Optimal Monitoring and Mitigation of Systemic Risk in Financial Networks

Zhang Li, Borja Peleato-Inarrea, Ben Craig, and Ilya Pollak

Abstract—This paper studies the problem of optimally allocating a cash injection into a financial system in distress. 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 asset values are fixed and that there are no bankruptcy costs, we show that this problem is equivalent to a linear program. We show that if the defaulting nodes never pay anything, the optimal cash injection allocation problem is an NP-hard mixed-integer linear program. However, modern optimization software enables the computation of very accurate solutions to this problem on a personal computer in a few seconds for network sizes comparable with the size of the US banking system. In addition, we address the problem of allocating the cash injection amount so as to minimize the number of nodes in default. For this problem, we develop two heuristic algorithms: a reweighted \( \ell_1 \) minimization algorithm and a greedy algorithm. We illustrate these two algorithms using three synthetic network structures for which the optimal solution can be calculated exactly. We also compare these two algorithms on three types random networks which are more complex. 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.

Index Terms—financial networks, systemic risk, financial systems, optimal resource allocation

I. 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, Lagrangian, 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 [18], we construct algorithms for computing exact solutions for Problem I and its Lagrangian variant, by showing in Section III that both formulations are equivalent to linear programs under the payment scheme assumed in [18].

We show in Section VI that under the all-or-nothing payment scheme where the defaulting nodes do not pay at all, Problem I is an NP-hard mixed-integer linear program. However, we show through simulations that use optimization package CVX [30], [29] 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.

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: a reweighted \( \ell_1 \) minimization approach inspired by [10] 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 \( \ell_1 \) 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 $\ell_1$ minimization; and in the third example the reweighted $\ell_1$ 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 V that this problem is equivalent to a mixed-integer linear program.

### A. 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 [9], [44], [14], [34]. Real data informs the new approaches for assessing systemic financial stability of banking systems developed in [24], [20], [21], [45], [31], [15], [41], [11], [39], [25], [5], [32], [23], [4].

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 [1], [13], [38], [26], [3], [8], [6]. The relationships between the probability of a systemic failure and the average connectivity in the network are investigated in [26], [6], [1]. Other features, such as the distribution of degrees and the structure of the subgraphs of contagious links, are examined in [3].

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 [40] and [7] 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 [13], [33], [27], [2] 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. Our present paper extends our earlier work reported in [36], [37], [35]. In addition to ours, several other papers have recently considered cash injection policies for lending networks [16], [17], [42], [43], [12], all based on the framework proposed in [18].

A cash injection targeting policy is developed in [16], [17] 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 defined as the derivative of the unpaid liability with respect to the current asset value. However, extending this idea to construct an optimal cash injection policy for non-infinitesimal cash amounts produces an inefficient algorithm, as we show in Section III-B. We show that our own method proposed in Section III-A is more efficient.

In [42], [43], bankruptcy costs are incorporated into the model of [18]. 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 [18] is proposed in [12], 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, [12] does not describe an algorithm for solving this problem.

Another related work is [28]. Based on the clearing payment framework in [18], the authors of [28] study the probability of contagion and amplification of losses due to network effects when the system suffers a random shock.

### B. Outline of the Paper

The paper is organized as follows. Section II describes the model of financial networks, the clearing payment mechanism, and the notation. Section III shows that if each defaulting node pays its creditors in proportion to the owed amounts, then Problem I and its Lagrangian formulation are equivalent to linear programs. Two heuristic algorithms are developed in Section IV to solve Problem II under the proportional payment mechanism: a reweighted $\ell_1$ minimization algorithm and a greedy algorithm. Problem III is considered in Section V. Section VI 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 program and show that can be efficiently solved using modern optimization software for network sizes comparable to the size of the US banking system.

