Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Financial networks with singleton liability priorities *

Stavros D. Ioannidis^{*}, Bart de Keijzer, Carmine Ventre

King's College London, United Kingdom of Great Britain and Northern Ireland

ARTICLE INFO

Article history: Received 6 January 2023 Received in revised form 5 May 2023 Accepted 19 May 2023 Available online 24 May 2023

Keywords: FIXP Financial networks Systemic risk NP-hardness

ABSTRACT

Financial networks model debt obligations between economic firms. Computational and game-theoretic analyses of these networks have been recent focus of the literature. The main computational challenge in this context is the clearing problem, a fixed point search problem that essentially determines insolvent firms and their exposure to systemic risk, technically known as recovery rates. When Credit Default Swaps, a derivative connected to the 2008 financial crisis, are factored into the obligations, the clearing problem becomes more complex. Specifically, whenever insolvent firms pay their debts proportionally to their recovery rates, computing a weakly approximate solution was shown by Schuldenzucker et al. (2017) to be PPAD-complete. Additionally, Ioannidis et al. (2022) showed that computing a strongly approximate solution in the same framework is FIXP-complete. This paper addresses the computational complexity of the clearing problem in financial

networks with derivatives, whenever payment priorities among creditors are applied. This practically relevant model has only been studied from a game-theoretic standpoint. We explicitly study the clearing problem whenever the firms pay according to a singleton liability priority list and prove that it is FIXP-complete. Finally, we provide a number of NP-hardness results for the computation of priority lists that optimise specific objectives of importance in the domain.

© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

The financial services industry has been very creative with the constant introduction of new products designed as investment and/or risk management tools. This makes the web of liabilities between institutions in the market challenging to track and oversee. In fact, it is this inherent complex structure of the evolving modern financial system that has led to several somewhat unforeseen and deeply damaging crises, such as the Great Financial Crisis (GFC) of 2008. There is, therefore, the need to mathematically model and study such networks of obligations among interconnected financial agents in order to understand the impact of new products, regulations or even single contracts on the financial health of the system.

The main computational challenge in this context is the *clearing problem* introduced in [6]: Given the banks' funds and the face values of all the liabilities in the network, compute for each bank its exposure to systemic risk, in the form of what is known as its *clearing recovery rate*. It turns out that the complexity of this problem is closely related to the class of financial products populating the network. In fact, in financial networks where firms only subscribe simple debt contracts, clearing recovery rates can be computed in polynomial time [6,26]. This setup allows modelling financial networks as

* This article belongs to Section A: Algorithms, automata, complexity and games, Edited by Paul Spirakis.

* Corresponding author.

https://doi.org/10.1016/j.tcs.2023.113965 0304-3975/© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http:// creativecommons.org/licenses/by/4.0/).





E-mail address: stavros.ioannidis@kcl.ac.uk (S.D. Ioannidis).

directed graphs, where financial institutions are represented as nodes and debt obligations are represented as weighted edges. However this representation is considered too simplistic to capture more sophisticated products based on existing contracts like mortgages, loans, interest rates etc. These complex contracts are called *derivatives*. The addition of one such derivative, namely, *Credit Default Swaps (CDSes)*, introduced to financial networks in [29], makes the model more intriguing from a computational perspective. A CDS involves three parties *i*, *j*, *k*, where *i* must pay an amount of money to *j* on the condition that *k* cannot pay off all of its obligations. Since adding CDSes to a financial network may generate numerically irrational clearing recovery rates [29,17], our interest turns to finding approximate clearing recovery rates. Previously, it was proved that weakly (or "almost") approximate recovery rates are PPAD-complete to compute [30] and strong (or "near") approximate solutions are FIXP-complete to compute [17].

In this paper, we look at this problem from the perspective of the financial regulator. We ask whether rules that determine how insolvent banks pay off their debts can fundamentally change the computational hardness landscape above. The most-studied payment scheme is the *proportional payment scheme* where each bank pays off its debts proportionally to its recovery rate. However, defining priority classes amongst creditors and paying proportionally in each such class (using the funds available at the respective priority level) is another widely adopted measure used in practice in the industry. For example, some regulatory regimes require employees to be prioritised over other creditors whereas some advanced derivatives (such as, the renowned Collateralized Debt Obligations leading to the GFC) define their payoff via *tranching* (effectively a priority list) of the underlying securities. Whilst priority payments have been studied under a game-theoretic framework, see, e.g., [19,20,3,22], nothing is known about the computational complexity of the clearing problem with this payment scheme in presence of financial derivatives.

Our contribution. We study the clearing problem in financial networks with derivatives under the *priority list* payment scheme. Specifically, we examine financial networks consisting of both debt and CDS contracts and address the complexity of computing a *clearing recovery rate vector* whenever each bank has to pay its debts in the following way. For each bank, we define a partition of its liabilities into *priority classes*. With the funds available at a certain priority class (i.e., the money left after having paid off all the liabilities with higher priority), the liabilities at the current priority are paid proportionally – in particular, this means that these are paid in full if the funds are sufficient. This notion generalises the *proportional payment scheme* studied in related literature (i.e., the special case in which the partition contains only one part) and the class of *singleton priorities*, where each part is a singleton. We call this problem CDS-PRIORITY-CLEARING. Note that without CDSes, CDS-PRIORITY-CLEARING is known to be in P, both for proportional [6] and priority [20] payment schemes.

