As mentioned in Section 3.1, some applications require forecasting and planning for a wide variety of different contingencies. In this case, we aim to solve a stochastic

version of Problem I where $\mathbf{e}$ is modeled as a random vector, from which we are able to efficiently obtain independent samples. The remaining parameters—$\bar{\mathbf{p}}$, Π and $\mathbf{w}$—are still assumed to be deterministic and known, and are defined as in Section 1.2. According to Lemma 1, the clearing payment vector that minimizes the weighted sum of unpaid liabilities is a function of $\mathbf{e}$ and $\mathbf{c}$, which we denote as $\mathbf{p}^*(\mathbf{e}, \mathbf{c})$. If $\mathbf{e}$ is a random vector, so is $\mathbf{p}^*(\mathbf{e}, \mathbf{c})$. We use $W^*(\mathbf{e}, \mathbf{c})$ to denote the corresponding minimum value of the weighted sum of unpaid liabilities. If $\mathbf{e}$ is a random vector, then $W^*(\mathbf{e}, \mathbf{c})$ is a random variable. Given a total amount of cash C, our aim is to find the optimal cash allocation strategy $\mathbf{c}$ to minimize the expectation of the weighted sum of unpaid liabilities. This can be formulated as the following two-stage stochastic LP:

$$\begin{aligned} &\min_{\mathbf{c}} \mathbb{E}_{\mathbf{e}}[W^*(\mathbf{e}, \mathbf{c})] \\ &\text{subject to} \\ &\mathbf{1}^T\mathbf{c} \leq C, \\ &\mathbf{c} \geq \mathbf{0}, \end{aligned} \tag{3.1}$$

where

$$\begin{aligned} W^*(\mathbf{e}, \mathbf{c}) = \ &\min_{\mathbf{p}} \mathbf{w}^T(\bar{\mathbf{p}} - \mathbf{p}) \\ &\text{subject to} \\ &\mathbf{0} \leq \mathbf{p} \leq \bar{\mathbf{p}}, \\ &\mathbf{p} \leq \Pi^T\mathbf{p} + \mathbf{e} + \mathbf{c}. \end{aligned} \tag{3.2}$$

In this chapter, we use the notations in Chapter 1 as well as the following new notations:

- Assume α is an index set of nodes $1, 2, \ldots, N$ and $\mathbf{v}$ is a vector in $\mathbb{R}^N$, then $\mathbf{v}_\alpha$ is a sub-vector of $\mathbf{v}$ containing elements in $\mathbf{v}$ with indices in α. For example, if $\alpha = \{1, 2, N\}$, then $\mathbf{v}_\alpha = [v_1, v_2, v_N]^T$.

- Assume α is an index set of nodes $1, 2, \ldots, N$ and A is an $N \times N$ matrix, then A_α is a sub-matrix of A containing rows in A with index in α. If α contains M indices, A_α is a $M \times N$ matrix.

- Assume α, β are two index set of nodes $1, 2, \ldots, N$ and A is an $N \times N$ matrix, then $A_{\alpha\beta}$ is a sub-matrix of A containing rows in A with index in α and columns in A with index in β. In other words, A_{ij} would be an element in $A_{\alpha\beta}$ if and only if $i \in \alpha$ and $j \in \beta$. If α contains M_1 indices and β contains M_2 indices, $A_{\alpha\beta}$ is a $M_1 \times M_2$ matrix.

- I_N is an $N \times N$ identity matrix.

- ω is an index set containing all defaulting nodes; $\bar{\omega}$ is an index set containing all non-defaulting nodes. $|\omega|$ is the number of elements in ω.

3.3 Upper Bounds and Lower Bounds

Even if the joint distribution of $\mathbf{e}$ is known, the distributions of $\mathbf{p}^*(\mathbf{e}, \mathbf{c})$ and $W^*(\mathbf{e}, \mathbf{c})$ can rarely be computed in closed form. So for real-world problems, people are inclined to solve a simpler version of the stochastic linear program as an estimation, avoiding the computational difficulty.

One frequently used simpler version is to solve the deterministic linear program obtained by replacing random vector $\mathbf{e}$ by its expected value. In [56], it is defined as *expected value problem*, which is simply

$$EV = \min_{\mathbf{c}} W^*(\mathbb{E}[\mathbf{e}], \mathbf{c}). \tag{3.3}$$

Denote the optimal solution to LP (3.3) by $\mathbf{c}^+$, called the *expected value solution*. Compared to the stochastic linear program, LP (3.3) is easy to solve. But generally, $\mathbf{c}^+$ is not a good estimation of the solution of (3.1) with rare exceptions, such as when $W^*(\mathbf{e}, \mathbf{c})$ is linear with respect to $\mathbf{e}$. In fact, EV is proved in [56] to be a lower bound of SLP (3.1) by Jensen's Inequality since $W^*(\mathbf{e}, \mathbf{c})$ is convex with respect to $\mathbf{e}$.

The expected result of using the EV solution $\mathbf{c}^+$ is denoted:

$$EEV = \mathbb{E}_{\mathbf{e}}[W^*(\mathbf{e}, \mathbf{c}^+)]. \tag{3.4}$$

It measures the performance of using $\mathbf{c}^+$ as an approximation of the solution to (3.1). EEV is an upper bound to the solution of SLP (3.1).

Another lower bound is called *wait-and-see* solution which is defined in [56] as follows:

$$WS = \mathbb{E}_{\mathbf{e}}[\min_{\mathbf{c}} W^*(\mathbf{e}, \mathbf{c})]. \tag{3.5}$$

Intuitively, the *wait-and-see* solution WS is a lower bound of (3.1) because the regulator waits until the random shock occurs and then takes actions. More information leads to a better bailout strategy. Strict proof showing that this is a lower bound is provided in [56].

It is proved that $WS \geq EV$ by Jensen's Inequality in [56]. In other words, WS is a tighter lower bound than EV.

3.4 Benders Decomposition

Although the expected value solution is easy to obtain, it is not a good approximation in general. Thus, there is an acute need for other approximate solutions, which are both tractable and accurate. In order to solve (3.1), we take M independent samples of the asset vector, denoted as $\mathbf{e}^1, \mathbf{e}^2, \ldots, \mathbf{e}^M$, and use $\frac{1}{M}\sum_{m=1}^{M} W^*(\mathbf{e}^m, \mathbf{c})$ to approximate $\mathbb{E}_{\mathbf{e}}[W^*(\mathbf{e}, \mathbf{c})]$. By the weak law of large numbers, when M is large enough, the sample average is a good estimate of $\mathbb{E}_{\mathbf{e}}[W^*(\mathbf{e}, \mathbf{c})]$. This motivates approximating Eq. (3.1) as follows:

$$\min_{\mathbf{c}} \frac{1}{M}\sum_{m=1}^{M} W^*(\mathbf{e}^m, \mathbf{c}) \tag{3.6}$$

subject to

$$\mathbf{1}^T\mathbf{c} \leq C,$$

$$\mathbf{c} \geq \mathbf{0}.$$

Similar to Theorem 1, the optimization problems (3.2) and (3.6) can be combined into one single LP:

$$\max_{\mathbf{c},\mathbf{p}^m} \sum_{m=1}^{M} \mathbf{w}^T \mathbf{p}^m \tag{3.7}$$

subject to

$$\mathbf{1}^T \mathbf{c} \leq C,$$

$$\mathbf{c} \geq \mathbf{0},$$

$$\mathbf{0} \leq \mathbf{p}^m \leq \bar{\mathbf{p}}, \text{ for } m = 1, 2, \ldots, M,$$

$$\mathbf{p}^m \leq \Pi^T \mathbf{p}^m + \mathbf{e}^m + \mathbf{c}, \text{ for } m = 1, 2, \ldots, M.$$

Since $\mathbf{c}$ and $\mathbf{p}^m$ for $m = 1, 2, \ldots, M$ are all N-dimensional vectors, LP (3.7) contains $MN + N$ variables. The computational complexity of solving an LP with $MN + N$ variables is $O((MN + N)^3)$ [57]. To achieve a high accuracy, M needs to be a large number. Then the computational burden is large if we want to solve LP (3.7) directly. The memory complexity, which is $O(MN^2)$, may also be prohibitive for large M and N. Hence, efficient algorithms to solve LP (3.7) are needed.

If the cash injection vector $\mathbf{c}$ is fixed, then the LP (3.7) can be split into M smaller independent LPs—one for each sample $\mathbf{e}^m$—each of which can be solved independently for each $\mathbf{p}^m$. In this case, instead of solving an LP with MN variables, we solve M LPs with N variables each, which significantly reduces the computational complexity. Inspired by this idea, we apply Benders decomposition to the LP (3.7). Benders decomposition, which is described in [54, 58, 59], makes a partition of $\mathbf{c}$ and $\mathbf{p}^m$ $(m = 1, 2, \ldots, M)$ and allows us to find $\mathbf{p}^m$ iteratively with fixed $\mathbf{c}$ in each step. In fact, for our problem, Benders decomposition can be further simplified due to some special properties of (3.7).