### II. 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;
- $0$, $1$, $e$, $c$, $p$, $w$, $v$, and $d$ are all vectors in $\mathbb{R}^N$ defined in Table I;
- $W = w^T(\bar{p} - p)$ 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, $p_i < \bar{p}_i$;
- $\Pi_{ij}$ is what node $i$ owes to node $j$, as a fraction of the total amount owed by node $i$,
  \[ \Pi_{ij} = \begin{cases} \frac{L_{ij}}{\bar{p}_i} & \text{if } \bar{p}_i \neq 0, \\ 0 & \text{otherwise}; \end{cases} \]

- $\Pi$ and $L$ are the matrices whose entries are $\Pi_{ij}$ 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 [18], 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, 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 $\sum_{j=1}^{N} \Pi_{ij}p_j$.

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 $[0, \bar{p}]$. Under the all-or-nothing payment scenario, the defaulting nodes do not pay at all.

**All-or-nothing payment mechanism:**

- If $i$’s total funds are at least as large as its liabilities (i.e., $\sum_{j=1}^{N} \Pi_{ij}p_j + e_i + c_i \geq \bar{p}_i$) then all $i$’s creditors get paid in full.
- If $i$’s total funds are smaller than its liabilities, then $i$ pays nothing.

As defined in [18], a clearing payment vector $p$ is a vector of borrower-to-lender payments that is consistent with the conditions of the payment mechanism.

In this paper, 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.

### III. Optimal Solution to Problem I Under the Proportional Payment Mechanism

#### A. Minimizing the Weighted Sum of Unpaid Liabilities is a Linear Program

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

\[ W = w^T(\bar{p} - p), \]

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

\[ 1^T c \leq C. \]

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

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

**Proof:** First, note that since $w$ and $\bar{p}$ do not depend on $p$ or $c$, minimizing $W$ is equivalent to maximizing $w^T p$. With a fixed cash injection vector $e$, the financial system is equivalent to $(\Pi, \bar{p}, e + c)$. Since $w > 0$, we have that $w^T \bar{p}$ is a strictly increasing function of $p$. By Lemma 4 in [18], the clearing payment vector $\bar{p}$ can be obtained by solving the following linear program:

\[
\begin{align*}
\max_{\bar{p}} & \quad w^T \bar{p} \\
\text{subject to} & \quad 0 \leq \bar{p} \leq \bar{p}, \\
& \quad \bar{p} \leq \Pi^T p + e + c.
\end{align*}
\]

### TABLE I

**Notation for Several Vector Quantities.**

<table>
<thead>
<tr>
<th>VECTOR</th>
<th>i-TH COMPONENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0$</td>
<td>$0$</td>
</tr>
<tr>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>$e &gt; 0$</td>
<td>net external assets at node $i$ before cash injection</td>
</tr>
<tr>
<td>$c \geq 0$</td>
<td>external cash injection to node $i$</td>
</tr>
<tr>
<td>$p$</td>
<td>the amount node $i$ owes to all its creditors</td>
</tr>
<tr>
<td>$p \leq \bar{p}$</td>
<td>the total amount node $i$ actually repays all its creditors on the due date of the loans</td>
</tr>
<tr>
<td>$p - \bar{p}$</td>
<td>node $i$’s total unpaid liabilities</td>
</tr>
<tr>
<td>$d$</td>
<td>indicator variable of whether node $i$ defaults, i.e., $d_i = 1$ if node $i$ defaults; $d_i = 0$ otherwise</td>
</tr>
</tbody>
</table>

Table I Notation for Several Vector Quantities.
From Theorem 1 in [18], there exists a greatest clearing payment vector $p^*$. Since $W$ is a strictly increasing function of $p$, $p^*$ is a solution of LP (1-3). For any other $p \neq p^*$, we have $p_i \leq p_i^*$ for $i = 1, 2, \ldots, N$ and at least one of these inequalities is strict. Thus, $w^*T p < w^T p^*$. Therefore $p^*$ is the unique solution of LP (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 $e$, the weight vector $w$, 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 $c \geq 0$ to minimize the weighted sum of unpaid liabilities $W = w^T(p - p)$ subject to the budget constraint $1^T c \leq C$. A solution to this problem can be obtained by solving the following linear program:

$$\begin{align*}
\max_{p, c} & \quad w^T p \\
\text{subject to} & \quad 1^T c \leq C, \\
& \quad c \geq 0, \\
& \quad 0 \leq p \leq \bar{p}, \\
& \quad p \leq \Pi^T p + e + c.
\end{align*}$$