We first observe that whenever the partition defined by the priority list contains at least a part of size 2 or more, then CDS-PRIORITY-CLEARING is FIXP-complete. The class FIXP [8] is a complexity class that captures total search problems expressible in the form of computing a fixed point of a function that satisfies certain properties. In our context, it is important to observe that FIXP requires fixed-points of functions that can be expressed using the algebraic basis $A = \{+, -, *, /, \max, \min, \frac{k}{2}\}$. It is then not hard to see that the FIXP-completeness follows from the recent reduction given in [17] where all gadgets adopted have maximum out-degree 2. Therefore, the case that is left open is when all the parts of the priority list are singletons, hence we focus our attention on this setup in our work. We then call CDS-PRIORITY-CLEARING the problem of finding recovery rates for singleton priorities, i.e., each bank has an ordering of its liabilities according to which its debts are paid off.

Our first contribution gives technical evidence that, in comparison to enforcing proportional payments, a financial regulator cannot reduce the complexity of computing clearing recovery rates by instead enforcing priority payments. Specifically, we prove that CDS-PRIORITY-CLEARING is at least as hard as the *square root sum* (SQRT-SUM) problem [8,11,21,31] and that it is complete for FIXP. Thus, this provides a complete picture of the complexity for the clearing problem, under all payment schemes proposed so far in the literature. Our proof of FIXP-completeness adopts the general approach that was previously used in [17]. Specifically, for each arithmetic operation contained in basis *A*, we construct a financial network with priority payments. This network comprises of one or two input banks and one output bank, depending on the arithmetic operation. The clearing recovery rate of the output bank is determined by applying the arithmetic operation to the clearing recovery rates of the input banks. We refer to these constructions as "financial network gadgets" and use them to simulate the corresponding arithmetic operations. Moreover, our reduction introduces new financial network gadgets with priority payments for the operations in the basis *A*. In particular, the multiplication gadget we devise highlights the flexibility of financial networks in handling arithmetic operations.

Whilst the regulator cannot ease the computational complexity of the problem, we wonder whether one can efficiently compute the banks' priority lists to optimise certain objective functions of financial interest. These include maximising the equity of a specific bank, maximising the liquidity in the system, and minimising the number of activated CDSes. As our second main contribution, we present a set of NP-hardness results showing an interesting parallel with the known hardness of computing similarly "optimal" solutions with proportional payments [25].

Related work. Financial networks consist of interconnected nodes, representing economic entities in an arc-weighted graph with arcs representing debt obligations. Systemic risk, stability and contagion in financial networks have been studied extensively in the literature [7,1,5,16,15,14,12]. Eisenberg and Noe [6] are among the first to present a computational perspective of the clearing problem in financial networks. Their work established a fixed point computational framework for studying financial networks that contain simple debt contracts and provided a set of positive results for the existence and computation of the "proper" clearing payments to avoid financial crisis, whenever banks pay proportionally. A variation of this model

with the addition of default costs is studied by Rogers and Veraart in [26], where the authors approach ways of rescuing systemic failure through bailouts or bank merging. Recent mechanisms and clearing problems have been studied in [28,23].

The modern structure of financial systems consisting of both simple debt contracts and derivatives, is introduced by Schuldenzucker et al. in [29] with the addition of the Credit Default Swap-CDS. In [30], assuming proportional payments, the authors express computation of a clearing recovery rate vector as a fixed point computation, where a clearing recovery rate vector refers to the proportion of liabilities each bank can pay off in financial networks with debt contracts and credit default swaps. Moreover since instances that contain irrational solutions exist [29,17], they prove that computing a weak (or "almost") approximate solution is PPAD-complete, implying that no PTAS exists unless P = PPAD. As weakly approximate solutions may be far removed from actual solutions, Ioannidis et al. [17] study the complexity of computing a strong (or "near") approximate solution and establish that the exact computation of a clearing recovery rate vector is FIXP-complete, and that its strong approximation version is complete for the derived and closely associated class FIXP_a. Sufficient and necessary conditions for the existence of irrational solutions are presented in both [29,17]. The papers [22,24], further analyse properties of financial networks with CDSes.

Approaching financial networks game-theoretically is another current trend in the area. Among many, Bertschinger et al. [3] introduce the coin and edge ranking games. These are games where banks have incentive to define their payments so as to increase various objectives like their total income or equity. Papp and Wattenhofer in [23] study financial networks with CDSes where banks may influence the system via donation of assets or contract cancellations. Moreover they draw parallels among networks with credit default swaps and classical game theoretic settings like the prisoners' dilemma. Kanellopoulos et al. [20,19] introduce more financial network games and explore new strategy spaces.

The FIXP complexity class is introduced by Etessami and Yannakakis in [8] as a proper class for containing total search problems whose solutions can be cast as a fixed point computation over suitable continuous functions. The authors established among others FIXP-completeness of the fundamental problem of computing a Nash equilibrium in games with three or more players. A brief description of the FIXP class is presented in [32,17]. There are various further recent papers that show FIXP-completeness or hardness of a range of problems, including fair division problems under suitable assumptions [10], equilibrium computation in first-price auctions [9] and the computational version of the Hairy Ball Theorem [13].

