OPTIMAL MONITORING AND MITIGATION OF SYSTEMIC RISK IN FINANCIAL NETWORKS

March 31, 2015
8th Financial Risks International Forum

Zhang Li
Joint work with Ilya Pollak, Borja Peleato and Ben Craig

School of Electrical and Computer Engineering
Purdue University
At 3pm, Al must pay $3 to Bo who must pay $2 to Cy who must pay $1 to Di

None has any cash

Suppose we have $3 to inject into the system
Optimal cash injection (zero defaults):
Optimal cash injection (zero defaults):

Both insufficient and wasteful:
Problem Statement

Given a fixed amount of cash to be injected into the system, how should it be allocated among the nodes to minimize:

- [Problem 1]
  the weighted sum of unpaid liabilities?
- [Problem 2]
  the number of defaults?
Related Work

• L. Eisenberg and T.H. Noe. Systemic risk in financial systems
  – proved the existence and uniqueness of the clearing payment vector.
  – proved that the clearing payment vector can be obtained via an LP.

• G. Demange. Contagion in financial networks: a threat index
  – developed a cash injection targeting policy for an infinitesimally small amount of injected cash.
  – can be extended to an optimal cash injection policy for non-infinitesimal cash amount, but it is less efficient than our LP method.

• L.C.G Rogers and L.A.M Veraart. Failure and rescue in an interbank network.
  – considered bankruptcy costs.
  – solvent banks rescue failing banks.

  – multi-period stochastic clearing framework.
  – proposed several strategies.
  – Max-liquidity policy aims to solve our Problem I, but it does not describe an algorithm for solving this problem.
Assumptions

• Single-period model
  – All loans are due at the same time

• Simultaneous clearing
  – I can use payments from borrowers to pay lenders

• All debts have the same seniority

• Payment mechanisms:
  – Proportional: If I owe more than I have, I pay out everything I have, proportionally to amounts owed
  – All-or-nothing: Defaulting nodes pay nothing

• All assets and loan amounts are known
How to Handle Cycles?

$3001$ unpaid liabilities
How to Handle Cycles?

$3001$ unpaid liabilities

$0$ unpaid liabilities
Our Contributions

• A linear programming algorithm for Problem 1 with proportional payments.

• A mixed-integer linear programming algorithm for Problem 1 with all-or-nothing payments.

• Two heuristic algorithms for Problem 2, based on reweighted $\ell_1$ minimization and greedy algorithm.
Variation on the Theme of Problem 1

- [Problem 1a]: Determine both the optimal cash injection amount $C$ and its corresponding optimal allocation among the nodes, to minimize $\lambda C + W$ where $W$ is the weighted sum of unpaid liabilities, and $\lambda$ is a given weight parameter.
- This is also a linear program.
Notation

c_i = external cash injection to node i

L_{ij} = amount owed by i to j

L_{ji} = amount owed by j to i

e_i = assets at node i from external sources
Notation, continued

\[ p_i' = \sum_{j} L_{ij} = \text{amount node } i \text{ owes to all its creditors} \]

\[ p_i = \text{amount node } i \text{ actually repays to its creditors, } p_i \leq p_i' \]

\[ d_i = \text{default indicator (1 if } i \text{ defaults, 0 if } i \text{ does not default)} \]

\[ p_i' - p_i = \text{node } i\text{'s total unpaid liabilities} \]
Notation, continued

\[ \Pi_{ji} = \begin{cases} \frac{L_{ji}}{p_j'} & \text{if } p_j' \neq 0 \\ 0 & \text{otherwise} \end{cases} = \text{what } j \text{ owes to } i, \]

what j owes to i, as a fraction of j’s total liabilities

\[ \sum_j \Pi_{ji} p_j = \text{total amount received by node } i \text{ from all its borrowers} \]

\[ \sum_j \Pi_{ji} p_j + e_i + c_i = \text{funds available to node } i \text{ for making payments to its creditors} \]
Weighted Unpaid Liabilities

\[ W = w^T (p' - p), \]

where \( w \) is a given weight vector (e.g., \( w = 1 \)).