(4-6)

**Proof:** Since the constraints on $c$ and $p$ in LP (4-6) form a closed and bounded set in $\mathbb{R}^{2N}$, a solution exists. Moreover, for any fixed $c$, it follows from our Lemma 1 and Lemma 4 in [18] that the linear program has a unique solution for $p$ which is the clearing payment vector for the system.

Let $(p^*, c^*)$ be a solution to (4-6). Suppose that there exists a cash injection allocation that leads to a smaller cost $W$ than does $c^*$. In other words, suppose that there exists $c' \geq 0$, with $1^T c' \leq C$, such that the corresponding clearing payment vector $p'$ satisfies $w^T(p' - p') < w^T(p' - p^*)$, or, equivalently,

$$w^T p^* < w^T p.'$$

Note that $c'$ satisfies the first two constraints of (4-6). Moreover, since $p'$ is the corresponding clearing payment vector, the last two constraints are satisfied as well. The pair $(p^*, c')$ is thus in the constraint set of our linear program. Therefore, Eq. (7) contradicts the assumption that $(p^*, c^*)$ is a solution to (4-6). This completes the proof that $c^*$ is the allocation of $C$ that achieves the smallest possible cost $W$.

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

$$\begin{align*}
\max_{C, c, p} & \quad w^T p - \lambda C \\
\text{subject to} & \quad 1^T c = C, \\
& \quad c \geq 0, \\
& \quad 0 \leq p \leq \bar{p}, \\
& \quad p \leq \Pi^T p + e + c.
\end{align*}$$

(8)

This equivalence follows from Theorem 1: denoting a solution to (8) by $(C^*, p^*, c^*)$, we see that the pair $(p^*, c^*)$ must be a solution to (4-6) for $C = C^*$. At the same time, the fact that $C^*$ maximizes the objective function in (8) means that it minimizes $\lambda C + W = \lambda C + w^T(p - \bar{p})$, since $\bar{p}$ is a fixed constant.

**B. Comparison with Demange’s Algorithm**

A cash injection targeting policy is developed in [16], [17] for an infinitesimally small amount of the injected cash. The basic idea of Proposition 4 in [17] is to inject the cash into the node with the largest threat index, which is defined as the derivative of the sum of the unpaid liabilities with respect to the current asset value. Moreover, as small amounts of cash are gradually injected, the target remains the same until at least one node is fully rescued (i.e., changes from defaulting to solvent), so the optimal policy for non-infinitesimal amounts of cash would be to keep injecting cash into the same node until one node changes its state. While no algorithm for the injection of a non-infinitesimal amount of cash (i.e., for our Problem I) is proposed in [16], [17], we construct such an algorithm based on the ideas from [16], [17].

**Algorithm for Problem I based on [16], [17]:**

1) Initialization: set cash injection vector $c \leftarrow 0$ and the remaining cash still to be allocated $C_r \leftarrow C$.

2) Compute the clearing payment vector $p$ for system $(\Pi, p, e + c)$.

3) Compute the threat index for system $(\Pi, p, e + c)$ by solving the linear program (13) in [17]. Select the one with the largest threat index, denoted as node $i_0$.

4) Inject a small amount of cash $\Delta c$ into node $i_0$ and update the clearing payment vector $p'$. Define $\Delta p = p' - p$ as the increase of the payment vector after injecting $\Delta c$ into node $i_0$.

5) Compute $\frac{\bar{p}_i - \bar{p}_i}{\bar{p}_i}$ for $i = 1, 2, \ldots, N$. Select the smallest one, denoted as node $i_1$. Then node $i_1$ will be the first node that changes from defaulting to being solvent when we keep injecting cash into node $i_0$.