2. Model and preliminaries

Financial networks. A financial network consists of a set of financial entities (which we refer to as *banks* for convenience), interconnected through a set of financial contracts. Let $N = \{1, ..., n\}$ be the set of n banks. Each bank $i \in N$ has *external assets*, denoted by $e_i \in \mathbb{Q}_{\geq 0}$. We let $e = (e_1, ..., e_n)$ be the vector of all external assets. We consider two types of liabilities among banks: *debt contracts* and *credit default swaps* (*CDSes*). A debt contract requires one bank i (debtor) to pay another bank j (creditor) a certain amount $c_{i,j} \in \mathbb{Q}_{\geq 0}$. A CDS requires a debtor i to pay a creditor j on condition that a third bank called the *reference bank* R is in default, meaning that R cannot fully pay its liabilities. Formally, we associate each bank i a variable $r_i \in [0, 1]$, called the *recovery rate*, that indicates the proportion of liabilities it is able to pay. Having $r_i = 1$ means that bank i can fully pay its liabilities, while $r_i < 1$ indicates that i is in default. In case a reference bank R of a CDS is in default, the debtor i of that CDS pays the creditor j an amount of $(1 - r_R)c_{i,j}^R$, where $c_{i,j}^R \in \mathbb{Q}_{\geq 0}$ is the face value of the CDS. The value $c_{i,j}$ ($c_{i,j}^R$, resp.) of a debt contract (CDS, resp.) is also called the *notional* of the contract. Finally, we let c be the collection of all contracts' notionals. We do not allow any bank to have a debt contract with itself, and assume that all three banks in any CDS are distinct.

The financial system \mathcal{F} can therefore be represented as the triplet (N, e, c). Given \mathcal{F} , we let $\mathcal{DC}_{\mathcal{F}}$ denote the set of all pairs of banks participating in a debt contract in \mathcal{F} . Similarly, $\mathcal{CDS}_{\mathcal{F}}$ denotes the set of all triplets participating in a CDS in \mathcal{F} . (We drop \mathcal{F} from the notation when it is clear from the context.)

The contract graph of $\mathcal{F} = (N, e, c)$ is defined as a coloured directed multigraph-like structure $G_{\mathcal{F}} = (V, A)$, where V = Nand $A = (\bigcup_{k \in N} A_k) \cup A_0$ where $A_0 = \{(i, j) \mid i, j \in N \land c_{i,j} \neq 0\}$ and $A_k = \{(i, j) \mid i, j \in N \land c_{i,j}^k \neq 0\}$. Each arc $(i, j) \in A_0$ is coloured blue and each $(i, j) \in A_k$ orange. For all $(i, j, R) \in CDS$ we draw a dotted orange line from node R to arc $(i, j) \in A_R$, denoting that R is the reference bank of the CDS between i and j.¹ Finally, we label each blue and orange arc with the notional of the corresponding contract, and each node with the external assets of the corresponding bank.

Payment schemes. All banks are obliged to pay off their liabilities according to a set of pre-specified rules, which we refer to as a payment scheme. If a bank has sufficient assets, then the payment scheme is trivial and prescribes to simply make payments that correspond exactly to each of the bank's liabilities. If there are insufficient assets, on the other hand, the payment scheme will determine for each of the outgoing contracts how much of it is paid off. The most studied payment scheme is the *proportional payment scheme*, where each bank *i* submits an r_i proportion of each liability, leaving a $(1 - r_i)$ fraction of each liability unpaid. In this paper we study payments resulting from another rule, called the *singleton liability priority lists* payment scheme. More specifically, given a financial system $\mathcal{F} = (N, e, c)$, for each bank *i* we define a total order over the arcs going out of *i* in $G_{\mathcal{F}}$. We denote the singleton liability priority list of *i* as $P_i = (i_1 | i_2 | \dots | i_{outdeg(i)})$, where i_k stands for the *k*-th element in the order, or *k*-th *priority* of node *i*, and outdeg(*i*) denotes the out degree of node *i* in $G_{\mathcal{F}}$. The payments under this scheme are now formed through an iterative process where each bank pays off its liabilities, one

¹ This means that $G_{\mathcal{F}}$ can be represented as a directed hypergraph with arcs of size 2 and 3.



Fig. 1. The contract graph of the financial network presented in Example 1. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

after the other, according to the ordering given by its priority list. We denote by c_{i_k} the contract notional of the i_k th priority and denote by $\mathcal{P} = (P_1, \ldots, P_n)$ the profile of singleton liability priority lists. We denote a financial system \mathcal{F} endowed with a singleton liability priority profile \mathcal{P} as $(\mathcal{F}, \mathcal{P})$. The next example illustrates the model.

Example 1. The financial network of Fig. 1 consists of six banks, $N = \{1, 2, 3, 4, 5, 6\}$. Banks 2 and 5 have external assets $e_2 = e_5 = 1 - c$, for some constant $c \in (0, 1)$, while all other banks have zero external assets. The set of debt contracts is $\mathcal{DC} = \{(2, 3), (5, 4)\}$ and the set of CDS contracts is $\mathcal{CDS} = \{(2, 1, 5), (5, 6, 2)\}$. All contract notionals are set to 1. For example, $c_{2,3} = c_{2,1}^5 = 1$. Node 2 has two candidate singleton liability priority lists, one is $P_2^1 = ((2, 3) \mid (2, 1, 5))$, where $2_1 = (2, 3)$ with contract notional $c_{2_1} = c_{2,1} = 1$ and $2_2 = (2, 1, 5)$ with contract notional $c_{2_2} = c_{2,1}^5 = 1$. The other one is $P_2^2 = ((2, 1, 5) \mid (2, 3))$ where $2_1 = (2, 1, 5)$ with $c_{2_1} = c_{2,1}^5 = 1$ and $2_2 = (2, 3)$ with $c_{2_2} = c_{2,3} = 1$. Symmetrically one can derive the lists for node 5.