Note: \( p' \) are the amounts owed---these are constants that cannot be changed. Therefore,

\[ \min_p W \iff \max_p w^T p \]
Minimizing Unpaid Liabilities
(Proportional Payment Mechanism)
Minimizing Unpaid Liabilities

**Theorem 1.** Under the proportional payment mechanism, injecting cash amount $C$ to minimize the weighted sum of unpaid liabilities (Problem 1) is equivalent to the following linear program:

Find cash injection vector $c$ and clearing payment vector $p$ to maximize $w^T p$ subject to

- $1^T c \leq C$
- $c \geq 0$
- $0 \leq p \leq p'$
- $p \leq \Pi^T p + e + c$

[Proof in Appendix 2.]
Minimizing $\lambda C + W$

Theorem 2. Under the proportional payment mechanism, minimizing a linear combination $\lambda C + W$ of the cash injection amount $C$ and the weighted sum of unpaid liabilities $W$ given a weight $\lambda$ (Problem 1a) is equivalent to the following linear program:

Find cash injection vector $c$ and clearing payment vector $p$ to minimize $\lambda \mathbf{1}^T c - \mathbf{w}^T p$ subject to

- $c \geq 0$
- $0 \leq p \leq p'$
- $p \leq \Pi^T p + e + c$

[Proof in Appendix 2.]
Minimizing Unpaid Liabilities (All-or-Nothing Payment Mechanism)
Theorem 3. Under the all-or-nothing payment mechanism, injecting cash amount \( C \) to minimize the weighted sum of unpaid liabilities (Problem 1) is an NP-hard problem and is equivalent to the following mixed-integer linear program:

Find cash injection vector \( \mathbf{c} \), default indicator vector \( \mathbf{d} \) and clearing payment vector \( \mathbf{p} \) to maximize \( \mathbf{w}^T \mathbf{p} \) subject to

\[
\begin{align*}
1^T \mathbf{c} & \leq C \\
\mathbf{c} & \geq \mathbf{0} \\
p_i & = p'_i (1 - d_i), \text{ for } i = 1, 2, \ldots, N \\
p'_i - \sum_{j=1}^N \prod_{j+i} p_j - e_i - c_i & \leq p'_i d_i, \text{ for } i = 1, 2, \ldots, N \\
d_i & \in \{0, 1\}, \text{ for } i = 1, 2, \ldots, N
\end{align*}
\]

[Proof in Appendix 2.]
Good News

- Accurate and quick solutions of the mixed-integer program achievable for networks of interest.
- Use simulated 1000-node core-periphery networks with a core of 15 as a caricature of the US banking system.
- Use modern Matlab-based optimization package CVX.
Example: A Core-Periphery Network

- A fully connected core of 15 nodes
- 70 periphery nodes connected to each core node
- Every periphery node connected to only one core node
- Total of 1065 nodes
- A crude model of the US banking system
- Each $L_{core,core}$ uniform in $[0,10]$ 
- Each $L_{periphery,core}$ uniform in $[0,1]$
To solve the MILP, we use CVX, a package for specifying and solving convex programs [3,4].

Simulations for Core-Periphery Networks

Generate 100 samples, run CVX code:

- CPU: 2.66GHz Intel Core2 Duo Processor P8800
- Average running time: 1.7598s
- Sample standard deviation: 0.9751s
- The relative gap between the objective of the solution and the optimal objective: less than $10^{-4}$
Minimizing Number of Defaults
(Proportional Payment Mechanism)
Number of Defaults

- $N_d =$ the number of nodes whose amount owed is bigger than the amount actually paid: $p'_i > p_i$

- $N_d =$ the number of nonzero entries in $(p' - p)$
Basic Idea

Al looks more hopeful than Bo who looks more hopeful than Cy – unless Cy gets big repayment from his borrowers.
Al looks more hopeful than Bo who looks more hopeful than Cy – unless Cy gets big repayment from his borrowers.