6) Set $c_{i_0} \leftarrow c_{i_0} + \min \left\{ C_r, \frac{\Delta c}{\bar{p}_i} \frac{\bar{p}_i - \bar{p}_i}{\Delta p_i} \right\}$. Set $C_r \leftarrow C_r - \min \left\{ C_r, \frac{\Delta c}{\bar{p}_i} \frac{\bar{p}_i - \bar{p}_i}{\Delta p_i} \right\}$. If $C_r = 0$, stop; otherwise, go to Step 2.

Each iteration of this algorithm computes the clearing payment vector twice: in Steps 2 and 4. Step 3 moreover involves solving a linear program to obtain the threat index. In the worst case, the algorithm will stop after $N$ iterations since at
each iteration, only one defaulting node is guaranteed to be rescued. Thus, we would need to solve \( N \) LPs and compute the clearing payment vector \( 2N \) times in the worst case—much less computationally efficient than our approach of Theorem 1 which requires solving a single LP. Note that the above algorithm makes a simplifying assumption in Step 4 that a small number \( \Delta \epsilon \) can be found in advance such that the injection of \( \Delta \epsilon \) in Step 4 does not lead to the rescue of any nodes. Our algorithm based on Theorem 1 does not require this simplifying assumption.

IV. 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 \( \epsilon \) to minimize the number of defaults \( N_d \), i.e., the number of nonzero entries in the vector \( \epsilon - p \).

In this section, we propose two heuristic algorithms to solve Problem II approximately. First, we adapt the reweighted \( \ell_1 \) minimization strategy approach from Section 2.2 of [10]. Our algorithm solves a sequence of weighted versions of the linear program (4-6), with the weights designed to encourage sparsity of \( \epsilon - p \). In the following pseudocode of our algorithm, \( w^{(m)} \) is the weight vector during the \( m \)-th iteration.

**Reweighted \( \ell_1 \) minimization algorithm:***

1) \( m = 0 \).
2) Select \( w^0 \) (e.g., \( w^0 \leftarrow 1 \)).
3) Solve linear program (4-6) with objective function replaced by \( w^{(m)} \epsilon - p \).
4) Update the weights: for each \( i = 1, \ldots, N \),
   \[
   w_i^{(m+1)} \leftarrow \frac{1}{\exp \left( \frac{\bar{p}_i - p_i^{(m)}}{\epsilon} \right) - 1 + \epsilon},
   \]
   where \( \epsilon > 0 \) is constant, and \( p^{(m)} \) is the clearing payment vector obtained in Step 3.
5) If \( \| w^{(m+1)} - w^{(m)} \|_1 < \delta \), 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:**

1) \( C_r \leftarrow C \), \( c \leftarrow 0 \), \( w \leftarrow 1 \).
2) Solve linear program (1-3) to obtain the clearing payment vector \( p \).
3) Calculate the surplus of each node after clearing: \( r \leftarrow \Pi^T p + e + c - p \).
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 \( i = 1, 2, \ldots, N \).
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.

A. 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, levels 0 and \( S - 1 \) correspond to the root and the leaves, respectively. Every node at level \( s < S - 1 \) owes \( 2^{s} \) to each of its two creditors (children). We set \( e = 0 \).

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
where an and owes six nodes each. The nodes in the cycles shown in Fig. 3. The network contains unpaid liabilities. For cash into the nodes at level $S$, this would prevent the defaults of this node and all its $2^{S-1} - 2$ 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:

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

(9)

In our test, we set $S = 10$. The green line in Fig. 2 is a plot of the minimum number of defaults as a function of $C$ from Eq. (9). The blue line is the solution calculated by the reweighted $\ell_1$ minimization algorithm with $\epsilon = 0.001$ and $\delta = 10^{-6}$. The algorithm was run using six different initializations: five random ones and $w^{(0)} = 1$. 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. 2, the results of the reweighted $\ell_1$ minimization algorithm are very close to the optimal for the entire range of $C$. 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 $C \geq 16$, 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.