From the proof of Lemma 1, we know that for any fixed $\mathbf{c}$, the feasible region of $\mathbf{p}^m$ is non-empty. Thus, (3.7) is equivalent to the following problem:

$$\begin{aligned} &\max_{\mathbf{c}} V(\mathbf{c}) \\ &\text{subject to} \\ &\mathbf{1}^T\mathbf{c} \leq C, \\ &\mathbf{c} \geq \mathbf{0}. \end{aligned} \tag{3.8}$$

where

$$V(\mathbf{c}) = \max_{\mathbf{p}^m} \sum_{m=1}^{M} \mathbf{w}^T\mathbf{p}^m \tag{3.9}$$

$$\text{subject to}$$

$$\mathbf{p}^m \geq \mathbf{0}, \text{ for } m = 1, 2, \ldots, M,$$

$$\mathbf{p}^m \leq \bar{\mathbf{p}}, \text{ for } m = 1, 2, \ldots, M, \tag{3.10}$$

$$\mathbf{p}^m \leq \Pi^T\mathbf{p}^m + \mathbf{e}^m + \mathbf{c}, \text{ for } m = 1, 2, \ldots, M. \tag{3.11}$$

We let $\boldsymbol{\mu}^1, \ldots, \boldsymbol{\mu}^M$ be the dual variables for the M constraints (3.10), and we let $\boldsymbol{\nu}^1, \ldots, \boldsymbol{\nu}^M$ be the dual variables for the M constraints (3.11). Then $V(\mathbf{c})$ can be obtained from the following dual problem:

$$V(\mathbf{c}) = \min_{\boldsymbol{\mu}^m, \boldsymbol{\nu}^m} \sum_{m=1}^{M} \left[\bar{\mathbf{p}}^T\boldsymbol{\mu}^m + \mathbf{e}^{mT}\boldsymbol{\nu}^m + \mathbf{c}^T\boldsymbol{\nu}^m\right] \tag{3.12}$$

$$\text{subject to}$$

$$\boldsymbol{\mu}^m \geq \mathbf{0}, \text{ for } m = 1, 2, \ldots, M,$$

$$\boldsymbol{\nu}^m \geq \mathbf{0}, \text{ for } m = 1, 2, \ldots, M,$$

$$\boldsymbol{\nu}^m \geq \Pi\boldsymbol{\nu}^m + \mathbf{w} - \boldsymbol{\mu}^m, \text{ for } m = 1, 2, \ldots, M.$$

Note that, since $V(\mathbf{c})$ minimizes the objective function of LP (3.12) subject to the constraints of LP (3.12), we have that $V(\mathbf{c})$ is the greatest lower bound of this objective function, subject to these constraints. Therefore, LP (3.8) is equivalently rewritten as

the maximization of the lower bound to the objective function of LP (3.12), subject to the constraints of both LP (3.8) and LP (3.12):

$$\max_{\mathbf{c},\theta} \theta \tag{3.13}$$

subject to

$$\mathbf{1}^T\mathbf{c} \leq C,$$

$$\mathbf{c} \geq \mathbf{0},$$

$$\theta \leq \sum_{m=1}^{M} \left[\bar{\mathbf{p}}^T\boldsymbol{\mu}^m + \mathbf{e}^{mT}\boldsymbol{\nu}^m + \mathbf{c}^T\boldsymbol{\nu}^m\right] \tag{3.14}$$

for all $(\boldsymbol{\mu}^m, \boldsymbol{\nu}^m)$ in the feasible region of (3.12).

LP (3.13) is equivalent to (3.7), with fewer variables but an infinite number of constraints because constraint (3.14) must be satisfied by every pair $(\boldsymbol{\mu}^m, \boldsymbol{\nu}^m)$ from the feasible region of LP (3.12). The key idea is solving a relaxed version of (3.13) by ignoring all but a few of the constraints (3.14). Assume the optimal solution of this relaxed program is $(\mathbf{c}^*, \theta^*)$. If the solution satisfies all the ignored constraints, the optimal solution has been found; otherwise, we generate a new constraint by solving (3.12) with fixed $\mathbf{c} = \mathbf{c}^*$ and add it to the relaxed problem. Here is the summary of the Benders decomposition algorithm:

1. Initialization: set $\theta^0 \leftarrow -\infty$, $K \leftarrow 0$, $\mathbf{c}^0 \leftarrow \mathbf{0}$, $l \leftarrow 0$.

2. Fix $\mathbf{c}^l$, solve the following M sub-programs for $m = 1, 2, \ldots, M$:

$$V^m = \min_{\boldsymbol{\mu}^m, \boldsymbol{\nu}^m} \sum_{m=1}^{M} \left[\bar{\mathbf{p}}^T\boldsymbol{\mu}^m + \mathbf{e}^{mT}\boldsymbol{\nu}^m + \mathbf{c}^{lT}\boldsymbol{\nu}^m\right]$$

subject to

$$\boldsymbol{\mu}^m \geq \mathbf{0}, \text{ for } m = 1, 2, \ldots, M,$$

$$\boldsymbol{\nu}^m \geq \mathbf{0}, \text{ for } m = 1, 2, \ldots, M,$$

$$\boldsymbol{\nu}^m \geq \Pi\boldsymbol{\nu}^m + \mathbf{w} - \boldsymbol{\mu}^m, \text{ for } m = 1, 2, \ldots, M.$$

Denote the solution as $(\boldsymbol{\mu}^{m*}, \boldsymbol{\nu}^{m*})$, for $m = 1, 2, \ldots, M$.

3. If $\sum_{m=1}^{M} V^m = \theta^l$, terminate and $\mathbf{c}^l$ is the optimal.

4. Set $l \leftarrow l+1$, $K \leftarrow K+1$, $(\boldsymbol{\mu}^m(K), \boldsymbol{\nu}^m(K)) \leftarrow (\boldsymbol{\mu}^{m*}, \boldsymbol{\nu}^{m*})$, for $m = 1, 2, \ldots, M$.

5. Solve the following master problem:

$$\begin{aligned} &\max_{\mathbf{c},\theta} \theta \qquad (3.15)\\ &\text{subject to}\\ &\mathbf{1}^T\mathbf{c} \le C,\\ &\mathbf{c} \ge \mathbf{0},\\ &\theta \le \sum_{m=1}^{M} \left[\bar{\mathbf{p}}^T\boldsymbol{\mu}(k)^m + \mathbf{e}^{mT}\boldsymbol{\nu}(k)^m + \mathbf{c}^T\boldsymbol{\nu}(k)^m\right]\\ &\text{for } k = 1, 2, \ldots, K. \end{aligned}$$

Denote the solution as $(\mathbf{c}^*, \theta^*)$. Set $\theta^l \leftarrow \theta^*$, $\mathbf{c}^l \leftarrow \mathbf{c}^*$. Then go to Step 2.

In this algorithm, at each iteration, we solve $M+1$ LPs with N variables instead of one LP with $MN+N$ variables, which saves both computational complexity and memory cost. The above algorithm is simpler than the general form of Benders decomposition (Section 2.3 in [58]), since LP (3.9) and (3.15) are always feasible and bounded.

3.5 Projected Stochastic Gradient Descent Algorithm

In this section, we introduce the projected stochastic gradient descent algorithm (PSGD) to solve (3.1). This is an online learning algorithm, which allows us to handle one sample at a time, without building a huge linear program. The basic idea is that for each sample $\mathbf{e}^m$, we move the solution $\mathbf{c}$ along the direction of the negative gradient of $W^*(\mathbf{e}^m, \mathbf{c})$ with respect to $\mathbf{c}$ and then project the result onto the set defined by the constraints of (3.1). This procedure will converge to the optimal solution if the step size is selected properly [55]. In practice, the regulator usually fully utilizes the

budget, so in this section, we just consider $\mathbf{1}^T\mathbf{c} = C$ to speed up the convergence and facilitate the projection. The algorithm proceeds as follows. At iteration m,

1. Sample an asset vector $\mathbf{e}^m$.

2. Move $\mathbf{c}$ along the negative gradient of $W^*(\mathbf{e}^m, \mathbf{c}^{m-1})$ according to the following equation:
$$\tilde{\mathbf{c}}^m = \mathbf{c}^{m-1} - \gamma^m \nabla_{\mathbf{c}} W^*(\mathbf{e}^m, \mathbf{c}^{m-1}). \tag{3.16}$$

3. Set $\mathbf{c}^m$ as the projection of $\tilde{\mathbf{c}}^m$ onto the set $\{\mathbf{c} : \ \mathbf{1}^T\mathbf{c} = C, \ \mathbf{c} \geq \mathbf{0}\}$.

According to [55], step size γ^m should satisfy the condition that $\sum_{m=1}^{\infty}(\gamma^m)^2 < \infty$ and $\sum_{m=1}^{\infty}\gamma^m = \infty$. Thus, a proper choice could be $\gamma^m = 1/m$.

In this projected stochastic gradient descent algorithm, instead of generating a huge number of possible scenarios and solving a large-scale linear program, we solve one N-variable LP and one N-variable quadratic program at each iteration. This algorithm is memory efficient because it requires no storage except the current solution of $\mathbf{c}$.

Now we discuss the details in the above algorithm: obtaining the gradient in Step 2, calculating the projection in Step 3 and the stopping criterion for the iterations.

Note that $W^*(\mathbf{e}^m, \mathbf{c}^{m-1}) = \mathbf{w}^T\bar{\mathbf{p}} - U(\mathbf{e}^m, \mathbf{c}^{m-1})$, where

$$\begin{aligned} U(\mathbf{e}^m, \mathbf{c}^{m-1}) = \ & \max_{\mathbf{p}} \mathbf{w}^T\mathbf{p} \\ & \text{subject to} \\ & \mathbf{0} \leq \mathbf{p} \leq \bar{\mathbf{p}}, \\ & \mathbf{p} \leq \Pi^T\mathbf{p} + \mathbf{e}^m + \mathbf{c}^{m-1}. \end{aligned} \tag{3.17}$$

To obtain the gradient of $W^*(\mathbf{e}^m, \mathbf{c}^{m-1})$ in Step 2, we consider the dual problem of LP (3.17):

$$U(\mathbf{e}^m, \mathbf{c}^{m-1}) = \min_{\boldsymbol{\mu},\boldsymbol{\nu}} \left[\bar{\mathbf{p}}^T\boldsymbol{\mu} + \mathbf{e}^{mT}\boldsymbol{\nu} + \mathbf{c}^{(m-1)}\boldsymbol{\nu}\right] \tag{3.18}$$

$$\text{subject to}$$

$$\boldsymbol{\mu} \geq \mathbf{0},$$

$$\boldsymbol{\nu} \geq \mathbf{0},$$

$$\boldsymbol{\nu} \geq \Pi\boldsymbol{\nu} + \mathbf{w} - \boldsymbol{\mu}.$$

Assuming that $(\boldsymbol{\mu}^*, \boldsymbol{\nu}^*)$ is a solution of (3.18), we have:

$$\nabla_{\mathbf{c}} W^*(\mathbf{e}^m, \mathbf{c}^{m-1}) = -\nabla_{\mathbf{c}} U(\mathbf{e}^m, \mathbf{c}^{m-1}) = -\boldsymbol{\nu}^*.$$

Generally, an LP solver is needed to obtain the gradient. In this case, however, since we only need $\boldsymbol{\nu}$, not $\boldsymbol{\mu}$, there is an easier way to find $\boldsymbol{\nu}$, which avoids wasting time on environment setting, problem establishment and algorithm selection in the LP solver.