Repeatedly solve Problem 1 emphasizing nodes with small default amounts and deemphasizing nodes with large default amounts.

Objective $w^T p$

- $w_i$ small for large default amount $p_i' - p_i$ from previous iteration
- $w_i$ large for small default amount $p_i' - p_i$ from previous iteration
Algorithm

1. Set iteration number $m$ to 0.
2. Select initial weights $w^{(0)}$.
3. Solve Problem 1 with objective function replaced by $p^T w^{(m)}$, to obtain a clearing payment vector $p^{*(m)}$.
4. Update weights: $w_i^{(m+1)} \leftarrow \frac{1}{\exp\left(p_i' - p_i^{*(m)}\right) - 1 + \varepsilon}$
5. If $\|w^{(m+1)} - w^{(m)}\|_1 < \delta$, stop; else, increment $m$ and go to Step 3.
Greedy Algorithm

Al looks more hopeful than Bo who looks more hopeful than Cy – unless Cy gets big repayment from his borrowers.

Inject cash into the node with the smallest unpaid liability among all the defaulting nodes and rescue it at each iteration.

If a rescued node ends up with a surplus, the node would use its surplus to repay its cash injection.
Algorithm

1. Initialization: $C_r \leftarrow C \quad c \leftarrow 0 \quad w \leftarrow 1$

2. Solve Problem 1 to obtain a clearing payment vector $p$.

3. Calculate the surplus: $r \leftarrow \prod^T p + e + c - p$

4. Update the remaining cash:
   \[ C_r \leftarrow C_r + \sum_{i=1}^{N} \min \{ r_i, c_i \} \quad c_i \leftarrow c_i - \min \{ r_i, c_i \} \]

5. If $C_r = 0$, or no defaults, stop.

6. Find node $k$ with the minimum unpaid liability.

7. $c_k \leftarrow \min \{ C_r, p'_k - p_k \} \quad C_r \leftarrow C_r - c_k$  go to Step 2.
Testing the Algorithms, Part 1

• Create a network for which the optimal solution can be computed analytically.

• Run the algorithms and compare the result to the theoretical optimum.

• Examples: a core-periphery network (optimal solution derived in Appendix 4), a tree-structured network (Appendix 5), a network with M cycles (Appendix 6).
Example 1: Core-Periphery Network

Zero assets: $e = 0$
Example 1: Our Algorithms

Reweighted $\ell_1$ minimization: $\varepsilon = 0.001$, $\delta = 0.001$;
six initial weight vectors: $\mathbf{1}$ and five random vectors.
Testing the Algorithms, Part 2

- Create random networks.
- Compare the reweighted $\ell_1$ minimization algorithm and the greedy algorithm.
Example 1: Random Graph

- 30 nodes
- Zero assets: $e = 0$
- For any pair of nodes $i$ and $j$:
  - with probability 0.8, $L_{ij}$ is zero;
  - with probability 0.2, $L_{ij}$ is uniformly distributed between [0,2].
- Generate 100 samples and run both our algorithms.
Comparison: Reweighted $\ell_1$ vs Greedy Random Graph

Graph showing the average number of defaults vs bailout amount $C$. The graph includes lines for reweighted $\ell_1$ minimization, ±2 standard errors, greedy algorithm, and ±2 standard errors. The x-axis represents the bailout amount $C$, and the y-axis represents the average number of defaults.
Example 2: Core-Periphery

5 core nodes: fully connected

20 periphery nodes for each core node:
1 link to the core node

Unif. [0,1]

Unif. [0,20]

5 core nodes: fully connected
Comparison: Reweighted $\ell_1$ vs Greedy Core-Periphery
Example 3: C-P with Long Chains

- 5 core nodes: fully connected
- 20 periphery chains for each core node: 1 link to the core node
- Unif. [0,1]
- Unif. [0,20]
- Unif. [0,1]
- 5 core nodes: fully connected
Comparison: Reweighted $\ell_1$ vs Greedy C-P with Long Chains