We are interested in computing for a pair $(\mathcal{F}, \mathcal{P})$, for each bank *i*, the proportion of liabilities that it is able to pay. This proportion is captured by the *recovery rate*, mentioned earlier: For each bank *i* we associate a variable $r_i \in [0, 1]$, that indicates the proportion of liabilities that bank *i* can pay. Recall that, to define the liability generated from a CDS contract, we need the recovery rate of the reference banks. Consequently in order to define all liabilities of banks in a financial system, we need to be presented with an a-priori recovery rate vector $r = (r_1, \dots, r_n)$. So given a $(\mathcal{F}, \mathcal{P})$ and assuming a vector $r \in [0, 1]^n$, we define the liabilities, the payments that each bank submits and the assets for each bank as follows.

We denote by $l_{i_k}(r)$ the *k*-th *liability priority* of node *i*. If $i_k = (i, j) \in DC$ for some $j \in N$, then $l_{i_k}(r) = c_{i,j}$ and if $i_k = (i, j, R) \in CDS$ for some $j, R \in N$, then $l_{i_k}(r) = (1 - r_R)c_{i,j}^R$. The liability of bank $i \in N$ to a bank $j \in N$ is denoted by $l_{i,j}(r)$ and it holds that

$$l_{i,j}(r) = c_{i,j} + \sum_{k \in \mathbb{N}} (1 - r_k) c_{i,j}^k.$$
(1)

We denote by $l_i(r)$ the total liabilities of node *i*, and it holds that

$$l_i(r) = \sum_{j \neq i} l_{i,j}(r) = \sum_{j \in N \setminus \{i\}} \left(c_{i,j} + \sum_{k \in N \setminus \{i,j\}} (1 - r_k) c_{i,j}^k \right).$$

Node *i* can fully pay its *k*-th priority only if he has sufficient assets left after paying off the liabilities corresponding to priorities i_1, \ldots, i_{k-1} . We denote by $p_{i_k}(r)$ the payment of node *i* to its *k*-th priority, and by $a_i(r)$ its assets, which are defined as the external assets it possesses plus all incoming payments received by its debtors (see below for a more formal definition). Under our singleton priority lists payment scheme:

$$p_{i_j}(r) = \max\left\{0, \min\left\{l_{i_j}(r), a_i(r) - \sum_{j' < j} l_{i_{j'}}(r)\right\}\right\}.$$
(2)

Moreover, we denote by $p_{i,j}(r)$ the payment of node *i* to node *j* under recovery rate vector *r*: Let $C_j = \{i_j \mid i_j \text{ is a contract with node$ *j* $as the creditor}, then <math>p_{i,j}(r) = \sum_{i_j \in C_j} p_{i_j}(r)$. The total payment made by a node is the sum of its individual payments to its priorities which is equal to the total sum of its payments to its creditors. Also, the payment of node *i* needs to be equal to the proportion of its total liabilities it can pay off. Therefore, the following equations hold.

$$p_i(r) = \sum_{j=1}^{\text{outdeg}(i)} p_{i_j}(r) = \sum_{j \in N} p_{i,j}(r) = r_i l_i(r).$$
(3)

The assets of a bank *i* are denoted as $a_i(r)$ and are the total amount of money it possesses summing its external assets and all incoming payments made by its debtors. It holds that

S.D. Ioannidis, B. de Keijzer and C. Ventre

$$a_i(r) = e_i + \sum_{j \neq i} p_{j,i}(r).$$
 (4)

We are interested in computing a specific recovery rate vector in $[0, 1]^n$, such that (3) holds (i.e., $p_i(r) = r_i l_i(r)$ for all $i \in N$), under the singleton liability priority list payment rule that we just defined through (2) and (4)). Formally:

Definition 1 (*Clearing recovery rate vector (CRRV*)). Given a financial system and a singleton liability priority profile (\mathcal{F} , \mathcal{P}), a recovery rate vector r is called clearing if and only if for all banks $i \in N$,

$$r_i = \min\left\{1, \frac{a_i(r)}{l_i(r)}\right\} \text{ if } l_i(r) > 0, \text{ and } r_i = 1 \text{ if } l_i(r) = 0.$$
(5)

We illustrate the above notions and the clearing recovery rate vectors by reconsidering Example 1 and computing them for some priority profile \mathcal{P} .