First, we calculate the clearing payment vector in the financial system given the asset vector $\mathbf{e}^m$ and the cash injection vector $\mathbf{c}^{m-1}$ by the fixed-point method in Appendix A. Then we separate the nodes into two groups: defaulting nodes and non-defaulting nodes. For non-defaulting nodes, they pay their liabilities in full so injecting more cash into these nodes will not change the weighted sum of unpaid liabilities. In other words, the derivatives with respect to cash injection into non-defaulting nodes are all zero. For defaulting nodes, their payments will increase if they receive more external cash injections. Thus, the derivatives with respect to cash injection into defaulting nodes are non-zeros and could be obtained via the following theorem.

Theorem 8 *Assume that the liabilities matrix L and the weight vector $\mathbf{w}$ are fixed and known. The asset vector is $\mathbf{e}^m$, and the cash injection vector is $\mathbf{c}^{m-1}$. Assume that the system utilizes the proportional payment mechanism with no bankruptcy costs.*

Using the model and notations defined in Section 1.2, the gradient of the weighted sum of unpaid liabilities $W^*(\mathbf{e}^m, \mathbf{c}^{m-1})$ *with respect to* $\mathbf{c}$ *is given by*

$$\nabla_{\mathbf{c}} W^*(\mathbf{e}^m, \mathbf{c}^{m-1}) = \begin{pmatrix} \nabla_{\mathbf{c}_{\bar{\omega}}} W^*(\mathbf{e}^m, \mathbf{c}^{m-1}) \\ \nabla_{\mathbf{c}_{\omega}} W^*(\mathbf{e}^m, \mathbf{c}^{m-1}) \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ -\mathbf{w}_{\omega}^T (I_{|\omega|} - (\Pi^T)_{\omega\omega})^{-1} \end{pmatrix} \tag{3.19}$$

Proof According to proportional payment mechanism with no bankruptcy costs, the non-defaulting nodes in index set ω pay their liabilities in full and the defaulting nodes in index set $\bar{\omega}$ pay all their available funds, so we have:

$$\begin{pmatrix} \mathbf{p}_{\bar{\omega}} \\ \mathbf{p}_{\omega} \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{p}}_{\bar{\omega}} \\ (\Pi^T)_{\omega} \mathbf{p} + \mathbf{e}_{\omega}^m + \mathbf{c}_{\omega}^{m-1} \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{p}}_{\bar{\omega}} \\ (\Pi^T)_{\omega\bar{\omega}} \bar{\mathbf{p}}_{\bar{\omega}} + (\Pi^T)_{\omega\omega} \mathbf{p}_{\omega} + \mathbf{e}_{\omega}^m + \mathbf{c}_{\omega}^{m-1} \end{pmatrix} \tag{3.20}$$

From Eq. (3.20), we have

$$\mathbf{p}_{\omega} = (I_{|\omega|} - (\Pi^T)_{\omega\omega})^{-1} ((\Pi^T)_{\omega\bar{\omega}} \bar{\mathbf{p}}_{\bar{\omega}} + \mathbf{e}_{\omega}^m + \mathbf{c}_{\omega}^{m-1})$$

Thus, the weighted sum of unpaid liabilities in the financial system given the asset vector $\mathbf{e}^m$ and the cash injection vector $\mathbf{c}^{m-1}$ can be calculated directly by follows:

$$\begin{aligned} W^*(\mathbf{e}^m, \mathbf{c}^{m-1}) &= \mathbf{w}^T (\bar{\mathbf{p}} - \mathbf{p}) \\ &= \mathbf{w}_{\bar{\omega}}^T (\bar{\mathbf{p}}_{\bar{\omega}} - \mathbf{p}_{\bar{\omega}}) + \mathbf{w}_{\omega}^T (\bar{\mathbf{p}}_{\omega} - \mathbf{p}_{\omega}) \\ &= \mathbf{w}_{\omega}^T (\bar{\mathbf{p}}_{\bar{\omega}} - (I_{|\omega|} - (\Pi^T)_{\omega\omega})^{-1} ((\Pi^T)_{\omega\bar{\omega}} \bar{\mathbf{p}}_{\bar{\omega}} + \mathbf{e}_{\omega}^m + \mathbf{c}_{\omega}^{m-1})) \end{aligned} \tag{3.21}$$

From Eq. (3.21), we have:

$$\nabla_{\mathbf{c}_{\bar{\omega}}} W^*(\mathbf{e}^m, \mathbf{c}^{m-1}) = \mathbf{0},$$

$$\nabla_{\mathbf{c}_{\omega}} W^*(\mathbf{e}^m, \mathbf{c}^{m-1}) = -\mathbf{w}_{\omega}^T (I_{|\omega|} - (\Pi^T)_{\omega\omega})^{-1}.$$

■

In Step 3, we find the projection of $\tilde{\mathbf{c}}^m$ on the feasible region. This could be done by solving the following quadratic program:

$$\min_{\mathbf{c}} \|\mathbf{c} - \tilde{\mathbf{c}}^m\|_2^2 \tag{3.22}$$

subject to

$$\mathbf{1}^T\mathbf{c} = C, \tag{3.23}$$

$$\mathbf{c} \geq \mathbf{0}. \tag{3.24}$$

But, once again, there is a simpler algorithm to solve Eq. (3.22) with a $O(N)$ computational complexity. $\mathbf{c}$ can be found explicitly as $c_i = \max\{\tilde{c}_i - \frac{\kappa}{2}, 0\}$ for $i = 1, 2, \ldots, N$, where κ is a scalar that needs to be adjusted so that $\mathbf{1}^T\mathbf{c} = C$.

If $\tilde{\mathbf{c}}^m \geq \mathbf{0}$ and $\mathbf{1}^T\tilde{\mathbf{c}}^m = C$, then $\tilde{\mathbf{c}}^m$ locates in the feasible region of quadratic program (3.22) and the optimal solution is $\mathbf{c} = \tilde{\mathbf{c}}^m$. If $\tilde{\mathbf{c}}^m$ violates the constraints (3.23), i.e., $\mathbf{1}^T\tilde{\mathbf{c}}^m \neq C$, we simplify the quadratic problem by *Karush-Kuhn-Tucker* (KKT) conditions.

Let a scalar κ and an $N \times 1$ vector $\boldsymbol{\iota}$ be Lagrange multipliers for constraint (3.23) and (3.24), respectively. We define the Lagrangian as follows:

$$F(\mathbf{c}, \kappa, \boldsymbol{\iota}) = \|\mathbf{c} - \tilde{\mathbf{c}}^m\|_2^2 + \kappa(\mathbf{1}^T\mathbf{c} - C) - \boldsymbol{\iota}^T\mathbf{c} \tag{3.25}$$

Then the primal and dual optimal $\mathbf{c}$ and $(\kappa, \boldsymbol{\iota})$ will satisfy the following KKT conditions:

$$\nabla_{\mathbf{c}} F(\mathbf{c}, \kappa, \boldsymbol{\iota}) = \mathbf{0}, \tag{3.26}$$

$$\mathbf{1}^T\mathbf{c} = C, \tag{3.27}$$

$$\mathbf{c} \geq \mathbf{0}, \tag{3.28}$$

$$\boldsymbol{\iota} \geq \mathbf{0}, \tag{3.29}$$

$$\iota_i c_i = 0, \text{ for } i = 1, 2, \ldots, N. \tag{3.30}$$

From KKT constraint (3.26), we have $c_i = \tilde{c}_i^m - \kappa/2 + \iota_i/2$, for $i = 1, 2, \ldots, N$.

- If $\tilde{c}_i^m - \kappa/2 > 0$, then $c_i > 0$ since $\iota_i \geq 0$. By KKT constraint (3.30), $\iota_i = 0$. Thus, $c_i = \tilde{c}_i^m - \kappa/2$.

- If $\tilde{c}_i^m - \kappa/2 < 0$, ι_i should be positive to guarantee that $c_i \geq 0$. Since $\iota_i c_i = 0$ and $\iota_i > 0$, we conclude $c_i = 0$.

- If $\tilde{c}_i^m - \kappa/2 = 0$, then $c_i = \iota_i/2$. Since $\iota_i c_i = 0$, we have $c_i = \iota_i = 0$.

Thus, $c_i = \max\{\tilde{c}_i^m - \kappa/2, 0\}$, for $i = 1, 2, \ldots, N$.

Then combined with KKT constraint (3.27), the problem becomes to find a non-zero scalar κ making $\mathbf{1}^T\mathbf{c} = C$, where $c_i = \max\{\tilde{c}_i^m - \kappa/2, 0\}$, for $i = 1, 2, \ldots, N$. This is a typical single water-level and single constrained water filling problem which can be solved with complexity $O(N)$ [60]. A variety of both iterative and exact algorithms are provided in [61–65]. Here, we just list one of them to find $\mathbf{c}$:

1. $k \leftarrow 0$, for $i = 1, 2, \ldots, N$, $c_i^0 \leftarrow \tilde{c}_i^m$.

2. If $\mathbf{1}^T\mathbf{c}^k = C$, $\mathbf{c} \leftarrow \mathbf{c}^k$, stop.

3. If $\mathbf{1}^T\mathbf{c}^k < C$, for $i = 1, 2, \ldots, N$,

$$c_i \leftarrow c_i^k + (C - \mathbf{1}^T\mathbf{c}^k)/N,$$

   stop.

4. If $\mathbf{1}^T\mathbf{c}^k > C$, $Z \leftarrow \|c_i^k\|_0$, for i such that $c_i^k > 0$,

$$c_i^k = \max\{c_i^k - (\mathbf{1}^T\mathbf{c}^k - C)/Z, 0\},$$

   go to Step 2.