B. Example: A Network with Cycles

Second, we test our algorithms on a network with cycles shown in Fig. 3. 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 $e = 0$.

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

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 $M + 1 - \lfloor C/a \rfloor$.

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

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

(10)

In our test, we set $a = 10$ and $M = 100$. In Fig. 4, 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 $\ell_1$ minimization algorithm with $\epsilon = 0.001$ and $\delta = 10^{-6}$. The algorithm was run using six different initializations: five random ones and $w^{(0)} = 1$. 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. 4, the results produced by both algorithms are very close to the optimal ones. The greedy algorithm achieves the optimal 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 $k = 1, 2, \cdots, 100$.

C. 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 [19], [22], [34], [14]. In Fig. 5, 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 and therefore in the absence of an external injection of cash, all the nodes are in default except node iii.

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

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 $[(C - 100)/20]$ periphery nodes and give $20$ to each. This decreases the number of defaults by $C/20 + 1$.

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 $[(C - 200)/20]$ periphery nodes and give $20$ to each. This decreases the number of defaults by $C/20 + 2$.

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

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:

$$N_d(C) = \begin{cases} 
32 - \lfloor C/20 \rfloor & \text{if } C < 100, \\
31 - \lfloor C/20 \rfloor & \text{if } 100 \leq C < 200, \\
30 - \lfloor C/20 \rfloor & \text{if } 200 \leq C < 600, \\
0 & \text{if } C \geq 600.
\end{cases} \quad (11)$$

In Fig. 6, 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 $\ell_1$ minimization algorithm with $\epsilon = 0.001$ and $\delta = 10^{-6}$. The algorithm was run using six different initializations: five random ones and $w^{(0)} = 1$. 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. 6, the results produced by the reweighted $\ell_1$ 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 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.

**D. Example: Three Random Networks**

We now compare the reweighted $\ell_1$ 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 $e = 0$. The first one is a random graph with 30 nodes. For any pair of nodes $i$ and $j$, $L_{ij}$ 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. 7. 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

![Fig. 5. Core-periphery network topology.](image)

![Fig. 6. Our algorithms for minimizing the number of defaults vs the optimal solution, for the network of Fig. 5.](image)

![Fig. 7. Random core-periphery network to compare the reweighted $\ell_1$ algorithm and the greedy algorithm.](image)

![Fig. 8. Random core-periphery network with long chains to compare the reweighted $\ell_1$ algorithm and the greedy algorithm.](image)
The third one is a random core-periphery network with chains of periphery nodes. As shown in Fig. 8, 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 short chain are also uniformly distributed in \([0,1]\). 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 \(\ell_1\) minimization algorithm and the greedy algorithm on each sample network. In the reweighted \(\ell_1\) minimization algorithm, we set \(\epsilon = 0.001, \delta = 10^{-6}\). We run the algorithm using six different initializations: five random ones and \(w^{(0)} = 1\). Among the six solutions, the one with the smallest number of defaults is selected.

The results are shown in Figs. 9, 10, and 11. 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 \(\ell_1\) 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. 9, we see the performance of the reweighted \(\ell_1\) algorithm is close to the greedy algorithm on the random networks. From Fig. 10 and Fig. 11, we see that on random core-periphery networks, the greedy algorithm performs better than the reweighted \(\ell_1\) algorithm, while on random core-periphery networks with chains, the reweighted \(\ell_1\) algorithm is better.