Example 1 (*continued*). Let c = 1/4 in Fig. 1. Let $\mathcal{P} = (P_2 = ((2, 3) | (2, 1, 5)), P_5 = ((5, 4) | (5, 6, 2)))$. Both nodes 2 and 5 receive no payment from any other node thus their assets are defined as $a_2 = e_2 = 1 - c$ and $a_5 = e_5 = 1 - c$. For node 2, given P_2 , we get that $l_{21} = l_{2,1} = c_{2,3} = 1$ and $l_{22} = l_{2,1} = (1 - r_5)c_{2,1}^5 = (1 - r_5)$, thus the total liabilities for node 2 are $l_2 = l_{2,1} + l_{22} = 2 - r_5$. For node 5 we get that $l_{51} = l_{5,4} = c_{51} = c_{5,4} = 1$ and $l_{52} = l_{5,6} = c_{52} = (1 - r_2)c_{5,6}^2 = 1 - r_2$, thus the total liabilities for node 5 are $l_5 = l_{51} + l_{52} = 2 - r_2$. Let us compute the CRRV. By (5) it must be $r_2 = \min\{1, a_2(r)/l_2(r)\} = \min\{1, 1 - c/2 - r_5\}$ and $r_5 = \min\{1, a_5(r)/l_5(r)\} = \min\{1, (1 - c)/(2 - r_2)\}$. After solving this system we get that $r_2 = r_5 = 1 - \sqrt{c}$ and since we assumed c = 1/4 we finally get that $r_2 = r_5 = 1/2$. For the payments of node 2, we know that $a_2 = 3/4$ and it first prioritises node 3 for which it has a liability of 1, thus it cannot fully pay off that liability and submits all of its assets to node 3, namely $p_{21} = p_{2,3} = 3/4$ and $p_{22} = p_{2,1} = 0$. The payments of node 5 are symmetrical.

Our search problem. We define CDS-PRIORITY-CLEARING to be the search problem that asks for a clearing recovery rate vector r given a pair (\mathcal{F}, \mathcal{P}). The term CDS refers to the fact that \mathcal{F} may contain CDS contracts (the problem becomes polynomial time computable without CDSes [20]) and the term PRIORITY indicates that banks pay according to a singleton liability priority profile \mathcal{P} . Similarly to [30,17], we assume that \mathcal{F} is non-degenerate. We will discuss this assumption in more detail below.

Definition 2. A financial system is *non-degenerate* if and only if the following two conditions hold. Every debtor in a CDS either has positive external assets or is the debtor in at least one debt contract with a positive notional. Every bank that acts as a reference bank in some CDS is the debtor of at least one debt contract with a positive notional.

The problem can be formally stated as follows.

CDS-PRIORITY-CLEARING
INPUT: A pair $(\mathcal{F}, \mathcal{P})$, where \mathcal{P} is a profile of singleton liability priority lists and $\mathcal{F} = (N, e, c)$ is a financial system where
$N = \{1, \dots, n\}$ is a set of <i>n</i> banks, $e = (e_i)_{i \in N} \in \mathbb{Q}^n$ is an <i>n</i> -dimensional vector of external assets, and $c = n \times n \times n$ is a
3-dimensional matrix that contains all contract notionals.
TASK: Compute a clearing recovery rate vector r for $(\mathcal{F}, \mathcal{P})$.

Given an instance $I \in \text{CDS-PRIORITY-CLEARING}$ we transform (5) into a function defined on arbitrary recovery rate vectors $r = (r_1, \dots, r_n)$ as²:

$$f_I(r)_i = \frac{a_i(r)}{\max\{a_i(r), l_i(r)\}}.$$
(6)

From (6), we ascertain that r is a clearing recovery rate vector for I if and only if r is a fixed point of f_I , namely $r = f_I(r)$. Thus, solving CDS-PRIORITY-CLEARING comes down to computing the fixed points of f_I . We define CDS-PRIORITY-CLEARING to contain only non-degenerate financial networks, for the analytical convenience that non-degeneracy provides (note that a division by 0 never occurs in $f_I(r)_i$ for these instances). It is not hard to see that f_I has fixed points, see, e.g., [22].

Notably, there exist instances of $(\mathcal{F}, \mathcal{P})$ that admit multiple CRRVs. The financial network of Fig. 2 consists of three banks 1, 2, 3, two debt contracts, (1, 2) and (2, 3), and one CDS contract, (1, 3, 2). All contract notionals are set to one. Let node 1 pay according to the priority list $P_1 = ((1, 3, 2) | (1, 2))$. For node 1 it holds that $e_1 = 1$ and $l_1(r) = l_{1,2}(r) + l_{1,3}(r) = c_{1,2} + (1 - r_2)c_{1,3}^2 = 2 - r_2$, thus $r_1 = 1/(2 - r_2)$. Moreover node 1 can fully pay any liability generated from the contract (1, 3, 2)

² Strictly speaking, $f_I(r)_i$ is well-defined only for nodes *i* that are not sinks (i.e. nodes with no outgoing arcs) with 0 external assets in the contract graph. Sink nodes have recovery rate 1, cf. (5). Hence, we implicitly exclude them from the definition of f_I . Their exclusion simply allows to bypass potential divisions by 0 in f_I (e.g., take node 1 in Fig. 1 when c = 1) while preserving its continuity.



Fig. 2. A pair of a financial network and a singleton liability priority list that admits multiple clearing recovery rate vectors. Whenever 1 pays according to $P_1 = ((1, 3, 2) | (1, 2)))$ the clearing recovery rate vectors are of the form $r = (1/(2 - r_2), r_2, 1), r_2 \in [0, 1]$.

because $e_1 = 1 \ge 1 - r_2 = l_{1,3}$. This means that node 2 receives an incoming payment of $1 - p_{(1,3,2)} = 1 - (1 - r_2) = r_2$. It is not hard to verify that the set of all clearing recovery rate vectors are of the form $r = (1/(2 - r_2), r_2, 1), r_2 \in [0, 1]$.