For the stopping criterion for such iterative algorithm, we usually compare the current cash injection vector $\mathbf{c}$ and the one in the last iteration. If they are close enough, we terminate the iterations. However, in PSGD, since at each iteration the asset vector $\mathbf{e}$ is chosen randomly, it is possible that the stopping condition is satisfied by coincidence before the sequence of $\mathbf{c}$ converges. To reduce the probability of such coincidence, we define a stopping condition window S and instead of just considering one iteration, we check S consecutive iterations. The new stopping criterion requires

that the differences between a cash injection vector and the subsequent one are all small enough for S consecutive iterations. In other words, if at iteration $m > S$, for $s = 1, 2, \ldots, S$, $\|\mathbf{c}^{m-s+1} - \mathbf{c}^{m-s}\|_1 < \delta$, the algorithm will be terminated. Here, δ is a small number.

3.5.1 Importance Sampling

Sometimes, the weighted sum of unpaid liabilities will significantly change when the random asset vector $\mathbf{e}$ locates in a certain region, called important region. If the probability of getting samples in the important region is too small, we may need a large number of samples to make sure that we get enough samples in that region. In order to solve this problem and improve the performance of the projected stochastic gradient descent algorithm, we could apply importance sampling.

The basic idea of importance sampling is to change the probability measure of the random variables to make the probability of getting samples in important region larger. In this paper, our objective is to calculate the expectation of the weighted sum of unpaid liabilities $W^*(\mathbf{e}, \mathbf{c})$ where $\mathbf{e}$ is a random vector under the original measure P_1. We could calculate the expectation under a new measure P_2, which emphasizes the important region, by doing following transformation:

$$\begin{aligned}
\mathbb{E}_{P_1}\left[W^*(\mathbf{e}, \mathbf{c})\right] &= \int W^*(\mathbf{e}, \mathbf{c}) f_1(\mathbf{e}) \mathrm{d}\mathbf{e} \\
&= \int W^*(\mathbf{e}, \mathbf{c}) \frac{f_1(\mathbf{e})}{f_2(\mathbf{e})} f_2(\mathbf{e}) \mathrm{d}\mathbf{e} \\
&= \mathbb{E}_{P_2}\left[W^*(\mathbf{e}, \mathbf{c}) \frac{f_1(\mathbf{e})}{f_2(\mathbf{e})}\right]
\end{aligned}$$

where $f_1(\mathbf{e})$ is the pdf of $\mathbf{e}$ under measure P_1 and $f_2(\mathbf{e})$ is the pdf of $\mathbf{e}$ under the new measure P_2. If P_2 is selected properly, the regulator may need less samples to achieve the same accuracy.

To sum up, before using the projected stochastic gradient descent algorithm, the regulator should identify the pathological samples that could potentially cause sig-

nificant unpaid liabilities; increase their sampling probability, and then adjust the formula of the weighted sum of unpaid liabilities accordingly.

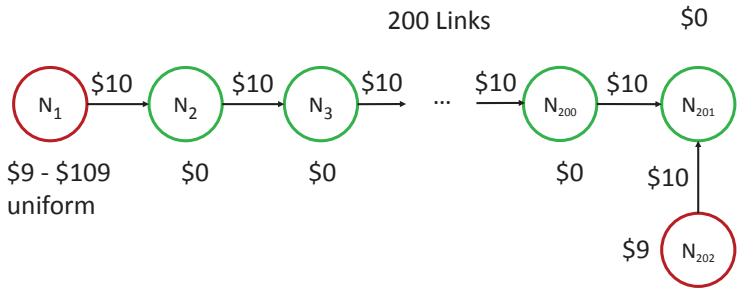


Fig. 3.1. An example illustrating the power of importance sampling.

For example, in Fig. 3.1, there are 202 nodes. Node 1 owes \$10 to node 2, node 2 owes \$10 to node 3, ..., node 200 owes \$10 to node 201 and node 202 owes \$10 to node 200. Node 201 has \$9 on hand and the asset of node 1 is uniformly distributed in $[9, 109]$. All other nodes have no cash on hand. We assume $\mathbf{w} = \mathbf{1}$ and the regulator has \$1 to be injected to the network. Without cash injection, node 201 is safe and node 202 will default with unpaid liability \$1. If the asset of node 1 is greater than or equal to \$10, node 1 to node 200 are all safe; if the asset of node 1 is less than \$10, all of them will default, causing a total unpaid liability of $200(10 - e_1)$, where e_1 is the asset value of node 1. In other words, without cash injection, node 202 will result in \$1 unpaid liability, while nodes 1 to 200 will be safe with probability 99% and will result in a large total unpaid liabilities with probability 1%. The regulator should only consider to inject cash into node 1 and node 202 since node 2 to 200 could get

money from node 1. Assume we inject $\$c$ to node 1 and $\$(1-c)$ to node 202, then the expected total unpaid liability is as follows:

$$\begin{aligned}
&\mathbb{E}_{P_1}[W^*(\mathbf{e},\mathbf{c})] \\
&= \mathbb{E}_{P_1}\{200 \times [10-(e_1+c)]^+ + [10-(e_{202}+1-c)]^+\} \\
&= 200 \times \int_9^{109} [10-(e_1+c)]^+ \times \frac{1}{109-9} \mathrm{d}e_1 \\
&\quad + [10-(9+1-c)]^+ \\
&= c^2 - c + 1
\end{aligned}$$

$\mathbb{E}_{P_1}[W^*(\mathbf{e},\mathbf{c})]$ is minimum when $c = 0.5$. Thus, the optimal allocation strategy is injecting \$0.5 to node 1 and \$0.5 to node 202.

When we apply the projected stochastic gradient descent algorithm on this network, we only have 1% of samples for e_1 in $[9, 10]$, which leads to a large amount of unpaid liabilities. The other 99% of the samples result in nodes 1 to 200 being safe so that the regulator inclines towards node 202 based on the algorithm. Thus, the algorithm will either converge to injecting \$1 to node 202 or require a large number of samples to achieve the global optimum.

In order to make the projected stochastic gradient descent algorithm more efficient, we apply importance sampling on node 1. We define a new probability measure P_2, where e_1 is $1/2$ in $[9, 10]$ and $1/198$ in $[10, 109]$. Then the expected weighted sum of unpaid liabilities could be expressed as follows:

$$\begin{aligned}
&\mathbb{E}_{P_1}[W^*(\mathbf{e},\mathbf{c})] \\
&= \mathbb{E}_{P_1}\left[200 \times [10-(e_1+c)]^+\right] + [10-(e_{202}+1-c)]^+ \\
&= \mathbb{E}_{P_2}\left[200 \times [10-(e_1+c)]^+ \times \frac{f_1(e_1)}{f_2(e_1)}\right] \\
&\quad + [10-(e_{202}+1-c)]^+ \\
&= c^2 - c + 1.
\end{aligned}$$

We can see that under the new measure the expectation remains the same but the speed of convergence is improved since we get more samples of e_1 in the important region $[9, 10]$.

3.5.2 Numerical Results

In this section, we test the performance of the projected stochastic gradient descent algorithm on two examples: a five-node network for which the optimal solution can be calculated explicitly and a large scale core-periphery network which has a comparable size with the US banking system.

3.5.2.1 Example 1: A Five-Node Network

First, we illustrate the convergence of PSGD to the optimal solution on a five-node network example, for which the optimal solution of the 2-stage stochastic linear program can be calculated directly. Also, we show that in this example, the expected value solution defined in Eq. (3.4) is not a good approximation to the optimum.

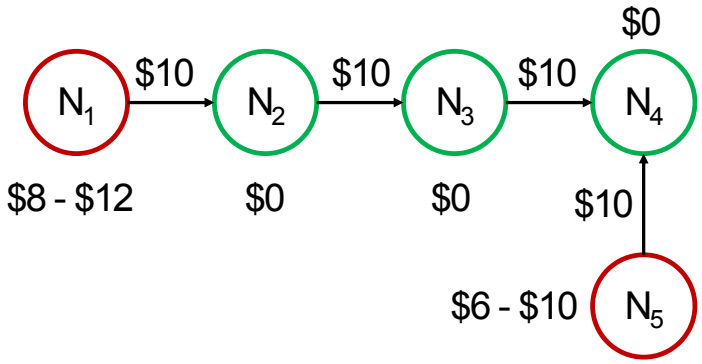


Fig. 3.2. A five-node financial network.

Consider the borrower-lender network in Fig. 3.2, where node N_i owes \$10 to N_{i+1} for $i = 1, 2, 3$, and node N_5 owes \$10 to node N_4. Nodes N_2, N_3 and N_4 have no

cash on hand, while nodes N_1 and N_5 have assets are uniformly and independently distributed in $[8, 12]$ and $[6, 10]$, respectively. Assume we have \$2 to be injected into the system. We are seeking the optimal cash allocation vector $\mathbf{c}$ to minimize the expected weighted sum of unpaid liabilities, $\mathbb{E}_{\mathbf{e}}[W^*(\mathbf{e}, \mathbf{c})]$. All liabilities have the same weight ($\mathbf{w} = \mathbf{1}$).

First, we consider the expected value solution, which is easy to obtain. When the asset vector is $\mathbb{E}[\mathbf{e}]$, N_1 has \$10 on hand. It is able to pay its liability in full and so are N_2 and N_3. On the other hand, N_5 only has \$8 on hand. The expected value solution injects all \$2 to node N_5 so that all nodes can avoid default. However, there is no reason to believe that this solution is in any way near the optimum. Intuitively, we should inject some cash to node N_1 because once N_1 cannot afford its liability ($e_1 < 10$), both N_2 and N_3 will default. It leads to a more severe shock than the default of N_5.

In fact, the optimal solution can be calculated explicitly for the network in Fig. 3.2. Since nodes N_2 and N_3 could get money from N_1 and node N_4 has zero liability, \$2 should be split into N_1 and N_5, while other three nodes receive nothing. Assume we inject c_1 into N_1 and c_5 into N_5, the funds available to make payment of N_1 is $e_1 + c_1$ and the funds available to make payment of N_5 is $e_5 + c_5$. Then the weighted sum of unpaid liabilities in this network is:

$$W^*[\mathbf{e}, \mathbf{c}] = 3 \times [10 - (e_1 + c_1)]^+ + [10 - (e_5 + c_5)]^+$$

where $[x]^+ = \max\{0, x\}$.