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

\[ D = w^T(p - \bar{p}) + v^T d, \]

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

Since \(D\) is strictly decreasing with respect to \(p\), Lemma 4 in [18] 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 2.** Assume that the liabilities matrix \(L\), the external asset vector \(e\), the weight vectors \(w > 0\) and \(v > 0\) 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 \(d\) as in Table 1. Then the optimal cash allocation policy to minimize the cost function \(D = w^T(p - \bar{p}) + v^T d\) can be obtained by solving the following mixed-integer linear program:

\[
\begin{aligned}
&\max_{p, c, d} w^T(p - \bar{p}) + v^T d \\
&\text{subject to} \\
&1^T c \leq C, \\
&c \geq 0, \\
&0 \leq p \leq \bar{p}, \\
&\Pi^T p + e + c, \\
&\bar{p}_i - p_i \leq \bar{p}_i d_i, \text{ for } i = 1, 2, \ldots, N, \\
&d_i \in \{0, 1\}, \text{ for } i = 1, 2, \ldots, N.
\end{aligned}
\]
Proof: Let \((p^*, c^*, d^*)\) be a solution of the mixed-integer linear program (12–18). We first show that \(p^*\) is a clearing payment vector, i.e., that for each \(i\), we have \(p^*_i = \bar{p}_i\) or \(p^*_i = \sum_{j=1}^{N} \Pi_{j} p^*_j + e_i + c_i\). 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 \(p^\xi\) which is equal to \(p^*\) in all components except the \(k\)-th component. We set the \(k\)-th component of \(p^\xi\) to be \(p^\xi_k = p^*_k + \xi\), where \(\xi > 0\) is small enough to ensure that \(p^\xi_k < \bar{p}_k\) and \(p^\xi_k < \sum_{j=1}^{N} \Pi_{jk} p^\xi_j + e_k + c_k\). Since \(\Pi\) is a matrix with non-negative entries, for any \(i \neq k\), we have:

\[
p^\xi_i = p^*_i < \sum_{j=1}^{N} \Pi_{ji} p^\xi_j + e_i + c_i < \sum_{j=1}^{N} \Pi_{ji} p^*_j + e_i + c_i.
\]

In addition, \(\bar{p}_k - p^\xi_k < \bar{p}_k - p^*_k \leq \bar{p}_k d_k\). Thus, \((p^\xi, c^\xi, d^\xi)\) is also in the feasible region of (12–18) and achieves a larger value of the objective function than \((p^*, c^*, d^*)\). This contradicts the fact that \((p^*, c^*, d^*)\) is a solution of (12–18).

Hence, \(p^*\) is a clearing payment vector.

Second, we show that \(d^* = \bar{d}_i - p^*_i > 0\). If \(\bar{p}_i - p^*_i > 0\), then \(d^*_i = 1\) due to constraints (17) and (18). If \(\bar{p}_i - p^*_i = 0\), then constraint (17) 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, \(d^*_i = \bar{d}_i - p^*_i > 0\).

So far, we have proved that \(p^*\) and \(d^*\) are the clearing payment vector and default indicator vector, respectively, for cash injection vector \(c^*\). We now prove by contradiction that \(c^*\) is the optimal cash injection allocation. Assume \(c' \neq c^*\) leads to a strictly smaller value of the cost function \(D\) than does \(c^*\). In other words, suppose that \(c'\) satisfies the constraints (13) and (14), and that the corresponding clearing payment vector \(p'\) and default indicator vector \(d'\) satisfy \(w^T (p' - p^*) + v^T d' < w^T (p - p^*) + v^T d^*\), which is equivalent to:

\[
w^T p' - v^T d' < w^T p^* - v^T d^*.
\]

Since \(p'\) is the corresponding clearing payment vector, constraint (15) and (16) are satisfied. Moreover, \(d'\) is the corresponding default indicator vector satisfying constraint (17) and (18) for \(c'\). So \((c', p', d')\) is in the feasible region of (12–18) and achieves a larger objective function than \((p^*, c^*, d^*)\), which contradicts the fact that \((p^*, c^*, d^*)\) is the solution of (12–18).

VI. 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 3.** With the all-or-nothing payment mechanism, Problem I can be reduced to a knapsack problem, which means that Problem I is NP-hard.

**Proof:** Consider the network depicted in Fig. 12. The network has \(N = 2M\) nodes where \(M\) is a positive integer. We let \(L_{i,M+i} = \bar{p}_i\) for \(i = 1, 2, \ldots, M\); for all other pairs \((i,j)\), we set \(L_{ij} = 0\). We set the external asset vector to zero: \(e = 0\). We set all the weights to 1: \(w = 1\). 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 \(i = 1, \ldots, M\).