![Graph showing reweighted $\ell_1$ minimization, ±2 standard errors, greedy algorithm, and ±2 standard errors. The graph plots the average number of defaults against the bailout amount $C$. The reweighted $\ell_1$ minimization curve is consistently below the greedy algorithm curve, indicating better performance over a range of bailout amounts.](image-url)
Future Work: Weighted Combination of Problems 1 and 2

- Given a fixed cash injection amount, minimize a linear combination of the sum of weights over the defaulted nodes and the weighted sum of unpaid liabilities.
- We can show that this is a mixed-integer linear program.
- Will investigate which network structures and weights lead to efficient solution.
- Can the heuristic algorithms for Problem 2 be adapted to solve this problem?
Q & A

Thank you!

Working Paper available in SSRN:
Appendix 1
More Future Work
Future Work: Stochastic Capital

- Testing the numerical algorithms.
- Are they fast enough for relevant network sizes?
- Models of stochastic capital?
- Using the framework in a stress testing context, e.g., to set capital requirements.
Future Work: Bankruptcy Costs

- Suppose that a defaulting node only has a fraction of its external assets and a fraction of its internal assets available for repaying its creditors (Rogers-Veraart 2013).
- Also, only a fraction of cash injection at a defaulting node goes to the creditors.
- Clearing payment vector is obtained through a fictitious default algorithm (Rogers-Veraart 2013).
- Cash injection problem is more difficult.
- All-or-nothing payment mechanism is a particular case.
Future Work: More Practical Assumptions

- How to handle different seniorities?
  - Payments that are always proportional to the amounts owed are the key feature that made our problem a linear program.
- How to handle different tenors?
- How to handle dynamics?
Future Work: Analysis of the Algorithm for Problem 2

- Role of parameters ($\epsilon$, $\delta$, initialization) and robustness with respect to parameter choice?
- Convergence results for some set of network structures?
- Alternative methods?
  - Problem is non-convex.
  - Is combinatorial search the only way to guarantee the optimal solution?
Future Work: Identifying Important Nodes

• How much do reductions in node i’s payments influence the overall unpaid liability in the system?

• Let $D_i(x) =$ overall unpaid liability in the system if the payment of node $i$ reduced from $p_i$ to $p_i - x$.

• Let $g_i(x) = \frac{dD_i(x)}{dx}$.

• Let $h_i(x) > 0$ be monotonically decreasing in $x$ --- roughly speaking, it represents our belief of how likely node $i$’s payments are to be reduced by $x$.

• Danger Index of node $i$: $\int_{0}^{p_i} h_i(x) g_i(x) \, dx$
Appendix 2
Proofs of Theorems 1 and 2
Programming Characterization of Clearing Vectors (Eisenberg-Noe)

Lemma 4 from [1]. If $f: [0, p'] \rightarrow \mathbb{R}$ is a strictly increasing function, then any solution to the following problem is a clearing payment vector:

$$\max_{p \in [0, p']} f(p)$$

subject to $p \leq \Pi^T p + e$

Theorem 1: Proof

*Theorem 1.* Problem 1 $\iff$ Find $c \geq 0$, $p$ to max $w^T p$

\[ \text{s.t. } 1^T c = C; \quad 0 \leq p \leq p'; \quad p \leq \Pi^T p + e + c \]

- For any fixed $c$, Lemma 4 from [1] implies a unique solution $p$ to the LP, which is the clearing payment vector.
- Let $(p^*, c^*)$ be a solution.
- Let $c^#$ be a cash injection allocation, with corresponding clearing payment vector $p^#$, that leads to a smaller weighted sum of unpaid liabilities: $w^T(p' - p^#) < w^T(p' - p^*)$.
- Then $(p^#, c^#)$ satisfies all the constraints of the LP, yet achieves a larger value of the objective function than the solution $(p^*, c^*)$: $w^T p^# > w^T p^*$.
- This is a contradiction.
Theorem 2: Proof

*Theorem 2.* Problem 1a $\iff$ Find $c \geq 0, p$ to min $\lambda 1^T c - w^T p$

s.t. $0 \leq p \leq p'$ and $p \leq \Pi^T p + e + c$.