Since e_1 is a random variable uniformly distributed in $[8, 12]$ and e_5 is uniform in $[6, 10]$, the expectation of the weighted sum of unpaid liabilities can be calculated as follows:

$$\begin{aligned} & \mathbb{E}_{\mathbf{e}}[W^*(\mathbf{e}, \mathbf{c})] \\ = \; & \mathbb{E}_{\mathbf{e}}\{3 \times [10 - (e_1 + c_1)]^+ + [10 - (e_5 + c_5)]^+\} \\ = \; & \frac{3}{8}(2 - c_1)^2 + \frac{1}{8}(4 - c_5)^2 \end{aligned}$$

Considering that $c_1 + c_5 = 2$, we find that the minimum of $\mathbb{E}_{\mathbf{e}}[W^*(\mathbf{e}, \mathbf{c})]$ is achieved at $c_1 = c_5 = 1$ and the minimum value of the expectation is \$1.5. We could see that the optimum is far away from the expected value solution, where $c_1 = 0$, $c_5 = 2$ and the expectation is \$2.

We run PSGD on the five-node network with initial $\mathbf{c} = \mathbf{0}$. The step size is $\gamma = 1/m$, where m is the number of iterations. The stopping condition window $S = 5$ and the tolerance $\delta = 10^{-5}$.

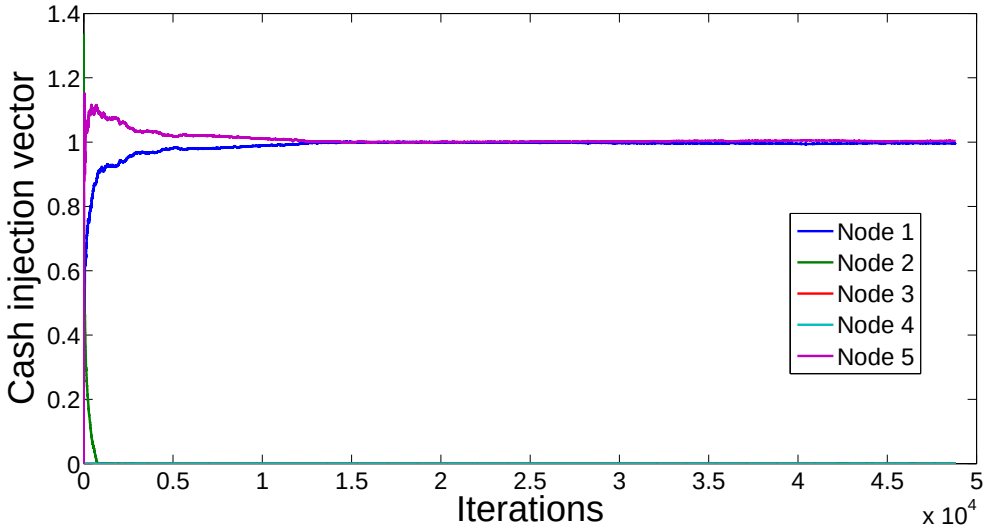


Fig. 3.3. Evolution of the cash injections through the iterations of PSGD for finding the optimal allocation of a \$2 cash injection into the network of Fig. 3.2.

Fig. 3.3 illustrates the evolution of cash injection vector $\mathbf{c}$, as a function of the number of iterations of PSGD. The cash injection vector $\mathbf{c}$ converges to $[0.9958, 0.0002, 0.0001, 0, 1.0037]$, which is close to the theoretical optimum $[1, 0, 0, 0, 1]$.

3.5.2.2 Example 2: A Core-Periphery Network

A variety of prior literature, e.g. [6], suggests that the US interbank network is well modeled as a core-periphery network that consists of a core of about 15 highly interconnected banks to which most other banks connect. Therefore, we test PSGD on the core-periphery network shown in Fig. 1.17. It contains 15 fully connected core nodes with 70 periphery nodes each. Each periphery node has a single link pointing to the corresponding core node. All the obligation amounts $L_{i,j}$ are independent and uniformly distributed in $[0, 10]$ if i and j are a pair of core nodes, and in $[0, 1]$ if i is a periphery node and j is a core node. We generate ten independent samples of a core-periphery network drawn from this distribution. For each sample, the external assets of each node is uniformly distributed in $[0, 0.2]$, independently across all nodes. All these samples have the same topology and the same distribution for $\mathbf{e}$, but the liabilities are fixed and different. The weight vector is $\mathbf{w} = \mathbf{1}$ and the amount of cash to be injected into the system is $C = 300$.

In PSGD, we set the initial cash injection vector to $\mathbf{c} = \mathbf{0}$. The step size is $\gamma = 1/m$, where m is the iteration number. The stopping condition window is $S = 5$ and the tolerance $\delta = 10^{-5}$.

The optimal solution for such a large-scale and complex network is generally intractable. Thus, we compare our results with the expected value solution, the upper bound defined in Eq. (3.4) and with the expected value EV, the lower bound defined in Eq. (3.3). Moreover, to test the robustness of PSGD, we run the algorithm twice with two different sequences of asset vector $\mathbf{e}$. The results should be close to each other if the performance of PSGD is stable and robust.

To sum up, we compare four results: the expected value EV in Eq. (3.3), two PSGD solutions found with different sequences of asset vectors (testing data) and the expected value solution defined in Eq. (3.4). We generate another 1000 independent asset vectors under the same distribution as test data and use their average weighted

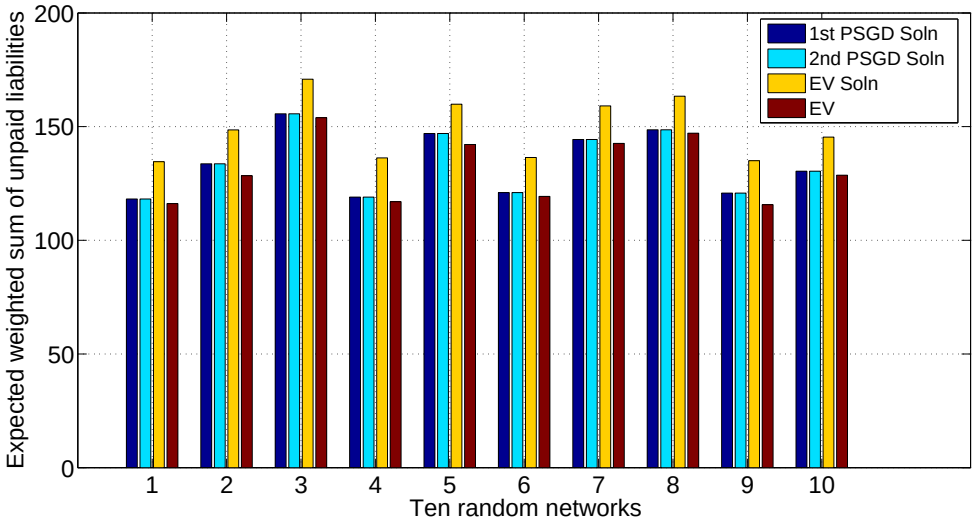


Fig. 3.4. Comparison of the expected value, two PSGD solutions and the expected value solution.

sum of unpaid liabilities to evaluate performance each of the three cash injection vectors.

Finally, we generate ten samples of random core-periphery networks and run PSGD on them. The average weighted sum of unpaid liabilities of the test data for the three cash injection vectors and the expected value EV are shown in Fig 3.4. For all ten networks, we could see that the performance of the two cash injection vectors calculated by PSGD with different random sequences of asset vectors are almost the same, which indicates that PSGD performs stably although it involves random samples. Furthermore, the results from PSGD are better than the results from the expected value solution, performing close to the lower bound EV.

3.6 Conclusions

In this chapter, we consider the situation where the capital of institutions at maturity is a random vector with known distribution. We develop a stochastic linear program to find the cash injection policy that minimizes the expectation of the weighted sum of unpaid liabilities. We discuss some efficient methods to obtain upper bounds and lower bounds for this expectation. For the cases that the bounds are loose, we have proposed two algorithms based on Monte Carlo sampling: Benders decomposition algorithm and projected stochastic gradient descent. We verify the convergence of the projected stochastic gradient descent algorithm by testing it on a five-node network for which the optimal solution can be calculated directly. We also test the performance of the projected stochastic gradient descent algorithm by comparing its result with the upper bound and lower bound on a core-periphery network with comparable size to the US banking system.

4. SUMMARY

In this thesis, we study the optimal monitoring and mitigation of systemic risk in financial networks. For various practical applications, we consider different models, objectives and as a result, adopt different methodologies for the design of proper algorithms.

First, we consider a one-period, single maturity deterministic model, all loans have the same maturity date and the same seniority. Given that the network structure is fixed and known by the regulators, we aim to find the optimal bailout strategies to minimize the weighted sum of unpaid liabilities. With zero bankruptcy cost, we show that this problem is equivalent to a linear program, which can be solved by any standard LP solver efficiently. In addition, we consider three extensions of our model by adding to it various features that characterize real-world lending networks. The first one is allowing the obligations in the network to have multiple seniorities; the second one is incorporating credit default swaps into our model and the third one is assuming the defaulting nodes do not pay at all, i.e., the bankruptcy cost is one hundred percents. We show that with each of these extensions, the problem of minimizing the weighted sum of unpaid liabilities is a mixed-integer linear program.

Second, we investigate how to minimize the number of defaults in the system. We propose two heuristic algorithms: the reweighted ℓ_1 minimization algorithm and the greedy algorithm. We illustrate our algorithms using examples with synthetic data for which the optimal solution can be calculated exactly. We show through numerical simulations that the solutions calculated by the reweighted ℓ_1 algorithm are close to optimal, and that the performance of the greedy algorithm highly depends on the network topology. We also compare these two algorithms using three types of random networks for which the optimal solution is not available.

For the scenarios that centralized gathering all the data of the entire lending network is impractical, we develop a duality-based distributed algorithm to find the optimal cash allocation to minimize the weighted sum of unpaid liabilities when the bankruptcy cost is zero. The algorithm is iterative and is based on message passing between each node and its neighbors. During each iteration of the algorithm, each node only needs to receive a small amount of data from its neighbors, perform simple calculations, and transmit a small amount of data to its neighbors. Simulations verify the convergence and the practicality of the algorithm.