Irrationality. As is the case for the proportional payments, the singleton liability priority list model contains instances that admit clearing recovery rate vectors that contain numerically irrational components. Throughout this paper the term irrationality is used to characterise numerical values. Observation 1 below provides such an example while Observations 2 and 3 present examples of how the priority profile affects the payments in the network. These examples also provide insights on an important difference between the two payment schemes: In the proportional model, whenever the CRRV is irrational then the clearing payment vector must be irrational as well. That is not the case in the singleton liability priority payment scheme, where irrationality of a CRRV need not cause any irrationality in the payments.

Observation 1. There exist pairs of $(\mathcal{F}, \mathcal{P})$ that admit irrational CRRVs.

Proof. From Example 1, we know that $r_2 = r_5 = 1 - \sqrt{c}$. Thus, it is clear that for many choices of $c \in (0, 1)$ (e.g., c = 1/2) the CRRV is irrational.

Observation 2. There exist a pair $(\mathcal{F}, \mathcal{P})$ with an irrational CRRV and irrational payments.

Proof. Take again Example 1 and fix c = 1/3. We have $e_2 = e_5 = 2/3$, $l_{2,3} = l_{5,6} = 2/3$ and $r_2 = r_5 = 1 - \sqrt{1/3}$. Now consider the singleton liability priority lists $P_2 = ((2, 1, 5) | (2, 3))$ and $P_5 = ((5, 6, 2) | (5, 4))$. Since node 2 prioritises the (2, 1, 5) contract, it has to pay an amount of $1 - \sqrt{1/3}$ to node 1. Given that its total assets are 2/3, it can fully pay this liability and so $p_{2,1} = 1 - \sqrt{1/3}$ and what is left is being paid to node 3. Symmetrically, one can compute that $p_{5,6} = 1 - \sqrt{1/3}$. \Box

Observation 3. There exist a pair $(\mathcal{F}, \mathcal{P})$ with an irrational CRRV and rational payments.

Proof. Consider Example 1 once more and fix c = 1/2. This yields $e_2 = e_5 = 1/2$, $l_{2,3} = l_{5,6} = 1/2$ and $r_2 = r_5 = 1 - \sqrt{1/2}$. Consider the singleton liability priority lists $P_2 = ((2, 1, 5) | (2, 3))$ and $P_5 = ((5, 6, 2) | (5, 4))$. Since node 2 prioritises the (2, 1, 5) contract, it has to pay a amount of $1 - \sqrt{1/2}$ to node 1 but only possesses total assets of 1/2. Thus it cannot fully pay this liability, meaning that $p_{2,1} = 1/2$. Symmetrically, we can compute that $p_{5,6} = 1/2$. \Box

A primer on FIXP. A useful framework for studying the complexity of fixed point computation problems is defined in [8]. Both exact and approximate computation of the solutions to such problems are considered. We begin by defining the notion of approximation we are interested in. Let *F* be a continuous function that maps a compact convex set to itself and let $\epsilon > 0$ be a small constant. An ϵ -approximate fixed point of *F* is a point *x* within a distance ϵ near a fixed point of *F*, i.e., $\exists x' : F(x') = x' \land \|x' - x\|_{\infty} < \epsilon$. This notion is also known as *strong approximation*.³ We now introduce the problems we are focusing on. A fixed point problem Π is defined as a search problem such that for every instance $I \in \Pi$ there is an associated continuous function $F_I : D_I \to D_I$ —where $D_I \subseteq \mathbb{R}^n$ (for some $n \in \mathbb{N}$) is a convex polytope described by a set of linear inequalities with rational coefficients that can be computed from *I* in polynomial time— such that the solutions of *I* are the fixed points of F_I .

Definition 3 (*FIXP*). The class FIXP consists of all fixed point problems Π for which for all $I \in \Pi$ the function $F_I : D_I \to D_I$ can be represented by an algebraic circuit C_I over the basis $\{+, -, *, /, \max, \min, \sqrt[k]{}\}$, using rational constants, such that C_I computes F_I , and C_I can be constructed from I in time polynomial in |I|.

Definition 4 ($FIXP_a$). The class $FIXP_a$ is defined as the class of search problems that are the strong approximation version of some fixed point problem that belongs to FIXP.

³ A weak ϵ -approximate fixed point of a continuous function F is a point x such that its image is within distance ϵ of x, i.e., $||x - F(x)||_{\infty} < \epsilon$. Under polynomial continuity, a mild condition on the fixed point problem under consideration, an ϵ -approximate strong fixed point is also a δ -approximate weak fixed point, where δ is bounded by a polynomial in ϵ [8].



Fig. 3. The financial system constructed from a given Square Root Sum instance.

Definition 5 (*Linear-FIXP*). The class Linear-FIXP is defined analogously to FIXP, but under the smaller arithmetic basis where only the gates $\{+, -, \max, \min\}$ and multiplication by rational constants are used.

These classes admit complete problems. The completeness results in [8] in fact show that it is without loss of generality to consider a restricted basis {+, *, max} ({+, max}) for FIXP (Linear-FIXP), and to assume that $D_I = [0, 1]^{n.4}$ Hardness of a search problem Π for FIXP is defined through the existence of a polynomial time computable function $\rho : \Pi' \to \Pi$, for all $\Pi \in \text{FIXP}$, such that the solutions of I can be obtained from the solutions of $\rho(I)$ by applying a (polynomial-time computable) linear transformation on a subset of $\rho(I)$'s coordinates. This type of reduction is known as a *polynomial time SL-reduction*.