Note that under the all-or-nothing payment mechanism, fully rescuing node \(i\) for any \(i = 1, \ldots, M\) in Fig. 12 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 \(x\) 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 \(C\):

\[
\max_x \sum_{i=1}^{M} x_i \bar{p}_i \leq C, \quad x_i \in \{0, 1\}, \text{ for } i = 1, 2, \ldots, M.
\]

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 (19) is a knapsack problem, a well-known NP-hard problem. Thus, Problem I under the all-or-nothing payment mechanism for the network of Fig. 12, which can be reduced to (19), 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 4.** Assume that the liabilities matrix \( L \), the external asset vector \( e \), the weight vector \( w > 0 \) 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:

\[
\begin{align*}
\max_{p,c,d} & \quad w^T p \\
\text{subject to} & \quad 1^T c \leq C, \\
& \quad c \geq 0, \\
& \quad p_i = \bar{p}_i(1 - d_i), \text{ for } i = 1, 2, \ldots, N, \\
& \quad \bar{p}_i - \sum_{j=1}^N \Pi_{ji}p_j - e_i - c_i \leq \bar{p}_i d_i, \text{ for } i = 1, 2, \ldots, N, \\
& \quad d_i \in \{0,1\}, \text{ for } i = 1, 2, \ldots, N.
\end{align*}
\]

**Proof:** Let \((p^*, e^*, d^*)\) be a solution of the mixed-integer linear program (20–25). We first show that \(p^*\) is the clearing payment vector corresponding to \(e^*\). For node \(i\), if

\[
\bar{p}_i > \sum_{j=1}^N \Pi_{ji}p_j^* + e_i + c_i,
\]

then from constraints (24) and (25) 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 (24) 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 \(p^*\) is the clearing payment vector corresponding to \(e^*\) under the all-or-nothing payment mechanism.

Second, we prove by contradiction that \(e^*\) is the optimal allocation. Assume that \(e^*\) leads to a smaller weighted sum of unpaid liabilities, or equivalently, a larger value of \(w^T p^*\), where \(p^*\) is the clearing payment vector corresponding to \(e^*\). Since \(p^*\) 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 \(d^*\) as \(d_i^* = 0\) for \(p_i^* = \bar{p}_i\) and \(d_i^* = 1\) otherwise. Then \((p^*, e^*, d^*)\) is located in the feasible region of MILP (20–25) but leads to a larger value of the objective function than \((p^*, e^*, d^*)\). This contradicts the fact that \((p^*, e^*, d^*)\) is a solution of (20–25).

**A. Numerical Simulations**

To solve MILP (20), we use CVX, a package for specifying and solving convex programs [30], [29]. A variety of prior literature, e.g. [44], 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 core-periphery network shown in Fig. 13. 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. Every node has zero external assets: \(e = 0\). All the obligation amounts \(L_{i,j}\) are independent uniform random variables. For each pair of core nodes \(i\) and \(j\) the obligation amount \(L_{i,j}\) 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 \(w_k = 1\). 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.) We can see that for the core-periphery network, MILP (20) can be solved by CVX efficiently and accurately. The CVX code is given in Figure 14. The parameters that appear in the code are defined in Table II.

**VII. Conclusions**

In this work, we have developed a linear program to obtain the optimal cash injection policy to minimize the weighted sum of unpaid liabilities in a one-period financial system. We have further proposed a reweighted \(\ell_1\) 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 $\ell_1$ 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 use optimization package CVX [30], [29] 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.

VIII. ACKNOWLEDGMENTS

We would like to thank Xiaojun Lin, David Marshall, Celso Brunetti, and seminar participants at the Federal Reserve (Washington, DC), Federal Reserve Bank of Chicago, Federal Reserve Bank of Cleveland, Northwestern University’s EECS Department, and Bloomberg L.P. for helpful comments and suggestions.

REFERENCES