Some applications, such as stress testing, require forecasting and planning for a wide variety of different contingencies. For such applications, we further consider a stochastic model where the external assets of financial institutions are random variables of which the joint distribution is known and fixed. We show that the optimal cash allocation could be calculated by a two-stage stochastic linear program. To solve this problem we develop two algorithms based on Monte Carlo sampling: the Benders decomposition algorithm and the projected stochastic gradient descent algorithm.

REFERENCES

REFERENCES

[1] L. Eisenberg and T. Noe, "Systemic risk in financial systems," *Management Science*, vol. 47, no. 2, pp. 236–249, February 2001.

[2] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1," Mar. 2014, available at cvxr.com/cvx.

[3] ——, "Graph implementations for nonsmooth convex programs," in *Recent Advances in Learning and Control*, ser. Lecture Notes in Control and Information Sciences, V. Blondel, S. Boyd, and H. Kimura, Eds. Springer-Verlag Limited, 2008, pp. 95–110, $stanford.edu/\ boyd/graph_dcp.html$.

[4] E. Candès, M. Wakin, and S. Boyd, "Enhancing sparsity by reweighted ℓ_1 minimization," *Journal of Fourier Analysis and Applications*, vol. 14, no. 5-6, pp. 877–905, 2008.

[5] M. Boss, H. Elsinger, M. Summer, and S. Thurner, "The network topology of the interbank market," 2003, arXiv preprint cond-mat/0309582.

[6] K. Soramäki, M. Bech, J. Arnold, R. Glass, and W. Beyeler, "The topology of interbank payment flows," *Physica A: Statistical Mechanics and its Applications*, vol. 379, no. 1, pp. 317–333, 2007.

[7] B. Craig and G. von Peter, "Interbank tiering and money center banks," *Journal of Financial Intermediation*, vol. 23, pp. 322–347, July 2014.

[8] D. in 't Veld and I. van Lelyveld, "Finding the core: Network structure in interbank markets," *Journal of Banking and Finance*, 2014.

[9] C. Furfine, "Interbank exposures: Quantifying the risk of contagion," *Journal of Money, Credit and Banking*, vol. 35, no. 1, pp. 111–128, February 2003.

[10] H. Elsinger, A. Lehar, and M. Summer, "Using market information for banking system risk assessment," August 22, 2005, available at SSRN: ssrn.com/abstract=787929.

[11] ——, "Risk assessment for banking systems," *Management science*, vol. 52, no. 9, pp. 1301–1314, 2006.

[12] I. van Lelyveld and F. Liedorp, "Interbank contagion in the dutch banking sector: A sensitivity analysis," *International Journal of Central Banking*, pp. 99–133, July 2006.

[13] G. Hałaj, "Contagion effect in banking system—measures based on randomized loss scenarios," *Bank and Credit, Journal of the National Bank of Poland*, vol. 6, pp. 69–80, 2007.

[14] H. Degryse and G. Nguyen, "Interbank exposures: An empirical examination of contagion risk in the Belgian banking system," *International Journal of Central Banking*, pp. 123–171, June 2007.

[15] M. Pröpper, I. van Lelyveld, and R. Heijmans, "Towards a network description of interbank payment flows," May 2008, dNB Working Paper No. 177.

[16] J. M. D. Canedo and S. M. Jaramillo, "A network model of systemic risk: Stress testing the banking system," *Intelligent Systems in Accounting, Finance, and Management*, vol. 16, pp. 97–110, 2009.

[17] S. Martínez-Jaramillo, O. P. Pérez, F. A. Embriz, and F. L. G. Dey, "Systemic risk, financial contagion and financial fragility," *Journal of Economic Dynamics & Control*, vol. 34, pp. 2358–2374, 2010.

[18] P. Gai, A. Haldane, and S. Kapadia, "Complexity, concentration and contagion," *Journal of Monetary Economics*, vol. 58, pp. 453–470, 2011.

[19] K. Anand, P. Gai, S. Kapadia, S. Brennan, and M. Willison, "A network model of financial system resilience," July 2012, bank of England Working Paper No. 458.

[20] G. Hałaj and C. Kok, "Assessing interbank contagion using simulated networks," January 2013, european Central Bank Working Paper Series, 1506.

[21] V. Fourel, J.-C. Héam, D. Salakhova, and S. Tavolaro, "Domino effects when banks hoard liquidity: The French network," April 2013, banque de France Working Paper No. 432.

[22] K. Anand, B. Craig, and G. von Peter, "Filling in the blanks: Network structure and interbank contagion," August 2014, bIS Working Papers, 455.

[23] F. Allen and D. Gale, "Financial contagion," *Journal of Political Economy*, vol. 108, no. 1, pp. 1–33, 2000. [Online]. Available: www.jstor.org/stable/10.1086/262109

[24] R. Cifuentes, G. Ferrucci, and H. Shin, "Liquidity risk and contagion," *Journal of the European Economic Association*, vol. 3, no. 2-3, pp. 556–566, 2005.

[25] J. Lorenz, S. Battiston, and F. Schweitzer, "Systemic risk in a unifying framework for cascading processes on networks," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 71, no. 4, pp. 441–460, 2009.

[26] P. Gai and S. Kapadia, "Contagion in financial networks," *Proceeding of Royal Society A: Mathematical, Physical and Engineering Science*, vol. 466, no. 2120, pp. 2401–2423, August 2010.

[27] H. Amini, R. Cont, and A. Minca, "Resilience to contagion in financial networks," December 1, 2010, available at SSRN: ssrn.com/abstract=1865997.

[28] L. Blume, D. Easley, J. Kleinberg, R. Kleinberg, and E. Tardos, "Which networks are least susceptible to cascading failures?" in *IEEE 52nd Annual Symposium on Foundations of Computer Science*, October 2011, pp. 393–402.

[29] S. Battiston, D. Gatti, M. Gallegati, B. Greenwald, and J. Stiglitz, "Liaisons dangereuses: Increasing connectivity, risk sharing, and systemic risk," *Journal of Economic Dynamics and Control*, vol. 36, no. 8, pp. 1121–1141, 2012. [Online]. Available: www.sciencedirect.com/science/article/pii/S0165188912000899

[30] B. Mundaca, "Optimal bailout during currency and financial crises: A sequential game analysis," September 17, 2002, department of Economics, University of Oslo, www.sv.uio.no/econ/english/research/unpublished-works/working-papers/pdf-files/2002/Memo-27-2002.pdf.

[31] A. Bernardo, E. Talley, and I. Welch, "A model of optimal government bailouts," May 3, 2011, working Paper Series, Berkeley Program in Law and Economics, UC Berkeley, www.law.berkeley.edu/files/bclbe/Model_of_Optimal_Bailouts_0503.pdf.

[32] A. Haldane and R. May, "Systemic risk in banking ecosystems," *Nature*, vol. 469, pp. 351–355, January 20, 2011.

[33] C. Gauthier, A. Lehar, and M. Souissi, "Macroprudential capital requirements and systemic risk," *Journal of Financial Intermediation*, pp. 594–618, 2012.

[34] A. Alter, B. Craig, and P. Raupach, "Centrality-based capital allocations," December 2014, iMF Working Paper.

[35] Z. Li and I. Pollak, "Sparsifying defaults: Optimal bailout policies for financial networks in distress," September 18, 2012, available at SSRN: ssrn.com/abstract=2148483.

[36] ——, "Sparsifying defaults: Optimal bailout policies for financial networks in distress," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, May 2013, pp. 6176–6180.

[37] Z. Li, X. Lin, and I. Pollak, "A distributed algorithm for systemic risk mitigation in financial systems," in *Proceedings of the IEEE Global Conference on Signal and Information Processing*, December 2013, pp. 1137–1137.

[38] S. Pokutta, C. Schmaltz, and S. Stiller, "Measuring systemic risk and contagion in financial networks," *Available at SSRN 1773089*, 2011.

[39] G. Demange, "Contagion in financial networks: a threat index," in *European Summer Symposium in Economic Theory, dev3.cepr.org/meets/wkcn/6/6692/papers/DemangeFinal.pdf*, July 4, 2011.

[40] ——, "Contagion in financial networks: a threat index," December 31, 2011, paris School of Economics Working Paper 2012-02.

[41] ——, "Contagion in financial networks: a threat index," Apr. 2015, working paper or preprint. [Online]. Available: https://halshs.archives-ouvertes.fr/halshs-00662513

[42] L. Rogers and L. Veraart, "Failure and rescue in an interbank network," September 23, 2011, available at SSRN: ssrn.com/abstract=1932911.

[43] ——, "Failure and rescue in an interbank network," *Management Science*, vol. 59, no. 4, pp. 882–898, 2013.

[44] A. Capponi and P.-C. Chen, "Systemic risk mitigation in financial networks," July 13, 2013, available at SSRN: ssrn.com/abstract=2293426.

[45] P. Glasserman and H. Young, "How likely is contagion in financial networks?" *Journal of Banking & Finance*, vol. 50, pp. 383–399, January 2015.

[46] M. ApS, *The MOSEK optimization toolbox for MATLAB manual. Version 7.1 (Revision 28).*, 2015. [Online]. Available: http://docs.mosek.com/7.1/toolbox/index.html

[47] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2015. [Online]. Available: http://www.gurobi.com

[48] M. Elliott, B. Golub, and M. Jackson, "Financial networks and contagion," 2013, available at SSRN: ssrn.com/abstract=2175056.

[49] M. Farboodi, "Intermediation and voluntary exposure to counterparty risk," August 30, 2014, working Paper, University of Chicago.

[50] C. Floudas, *Nonlinear and mixed-integer optimization: fundamentals and applications*. Oxford University Press, 1995.

[51] X. Lin and N. Shroff, "Utility maximization for communication networks with multipath routing," *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 766–781, 2006.

[52] D. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation*. Old Tappan, NJ (USA); Prentice Hall Inc., 1989.

[53] X. Lin and N. Shroff, "Utility maximization for communication networks with multi-path routing," Technical Report, Purdue University, https://engineering.purdue.edu/ linx/papers.html, Tech. Rep., 2004.