It is known that $FIXP_a \subseteq PSPACE$ and Linear-FIXP = PPAD [8]. An informal understanding of the hardness of FIXP versus the hardness of PPAD is as follows. PPAD captures a type of computational hardness stemming from an essentially combinatorial source. FIXP introduces on top of that a type of numerical hardness that emerges from the introduction of multiplication and division operations: These operations give rise to irrationality in the exact solutions to these problems, and may independently also require the computation of rational numbers of very high precision or very high magnitude.

3. Hardness of CDS-PRIORITY-CLEARING

We are interested in identifying the complexity of CDS-PRIORITY-CLEARING. Recall that in Example 1, we presented an instance which under suitable coefficients admits only irrational clearing recovery rate vectors, which means that either one should study this problem with respect to complexity classes compatible with real-valued solutions, or one should redefine the goal of the problem along the lines of finding a rational-valued approximation to a potentially irrational solution.

As an initial step, we first show that determining whether $r_i < 1$ for a specific bank *i* is at least as hard as solving the square root sum (SQRT-SUM) problem. An instance of SQRT-SUM consists of n + 1 integers $d_1, d_2, ..., d_n, k$ and asks whether $\sum_{i=1}^n \sqrt{d_i} \le k$. It is known that SQRT-SUM is solvable in PSPACE but it is unknown whether it is in P, or even in NP. In [31] it is shown that it can be solved in polynomial time in the unit-cost RAM model [31,27,2].

Lemma 1. For a given pair $(\mathcal{F}, \mathcal{P})$, deciding whether a specific bank is in default is SQRT-SUM-hard.

Proof. We prove the lemma by reducing CDS-PRIORITY-CLEARING to SQRT-SUM. Let $(d_1, ..d_n, k)$ be an instance of SQRT-SUM. Firstly, we note that in [4] it is shown that checking whether $\sum_{i=1}^{n} \sqrt{d_i} = k$ can be done in polynomial time. We check whether equality holds for our input first and proceed without loss of generality to the proof without minding equality.

We construct a financial system \mathcal{F} as follows, first we construct *n* financial subnetworks which we refer to as *square* root gadgets and denote by $g_{i,\sqrt{2}}$ the *i*-th square root gadget. Whenever referring to a node κ , belonging in a square root gadget $g_{i,\sqrt{2}}$, we use the notation κ_i . The square root gadget $g_{i,\sqrt{2}}$ consists of the financial network we presented in Fig. 1, augmented with two additional nodes x_i, y_i , and the CDS contract $(x_i, y_i, 2_i)$. Let $d_{\max} = \max_{i \in [n]} d_i$ and fix the external assets of nodes 2_i and 5_i to be $e_{2_i} = e_{5_i} = 1 - d_i/d_{\max}^2$. Moreover we let $e_{x_i} = 1$ and $e_{y_i} = 0$ and the CDS contract $(x_i, y_i, 2_i)$ has a notional of 1, i.e., $c_{x_i,y_i}^2 = 1$. The *n* square root gadgets are all connected to a single node τ with $e_{\tau} = 0$, by *n* debt contracts $\{(y_i, \tau) : i \in [n]\}$. There is one further node τ' to which τ is connected through debt contract (τ, τ') with notional $c_{\tau,\tau'} = k/d_{\max}$. The construction is illustrated in Fig. 3.

We claim that this resulting financial system has a clearing recovery rate vector r with $r_{\tau} = 1$ if and only if $\sum_{i=1}^{n} \sqrt{d_i} \ge k$. From the analysis of Example 1, it follows that under any clearing recovery rate vector r, the recovery rate of node 2_i is $r_{2_i} = 1 - \sqrt{d_i}/d_{\text{max}}$ for all $i \in [n]$. Since node 2_i is always in default (assuming all $d_i \ne 0$) the CDS ($x_i, y_i, 2_i$) is activated and since node x_i can fully pay its liabilities, node y_i receives a payment of $1 - r_{2_i} = \sqrt{d_i}/d_{\text{max}}$. This implies that τ receives a

⁴ We will make use of these facts in the proof of Theorem 1, below.

total payment of $(1/d_{\max}) \cdot \sum_{i \in [n]} \sqrt{d_i}$. Since τ has only one liability of k/d_{\max} , it holds that $r_{\tau} = 1$ if and only if the total payment that τ receives exceeds k/d_{\max} , i.e., if and only if $\sum_{i \in [n]} \sqrt{d_i} \ge k$, and this proves the claim. \Box

Next we show that CDS-PRIORITY-CLEARING and its strong approximation variant are FIXP and FIXP_a complete, respectively. Our hardness reduction does not start from a particular FIXP-hard problem. Rather, we show that we can take an arbitrary algebraic circuit and encode it in a direct way in the form of a financial system accompanied by a specific singleton liability profile. Hence, our polynomial time hardness reduction works from any arbitrary fixed point problem in FIXP. The reduction is constructed by devising various financial network gadgets which enforce that certain banks in the system have recovery rates that are the result of applying one of the operators in FIXP's arithmetic base to the recovery rates of two other banks in the system. In other words, we can design our financial systems and singleton liability priorities such that the interrelation between the recovery rates mimics a computation through an arbitrary algebraic circuit.