- If $(p^*, c^*)$ is a solution, then it must be a solution to Problem 1 for $C = 1^T c^*$. Thus, $p^*$ is the unique clearing payment vector for the cash injection allocation $c^*$.

- The fact that $(p^*, c^*)$ minimizes $\lambda 1^T c - w^T p$ means that it also minimizes $\lambda 1^T c + w^T (p' - p) = \lambda C + W$, since $p'$ is a constant.
Appendix 3

Proof of Theorem 3
Theorem 3: Proof, Part 1

**Theorem 3 – 1st part.** Under all-or-nothing payment mechanism, Problem 1 is NP-hard.

**Proof.** Consider the network on the right with $e = 0, w = 1$.

- Let $x_i$ = rescue indicator variable for $i$.
  - For each defaulting node $i$: $x_i = 0$, $c_i = 0$, $p_i = 0$.
  - For each rescued node $i$: $x_i = 1$, $c_i = p'_i$, $p_i = p'_i$.

- Problem 1 is reduced to the following *knapsack* problem, which is NP-hard.

\[
\max_{x \in \{0,1\}^M} \sum_{i=1}^{M} x_i p'_i
\]

subject to $\sum_{i=1}^{M} x_i p'_i \leq C$
Theorem 3: Proof, Part 2

Theorem 3 – 2nd part. Under all-or-nothing payment mechanism, Problem 1 is an MILP.

- Let \((p^*, c^*, d^*)\) be a solution.

- \(p^*\) is the clearing payment vector, because for node \(i\):
  - If \(p'_i > \sum_{j=1}^{N} \prod_{j' \neq j} p_{j'} - e_i - c_i\), \(d_i^* = 1\) and \(p_i^* = 0\);
  - If \(p'_i \leq \sum_{j=1}^{N} \prod_{j' \neq j} p_{j'} - e_i - c_i\), to \(\max w^T p\), \(d_i^* = 0\) and \(p_i^* = p'_i\).

- Let \(c^\#\) be a cash injection allocation, with corresponding clearing payment vector \(p^\#\), that leads to a smaller weighted sum of unpaid liabilities: \(w^T(p' - p^\#) < w^T(p' - p^*)\).

- Define \(d^\#\) as \(d_i = 1\) for \(p_{i'}^\# = p'_i\) and \(d_i = 0\) for \(p_{i'}^\# = 0\). Then \((p^\#, c^\#, d^\#)\) satisfies all the constraints of the MILP, yet achieves a larger value of the objective function than the solution \((p^*, c^*, d^*)\).

- This is a contradiction.
Appendix 4
Optimal Solution of Problem 2 for a Core-Periphery Network
Example 1: Core-Periphery Network

Zero assets: $e = 0$
Example 1: Optimal Solution

- If \( C < $100 \), then select any \([C/20]\) periphery nodes and give $20 to each of them. This reduces the number of defaults by \([C/20]\).
- If \( $100 \leq C < $200 \), then
  - select any five periphery nodes of core node ii and give $20 to each of them, which saves both node ii and these five periphery nodes
  - select any other \([ (C-100)/20 ]\) periphery nodes and give $20 to each
  - this reduces the number of defaults by \([C/20]+1\)
- If \( $200 \leq C < $600 \),
  - use $200 to rescue all 10 periphery nodes of core node i, saving i, ii, and these 10 periphery nodes
  - select any other \([ (C-200)/20 ]\) periphery nodes and give $20 to each
  - this reduces the number of defaults by \([C/20]+2\)
- If \( C \geq $600 \), then all the nodes are rescued by giving $20 to each periphery node.
Example 1: Summary of the Optimal Solution

\[ N_d = \begin{cases} 
32 - \left\lfloor \frac{C}{20} \right\rfloor & \text{if } C < $100 \\
31 - \left\lfloor \frac{C}{20} \right\rfloor & \text{if } $100 \leq C < $200 \\
30 - \left\lfloor \frac{C}{20} \right\rfloor & \text{if } $200 \leq C < $600 \\
0 & \text{if } C \geq $600 
\end{cases} \]
Optimal Solution, $100 \leq C < $200