[54] J. Benders, "Partitioning procedures for solving mixed-variables programming problems," *Numerische mathematik*, vol. 4, no. 1, pp. 238–252, 1962.

[55] L. Bottou, "Online learning and stochastic approximations," in *On-line learning in neural networks*. Cambridge University Press, 1998, pp. 9–42.

[56] J. R. Birge and F. Louveaux, *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[57] Y. Ye, "An $o(n^3l)$ potential reduction algorithm for linear programming," *Mathematical Programming*, vol. 50, no. 1-3, pp. 239–258, 1991. [Online]. Available: dx.doi.org/10.1007/BF01594937

[58] R. V. Slyke and R. Wets, "L-shaped linear programs with applications to optimal control and stochastic programming," *SIAM Journal on Applied Mathematics*, vol. 17, no. 4, pp. 638–663, 1969.

[59] G. Infanger, "Monte carlo (importance) sampling within a benders decomposition algorithm for stochastic linear programs," *Annals of Operations Research*, vol. 39, no. 1, pp. 69–95, 1992.

[60] D. P. Palomar and J. R. Fonollosa, "Practical algorithms for a family of water-filling solutions," *Signal Processing, IEEE Transactions on*, vol. 53, no. 2, pp. 686–695, 2005.

[61] J. Yang and S. Roy, "On joint transmitter and receiver optimization for multiple-input-multiple-output (mimo) transmission systems," *Communications, IEEE Transactions on*, vol. 42, no. 12, pp. 3221–3231, 1994.

[62] E. N. Onggosanusi, A. M. Sayeed, and B. D. Van Veen, "Efficient signaling schemes for wideband space-time wireless channels using channel state information," *Vehicular Technology, IEEE Transactions on*, vol. 52, no. 1, pp. 1–13, 2003.

[63] N. Amitay and J. Salz, "Linear equalization theory in digital data transmission over dually polarized fading radio channels," *AT&T Bell Laboratories Technical Journal*, vol. 63, no. 10, pp. 2215–2259, 1984.

[64] H. Sampath, P. Stoica, and A. Paulraj, "Generalized linear precoder and decoder design for mimo channels using the weighted mmse criterion," *Communications, IEEE Transactions on*, vol. 49, no. 12, pp. 2198–2206, 2001.

[65] D. P. Palomar and M. A. Lagunas, "Joint transmit-receive space-time equalization in spatially correlated mimo channels: a beamforming approach," *Selected Areas in Communications, IEEE Journal on*, vol. 21, no. 5, pp. 730–743, 2003.

APPENDICES

A. COMPARISON OF THREE ALGORITHMS FOR COMPUTING THE CLEARING PAYMENT VECTOR

A.1 Proportional Payment Mechanism

In [1], zero bankruptcy costs are assumed, and three methods of finding the clearing payment vector are proposed: a fixed-point algorithm, the fictitious default algorithm and an optimization method. In this section, we first introduce and analyze these three methods and then compare their computation times under different network topologies.

A.1.1 Fixed-Point Algorithm

By definition, the clearing payment vector is a fixed point of the following map:

$$\Phi(\mathbf{p}) = \min\{(\Pi^T \mathbf{p} + \mathbf{e}), \bar{\mathbf{p}}\}.$$

where the minimum of the two vectors is component-wise. Under certain mild assumptions specified in [1], the fixed point is unique. It can be found iteratively via the following algorithm [1].

Fixed-point algorithm:

1. Initialization: set $\mathbf{p}^0 \leftarrow \bar{\mathbf{p}}$, $k \leftarrow 0$, and set the stopping tolerance δ_0 to a small positive number based on the accuracy requirement.
2. $\mathbf{p}^{k+1} \leftarrow \Phi(\mathbf{p}^k)$.
3. If $\|\mathbf{p}^{k+1} - \mathbf{p}^k\|_\infty < \delta_0$, stop and output the clearing payment vector $\mathbf{p}^{k+1}$; else, set $k \leftarrow k + 1$ and go to Step 2.

At each iteration, the computational complexity is dominated by $\Pi^T\mathbf{p}$, which is $\Theta(N^2)$. The number of iterations is highly dependent on the network topology and the amounts of liabilities.

A.1.2 Fictitious Default Algorithm

The fictitious default algorithm is proposed in Section 3.1 in [1]. The basic idea is to first assume that all the nodes pay their liabilities in full. If, under this assumption, every node has enough funds to pay in full, then the algorithm terminates. If some nodes do not have enough funds to pay in full, it means that these nodes would default even if all the other nodes pay in full. Such defaults that are identified during the first iteration of the algorithm are called *first-order defaults*. In the second iteration, we assume that only the first-order defaults occur. Every non-defaulting node k pays in full, i.e., $p_k = \bar{p}_k$; every defaulting node i pays all its available funds, i.e., $p_i = \sum_{j=1}^{N} \Pi_{ji} p_j + e_i$. If there is no new defaulting nodes during this second iteration, then the algorithm is terminated. Otherwise, the new defaulting nodes are called *second-order defaults*, and we proceed to the third iteration. In the third iteration we assume that both the first-order and second-order defaults occur. We calculate the new payment vector and again check the set of defaulting nodes. We keep iterating until no new defaults occur. Since there are N nodes in the system, this algorithm is guaranteed to terminate within N iterations. The specifics of the fictitious default algorithm are as follows.

Fictitious default algorithm:

1. Initialization: $\mathbf{p}^1 \leftarrow \bar{\mathbf{p}}$, $k \leftarrow 1$, and $D^{(0)} \leftarrow \varnothing$.
2. For all nodes i, compute the difference between their incoming payments and their obligations:
$$v_i^{(k)} \leftarrow \sum_{j=1}^{N} \Pi_{ji} p_j^{(k)} + e_i - \bar{p}_i$$

3. Define $D^{(k)}$ as the set of defaulting nodes:

$$D^{(k)} = \left\{ i : \ v_i^{(k)} < 0 \right\}.$$

4. If $D^{(k)} = D^{(k-1)}$, terminate.

5. Otherwise, set $p_i^{(k+1)} \leftarrow \bar{p}_i$ for all $i \notin D^{(k)}$. For all $i \in D^{(k)}$, compute the payments $p_i^{(k+1)}$ by solving the following system of equations:

$$p_i^{(k+1)} = e_i + \sum_{j \in D^{(k)}} \Pi_{ji} p_j^{(k+1)} + \sum_{j \notin D^{(k)}} \Pi_{ji} \bar{p}_j, \text{ for all } i \in D^{(k)}$$

6. Set $k \leftarrow k+1$ and go to Step 2.

At each iteration of the fictitious default algorithm, the computational complexity is dominated by solving the linear equations in Step 5. The number of unknowns in these equations and the number of equations are both equal to the number of elements in $D^{(k)}$. In the worst case, the number of defaulting nodes in the system is of the same order as N. In this case the computational complexity per iteration is $O(N^3)$ [57]. Compared to the fixed-point algorithm, the fictitious default algorithm has a larger computational complexity per iteration, and, as shown below in Section A.1.4, larger running times on several network topologies. However, the advantage of the fictitious default algorithm is that it is guaranteed to terminate within N iterations. Moreover, the fictitious default algorithm will produce the exact value of clearing payment, unlike the fixed-point algorithm which produces an approximation.

Table A.1.
Comparison of the running times for the computation of the clearing payment vector under the proportional payment mechanism using the fixed-point algorithm, fictitious default algorithm, and linear programming.

	FP algorithm		FD algorithm		LP method	
	ave (s)	stdev	ave (s)	stdev	ave (s)	stdev
fully connected	0.9128	0.1045	10.7341	0.7182	53.1725	11.8947
core-periphery	0.0869	0.0342	7.8213	1.2843	0.1964	0.0507
linear chain	0.0462	0.0170	10.2574	1.0211	0.1610	0.0449

A.1.3 Linear Programming Method

Define $f(\mathbf{p}) = \sum_{i=1}^{N} p_i = \mathbf{1}^T\mathbf{p}$, which is a strictly increasing function of $\mathbf{p}$. By Lemma 4 in [1], the clearing payment vector can be obtained via the following linear program:

$$\begin{aligned} &\max_{\mathbf{p}} \mathbf{1}^T\mathbf{p} \\ &\text{subject to} \\ &\mathbf{0} \le \mathbf{p} \le \bar{\mathbf{p}}, \\ &\mathbf{p} \le \Pi^T\mathbf{p} + \mathbf{e}. \end{aligned} \tag{A.1}$$

The computational complexity of solving an LP is $O(N^3)$ [57].

A.1.4 Comparison of Running Times on Three Different Topologies

We calculate the clearing payment vector via the above three methods on three different network topologies and compare the running times. The first network topology is a fully connected network with 1000 nodes. All the obligation amounts L_{ij} and asset amounts e_i are independent random variables, uniformly distributed in $[0, 1]$. The second network topology is a core-periphery network shown in Fig. 1.17. It con-

tains 15 fully connected core nodes. Each core node has 70 periphery nodes. Each periphery node has a single link pointing to the corresponding core node. All the obligation amounts $L_{i,j}$ are independent uniform random variables. For each pair of core nodes i and j the obligation amount L_{ij} is uniformly distributed in $[0, 10]$. For a core node i and its periphery node k, the obligation amount L_{ki} is uniformly distributed in $[0, 1]$. The asset amounts e_i are uniformly distributed in $[0, 0.25]$. The third network topology is a long linear chain network with 1000 nodes. For $i = 1, 2, \ldots, N-1$, the obligation amount $L_{i(i+1)}$ is uniformly distributed in $[0, 10]$, and for other pairs of i and j, $L_{ij} = 0$. The asset amounts e_i are uniformly distributed in $[0, 1]$.

For each type of network, we generate 100 samples. We run the Matlab code on a personal computer with a 2.66GHz Intel Core2 Duo Processor P8800. The average running times and the sample standard deviations of the running times for the three methods are shown in Table A.1. For all three types of networks, the fixed-point algorithm is the most efficient one. Note that the computation time of linear program method is highly variable because simpler topologies result in Π being a sparse matrix, reducing the running time.

A.2 All-or-Nothing Payment Mechanism

A.2.1 Fixed-Point Algorithm and Fictitious Default Algorithm

We now assume the all-or-nothing payment mechanism where node i pays $\bar{p}_i$ if it is solvent and pays nothing if it defaults. Therefore, the clearing payment vector is a fixed point of the map Ψ defined as follows:

$$\Psi_i(\mathbf{p}) = \begin{cases} \bar{p}_i & \text{if } \sum_{j=1}^{N} \Pi_{ji} p_j + e_i \geq \bar{p}_i \\ 0 & \text{otherwise} \end{cases}$$

We find the fixed point of $\Psi(\cdot)$ iteratively via the following algorithm.

Fixed-point algorithm:

1. Initialization: set $\mathbf{p}^0 \leftarrow \bar{\mathbf{p}}$, $k \leftarrow 0$.

2. $\mathbf{p}^{k+1} \leftarrow \Psi(\mathbf{p}^k)$.

3. If $\mathbf{p}^{k+1} = \mathbf{p}^k$, stop and output the clearing payment vector $\mathbf{p}^{k+1}$; else, set $k \leftarrow k+1$ and go to Step 2.

In fact, under the all-or-nothing payment mechanism, this fixed-point algorithm can be interpreted as the following fictitious default algorithm. We initially assume that all the nodes pay their liabilities in full, i.e., $\mathbf{p}^0 = \bar{\mathbf{p}}$. If, under this assumption, every node has enough funds to pay in full, then the algorithm terminates. If some nodes do not have enough funds to pay in full, it means that these nodes would default even if all the other nodes pay in full. We define these nodes as *first-order defaults*. With function $\Psi(\cdot)$, we identify the first-order defaults and set their payments to zero. In the second iteration, we assume that only the first-order defaults occur. Every non-defaulting node k pays in full, i.e., $p_k = \bar{p}_k$; every defaulting node i pays 0, i.e., $p_i = 0$. Again, with function $\Psi(\cdot)$, we identify the new defaulting nodes, which are called *second-order defaults*, and set their payments to zero. If there are no such new defaulting nodes, the algorithm terminates; otherwise, we proceed to the third iteration. We keep iterating until no new defaults occur, i.e., $\mathbf{p}^{k+1} = \mathbf{p}^k$.

Since there are N nodes in the system, this algorithm is guaranteed to terminate within N iterations. At each iteration, the computational complexity is dominated by $\Pi^T\mathbf{p}$, which is $\Theta(N^2)$. Therefore, the computational complexity of the fixed-point algorithm (fictitious default algorithm) is $O(N^3)$.

A.2.2 Mixed-Integer Linear Programming Method

The clearing payment vector can also be obtained by solving MILP (1.49) with the assumption that no external cash would be injected, i.e., $C = 0$. With $C = 0$, MILP (1.49) is simplified to the following MILP:

$$\begin{aligned} &\max_{\mathbf{p},\mathbf{c},\mathbf{d}} \mathbf{w}^T\mathbf{p} \qquad (A.2)\\ &\text{subject to}\\ &p_i = \bar{p}_i(1-d_i), \text{ for } i = 1, 2, \cdots, N,\\ &\bar{p}_i - \sum_{j=1}^{N} \Pi_{ji} p_j - e_i \leq \bar{p}_i d_i, \text{ for } i = 1, 2, \cdots, N,\\ &d_i \in \{0, 1\}, \text{ for } i = 1, 2, \cdots, N. \end{aligned}$$

We solve MILP (A.2) via CVX [2, 3].

A.2.3 Comparison of Running Times on Three Different Topologies

We calculate the clearing payment vector under the all-or-nothing payment mechanism via the above two methods on three network topologies described in Section A.1.4, and compare the running times.

Similar to Section A.1.4, we generate 100 samples and run the Matlab code on a personal computer with a 2.66GHz Intel Core2 Duo Processor P8800. The average running times and the sample standard deviations of the running times for the two methods are shown in Table A.2. For all the three topologies, the fixed-point algorithm is significantly more efficient than the MILP method.

Table A.2.
Comparison of the running times for the computation of the clearing payment vector under the all-or-nothing payment mechanism using the fixed-point algorithm and mixed-integer linear programming.

	FP algorithm / FD algorithm		MILP	
	ave (s)	stdev	ave (s)	stdev
fully connected	0.0092	0.0129	1.2204	0.0909
core-periphery	0.0242	0.0175	0.5338	0.0255
linear chain	0.0279	0.0149	0.4700	0.0276

B. CVX CODE

B.1 CVX code for MILP (1.11)

Below is the CVX code to solve MILP (1.11). The parameters that appear in the code are defined in Table B.1.

```
cvx_solver mosek
cvx_begin
    variable u(K, N)
    variable c(N) % cash injection vector
    variable d(K, N) binary
    maximize(sum(w'.*sum(u.*sum(L, 3), 1)))
    subject to
        sum(c) <= C_total
        c >= 0
        0 <= u <= 1
        for i = 1:N
            sum(sum(u.*L(:,:,i))) + e(i) + c(i) >= ...
            sum(u(:,i).* sum(reshape(L(:,i,:), K, N),2))
        end
        d <= u
        for i = 1:K-1
            u(i,:) <= d(i+1,:)
        end
cvx_end
```

Table B.1.
Parameters in CVX codes for MILP (1.11).

Parameters in CVX codes	Notation in Section 1.4
L	L
u(k, i)	$p_i^k / \bar{p}_i^k$
e	$\mathbf{e}$
c	$\mathbf{c}$
w	$\mathbf{w}$
d(k, i)	d_i^k
C_total	C

B.2 CVX code for MILP (1.35)

Below is the CVX code to solve MILP (1.35). The parameters that appear in the code are defined in Table B.2.

```
cvx_solver gurobi
cvx_begin
    variable x(N)
    variable y(Nc)
    variable z(N)
    variable ys(Nc*N)
    variable f(N)
    variable r(N)
    variable d(N) binary
    variable d_cds(Nc) binary
    variable c(N)
    maximize sum(sum(L, 2).*(x+z)) + ...
    sum(sum(D,2).*ys) ...
    - sum(reshape(sum(D,2),Nc,N),1)*d
    subject to
        % constraints for cash injection vector
        sum(c) <= C_total
        c >= 0
        % constraints in Step 1
        0 <= x <= 1
        sum(L, 2) .* x <= L' * x + e + c
        L' * x + e + c - sum(L, 2) .* x <= (1 - d) / eps
        1 - d <= x
        1 - x >= eps * d
```

```
        % constraints in Step 2
        f == L' * x + e + c - sum(L, 2) .* x
        for i = 1:Nc
            ys((0:N-1)*Nc+i) <= d
            y(i) - ys((0:N-1)*Nc+i) <= 1 - d
            ys((0:N-1)*Nc+i) - y(i) <= 1 - d
        end
        sum(reshape(sum(D,2).*ys,Nc,N),2) <= ...
        D' * ys + f(1:Nc)
        D' * ys + f(1:Nc) - ...
        sum(reshape(sum(D,2).*ys,Nc,N),2) ...
        <= (1 - d_cds) / eps
        r(1:Nc) == D' * ys + f(1:Nc) - ...
        sum(reshape(sum(D,2).*ys,Nc,N),2)
        r(Nc+1:N) == f(Nc+1:N)
        1 - d_cds <= y
        % constraints in Step 3
        0 <= z <= 1 - x
        sum(L, 2) .* z <= L' * z + r
cvx_end
```

Table B.2.
Parameters in CVX codes for MILP (1.35).

Parameters in CVX codes	Notation in Section 1.5
N	N
Nc	number of core nodes
eps	ϵ
x	$\mathbf{x}$
y	$\mathbf{y}$
z	$\mathbf{z}$
ys(i,l)	y_i^l
f(i)	f_i
r(i)	$r(i)$
L	L
D(i+Nc*(l-1),j)	D_{ij}^l
e	$\mathbf{e}$
c	$\mathbf{c}$
d	$\mathbf{d}$
d_cds	$\mathbf{d}^{CDS}$
C_total	C

B.3 CVX code for MILP (1.41)

Below is the CVX code to solve MILP (1.41). The parameters that appear in the code are defined in Table B.3.

```
cvx_solver gurobi
cvx_begin
    variable d(N) binary
    variable c(N)
    variable p(N)
    maximize( w' * p - v' * d )
    subject to
        ones(1,N) * c <= C_total
        c >= 0
        p <= Pi' * p + e + c
        p >= 0
        p <= pbar
        pbar - p <= diag(pbar) * d
cvx_end
```

Table B.3.
Parameters in CVX codes for MILP (1.41).

Parameters in CVX codes	Notation in Section 1.7
pbar	$\bar{\mathbf{p}}$
Pi	Π
e	$\mathbf{e}$
c	$\mathbf{c}$
w	$\mathbf{w}$
v	$\mathbf{v}$
d	$\mathbf{d}$
I	I_N
C_total	C

Table B.4.
Parameters in CVX codes for MILP (1.49).

Parameters in CVX codes	Notation in Section 1.8
pbar	$\bar{\mathbf{p}}$
Pi	Π
e	$\mathbf{e}$
c	$\mathbf{c}$
w	$\mathbf{w}$
d	$\mathbf{d}$
I	I_N
C_total	C

B.4 CVX code for MILP (1.49)

Below is the CVX code to solve MILP (1.49). The parameters that appear in the code are defined in Table B.4.

```
cvx_solver mosek
cvx_begin
    variable d(N) binary % default indicator
    variable c(N) % cash injection vector
    minimize( pbar' * diag(w) * d )
    subject to
        ones(1,N) * c <= C_total;
        c >= 0;
        (Pi'-I) * diag(pbar) * d <= Pi' * pbar + e + c - pbar;
cvx_end
```

VITA

VITA

Zhang Li received his M.S. and B.S. in Electrical Engineering from Shanghai Jiao Tong University, Shanghai, China in 2012 and 2010. Since 2012, he has been working towards the Ph.D. degree in Electrical and Computer Engineering at Purdue University, West Lafayette, IN, USA, co-advised by Prof. Ilya Pollak and Prof. Borja Peleato. In 2015, he spent ten weeks in Akuna Capital, working as a quantitative analyst intern. His research interests include optimization over financial networks, systemic risk, portfolio optimization, machine learning and data analysis.