10 periphery nodes

$100

$100

$20

$20

$20

$20

$20

$20

$20

5 periphery nodes of ii get cash injection of $20

other [(C-100)/20] periphery nodes get cash injection of $20
Optimal Solution, $200 \leq C < $600

save all periphery nodes of i

10 periphery nodes

$20 \rightarrow$ i

$20 \rightarrow$ ii

$20 \rightarrow$ iii

$100 \rightarrow$ i

$100 \rightarrow$ ii

$100 \rightarrow$ iii

10 periphery nodes

other [(C-200)/20] periphery nodes get cash injection of $20
Appendix 5
Problem 2 for a Tree-Structured Network
Example 2: A Tree Network

Level $s = 0$ (root)

Level $s = 1$

Level $s = S - 2$

Level $s = S - 1$ (leaves)

Zero assets: $e = 0$
Example 2: Our Algorithms

\[ S = 10 \text{ levels, } \varepsilon = 0.001, \delta = 0.001; \]
\[ \text{six initial weight vectors: } \mathbf{1} \text{ and five random vectors.} \]
Some observations

- $2^s$ nodes at each level $s = 0, \ldots, S-1$.
- Leaves do not owe anything.
- Each node at level $s$ owes $2^{S-s}$ to each of its two children, $s = 0, \ldots, S-2$.
- Every non-leaf level $s$ owes, in aggregate, $2^{S+1}$ to level $s+1$.
- Therefore, if cash injection is $C \geq 2^{S+1}$, administering it at the root will achieve zero defaults.
- If $C < 8$, then not a single node can be saved, and all $2^{S-1} - 1$ non-leaf nodes are in default.
Optimal Solution for $8 \leq C < 2^{S+1}$

- If $C = 2^{S-s+1}$ is a power of 2, the optimal solution is to allocate it to some node at level $s$.
  - Prevent the defaults of this node and its $2^{S-s-1} - 2$ non-leaf descendants
  - Total number of defaults $2^{S-1} - 2^{S-s-1}$
- If $C$ is not a power of 2, apply this argument recursively:
  
  $$N_d = 2^{S-1} - 1 - \sum_{u=4}^{U} b(u)(2^{u-3} - 1)$$

- where $U$ is the number of bits in the binary representation of $C$, and $b(u)$ is the $u$-th bit from the right.
Optimal Solution for $8 \leq C < 2^{S+1}$

\[ C = b(U) \cdot 2^{U-1} + b(U - 1) \cdot 2^{U-2} + \cdots + b(4) \cdot 2^3 \]
Appendix 6
Problem 2 for a Network with Cycles
Example 3: Network with M Cycles

Zero assets: $e = 0$
Example 3: Our Algorithms

M = 100 cycles, $a = $10, $\varepsilon = 0.001$, $\delta = 0.001$;
six initial weight vectors: 1 and five random vectors.
Example 3: Optimal Solution

- If $C < a$, then
  - the root and all $M$ nodes connected to the root are in default;
  - the remaining $5M$ nodes are not in default.
- If $C \geq aM$, then allocating the entire amount to the root achieves zero defaults.
- If $a \leq C < aM$,
  - giving $a$ to a node connected to the root will prevent it from default;
  - total number of defaults is $M + 1 - \lceil C/a \rceil$.

$$N_d = \begin{cases} 
M + 1 & \text{if } C < a \\
M + 1 - \lceil C/a \rceil & \text{if } a \leq C < aM \\
0 & \text{if } C \geq aM
\end{cases}$$
Optimal Solution, $C \geq \alpha M$

0 defaults

Root node

$\alpha$

$2\alpha$

$\alpha$

$2\alpha$

$\alpha$

$\alpha$

$2\alpha$

$\alpha$

$2\alpha$

$\alpha$

$\alpha$

...
Optimal Solution, $a \leq C < aM$

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

Root node

$\lfloor C/a \rfloor$ cycles get cash injection of $a$
Appendix 7
Computing the Clearing
Vector for the Proportional
Payment Mechanism
The clearing payment vector is a fixed point of the map \( \Phi \):

\[
\Phi(p) = \min \left\{ \Pi^T p + e, \ p' \right\}
\]

**Fixed-point algorithm:**

Step 1: Initialization: set \( p^0 \leftarrow p' \), \( k \leftarrow 0 \), \( \delta \leftarrow \) a small positive number.

Step 2: \( p^{k+1} \leftarrow \Phi(p^k) \).

Step 3: if \( \| p^{k+1} - p^k \|_\infty < \delta \), stop and output the payment vector \( p^{k+1} \); else, set \( k \leftarrow k+1 \) and go to Step 2.

Computational complexity per iteration: \( \Theta(N^2) \).

The number of iterations is highly dependent on the network topology and the amounts of liabilities.
Fictitious Default Algorithm

Step 1: Initialization: set $p^1 \leftarrow p'$, $k \leftarrow 1$, and $D^{(0)} \leftarrow \emptyset$.

Step 2: For all nodes $i$, compute $v_i^{(k)} \leftarrow \sum_{j=1}^{N} \prod_{ji} p_j^{(k)} + e_i - p'_i$.

Step 3: Set $D^{(k)} = \{i : v_i^{(k)} < 0\}$.

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

Step 5: Otherwise, set $p_i^{(k+1)} \leftarrow p'_i$ for all $i \notin D^{(k)}$.

For all $i \in D^{(k)}$, compute the payments $p_i^{(k+1)}$ by solving:

$$p_i^{(k+1)} = e_i + \sum_{j \in D^{(k)}} \prod_{ji} p_j^{(k+1)} + \sum_{j \notin D^{(k)}} \prod_{ji} \bar{p}_j$$

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

Computational complexity per iteration: $O(N^3)$.

The algorithm terminates in at most $N$ iterations.
From Theorem 1, the clearing payment vector can be obtained by solving the linear program with $C = 0$.

$$\max_{p \in [0, p']} 1^T p$$

subject to $p \leq \prod^T p + e$

The computational complexity of solving an LP is $O(N^3)$. 
Comparison of the running times

Three topologies:

• Fully connected network:
  – 1000 nodes.
  – $L_{ij}$ and $e_i$ are independently and uniformly distributed in $[0,1]$.

• Core-periphery network:
  – 15 core nodes; 70 periphery nodes for each core node.
  – For each pair of core nodes $i$ and $j$, $L_{ij}$ is uniform in $[0,10]$; for a core node $i$ and its periphery node $k$, $L_{ki}$ is uniform in $[0,1]$.
  – $e_i = 0$ for all $i$.

• Linear chain network:
  – 1000 nodes.
  – For $i=1,2,\ldots,999$, $L_{i(i+1)}$ is uniform in $[0,10]$; for all other $(i, j)$, $L_{ij} = 0$.
  – $e_i$ is uniform in $[0,1]$ for all $i$. 
### Comparison of the running times

<table>
<thead>
<tr>
<th></th>
<th>Fixed-point algorithm</th>
<th>Fictitious default algorithm</th>
<th>LP algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ave (s)</td>
<td>stdev</td>
<td>ave (s)</td>
</tr>
<tr>
<td>fully connected</td>
<td>0.2187</td>
<td>0.0626</td>
<td>3.5049</td>
</tr>
<tr>
<td>core-periphery</td>
<td>0.2711</td>
<td>0.0702</td>
<td>3.5882</td>
</tr>
<tr>
<td>linear chain</td>
<td>0.0173</td>
<td>0.0264</td>
<td>3.1201</td>
</tr>
</tbody>
</table>
Appendix 8
Computing the Clearing Vector for the All-or-Nothing Payment Mechanism
Fixed-Point Algorithm / Fictitious Default Algorithm

The clearing payment vector is a fixed point of the map $\Psi$:

$$
\Psi_i(p) = \begin{cases} 
p'_{i} & \text{if } \sum_{j=1}^{N} \prod_{ji} p_{j} + e_{i} \geq p'_{i} \\
0 & \text{otherwise}
\end{cases}
$$

Fixed-point algorithm:
Step 1: Initialization: set $p^0 \leftarrow p'$, $k \leftarrow 0$.
Step 2: $p^{k+1} \leftarrow \Psi(p^k)$.
Step 3: if $p^{k+1} = p^k$, stop and output the clearing payment vector $p^{k+1}$; else, set $k \leftarrow k+1$ and go to Step 2.

Computational complexity per iteration: $\Theta(N^2)$.
The number of iterations is at most $N$. 
Mixed-Integer Linear Programming Method

From Theorem 3, the clearing payment vector can be obtained by solving the MILP with $C = 0$.

$$\max_{p,d} w^T p$$

subject to:

$$p_i = p'_i (1 - d_i), \text{ for } i = 1, 2, \ldots, N$$

$$p'_i - \sum_{j=1}^{N} \prod_{j,i} p_j - e_i \leq p'_i d_i, \text{ for } i = 1, 2, \ldots, N$$

$$d_i \in \{0, 1\}, \text{ for } i = 1, 2, \ldots, N$$

We solve this MILP via CVX.
Comparison of the running times

Three topologies:

• Fully connected network:
  – 1000 nodes.
  – $L_{ij}$ and $e_i$ are independently and uniformly distributed in $[0,1]$.

• Core-periphery network:
  – 15 core nodes; 70 periphery nodes for each core node.
  – For each pair of core nodes $i$ and $j$, $L_{ij}$ is uniform in $[0,10]$; for a core node $i$ and its periphery node $k$, $L_{ki}$ is uniform in $[0,1]$.
  – $e_i = 0$ for all $i$.

• Linear chain network:
  – 1000 nodes.
  – For $i=1,2,...,999$, $L_{i(i+1)}$ is uniform in $[0,10]$; for all other $(i, j)$, $L_{ij} = 0$.
  – $e_i$ is uniform in $[0,1]$ for all $i$. 
### Comparison of the running times

<table>
<thead>
<tr>
<th>Topology</th>
<th>Fixed-point algorithm</th>
<th>MILP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ave (s)</td>
<td>stdev</td>
</tr>
<tr>
<td>fully connected</td>
<td>0.0176</td>
<td>0.0252</td>
</tr>
<tr>
<td>core-periphery</td>
<td>0.0229</td>
<td>0.0205</td>
</tr>
<tr>
<td>linear chain</td>
<td>0.0331</td>
<td>0.0191</td>
</tr>
</tbody>
</table>

For all the three topologies, the fixed-point algorithm is significantly more efficient than the MILP method.
Appendix 9
Some examples
Problem 1 vs Problem 2

No cash injection =>
101 nodes in default,
$200 unpaid liabilities

$100 to Al =>
99 nodes in default,
$99 unpaid liabilities
No cash injection =>
101 nodes in default,
$200 unpaid liabilities

$100 to Al =>
99 nodes in default,
$99 unpaid liabilities

Proportional payment case:
$1 each to Al, Cy, ..., Zach =>
1 node in default,
$99 unpaid liabilities
Problem 1 vs Problem 2

Proportional payment case:
$1 each to Al, Cy, ..., Zach =>
1 node in default,
$99 unpaid liabilities

All-or-nothing:
$1 each to Bo, Cy, ..., Zach =>
1 node in default,
$100 unpaid liabilities

No cash injection =>
101 nodes in default,
$200 unpaid liabilities

$100 to Al =>
99 nodes in default,
$99 unpaid liabilities