Theorem 1. CDS-PRIORITY-CLEARING is FIXP-complete, and its strong approximation version is FIXP_a-complete.

Proof. Containment of CDS-PRIORITY-CLEARING in FIXP is almost immediate. The clearing vectors for an instance $I = (N, e, c) \in \text{CDS-PRIORITY-CLEARING}$ are the fixed points of the function f_I defined coordinate-wise by

$$f_I(r)_i = \frac{a_i(r)}{\max\{l_i(r), a_i(r)\}}$$

as in (6). The functions $l_i(r)$ and $a_i(r)$ are defined in (1) and (4), from which it is clear that f_I can be computed using a polynomial size algebraic circuit with only {max, +, *}, and rational constants. Note that non-degeneracy of I prevents division by 0, so that the output of the circuit is well-defined for every $x \in [0, 1]^n$. This shows that CDS-PRIORITY-CLEARING is in FIXP and that its strong approximation version is in FIXP_a.

For the FIXP-hardness of the problem, let Π be an arbitrary problem in FIXP. We describe a polynomial-time reduction from Π to CDS-PRIORITY-CLEARING. Let $I \in \Pi$ be an instance, let $F_I : [0, 1]^n \rightarrow [0, 1]^n$ be *I*'s associated fixed point function, and let C_I be the algebraic circuit corresponding to F_I . Our reduction is analogous to the proof of Theorem 3 in [18] (an extended preprint of [17]) where the variant of the problem with proportional payments is shown to be FIXP-hard. As we want this proof to be self-contained, there are therefore some passages of text in this proof that correspond identically to passages of text in the proof of Theorem 3 in [18]. The proof consists of a "pre-processing" step (in which the algebraic circuit is transformed into a specific form) followed by a transformation into a financial network, where a set of financial subnetwork gadgets are interconnected, and each such gadget corresponds to an arithmetic gate in the algebraic circuit. Henceforth, we will use the term *signal* to refer to a value propagated between two gates in an arithmetic circuit. The pre-processing step of this reduction is entirely equal to that of [18]: This step transforms C_I into a circuit C'_I that satisfies that all the signals propagated by all gates in C'_I and all the used rational constants in C'_I are contained in the interval [0, 1]. The transformed circuit C'_I may contain two additional types of gates: Division gates and gates that compute the absolute value of the difference of two operands. We will refer to the latter type of gate as an *absolute difference gate*. The circuit C'_I does not contain any subtraction gates, and will not contain max and min gates either. The full details of the pre-processing step, as they appear in [18], have been included for the sake of completeness in Appendix A.

The second part of the proof (the transformation step) differs from [18] in that a different set of gadgets needs to be used. For notational convenience, we may treat C'_I as the function F_I , hence we may write $C'_I(x) = y$ to denote $F_I(x) = y$.

In the remainder of the proof, we define our reduction ρ : We construct our instance $\rho(I)$ from the circuit C'_I . The instance $\rho(I)$ will have the property that its clearing vectors are in one-to-one correspondence with the fixed points of C'_I , and that banks 1, ..., *n* in our construction correspond to the input gates of C'_I . More precisely, our construction is such that for each fixed point *x* of C'_I , in the corresponding clearing vector *r* for $\rho(I)$ it holds that $(r_1, ..., r_n) = x$.

Our reduction works through a set financial system *gadgets*, of which we prove that their recovery rates (under the clearing condition) must replicate the behaviour of each type of arithmetic operation that can occur in the circuit C'_{I} .

Each of our gadgets has one or two *input banks* that correspond to the input signals of one of the types of arithmetic gate, and there is an *output bank* that corresponds to the output signal of the gate. For each of the gadgets, it holds that the output bank must have a recovery rate that equals the result of applying the respective arithmetic operation on the recovery rates of the input banks. Each gadget represents an arithmetic gate of the circuit, and the output banks of a gadget are connected to input banks of other gadgets such that there is a direct correspondence with the infrastructure of the arithmetic circuit.

In our financial system, these gadgets are then connected together according to the structure of the circuit C'_1 : Output banks of (copies of) gadgets are connected to input banks of other gadgets through a single unit-cost debt contract, which mimics the propagation of a signal between two gates of the arithmetic circuit. This results in a financial system whose behaviour replicates the behaviour of the arithmetic circuit. The first layer of the financial system consists of *n* banks representing the *n* input nodes of the circuit, and the last layer of the financial system has *n* banks corresponding to the *n* output nodes of the circuit. As a final step in our reduction, the *n* output banks in the last layer are connected through a single unit-cost debt contract to the *n* input banks. This last step enforces the recovery rates of the input banks (i.e. banks 1, ..., *n*) are equal to the recovery rates of the last layer, under the clearing requirement. Consequently, any vector of



(b) Positive subtraction gadget $g_{\text{pos}-}$. Generates the value $\max\{0, r_1 - r_2\}$ from the input recovery rates r_1 and r_2 . In this gadget node 6 pays according to the priority list $P_6 = ((6, 7, 4) | (6, 8))$ thus it submits to node 8 the amount of $r_1 - r_2$ in case $r_1 > r_2$ and 0 otherwise.

Fig. 4. Example gadgets from our reduction.

clearing recovery rates for $\rho(I)$ must then correspond to a fixed point of C'_I , where the recovery rates of the first *n* banks in the system equal those of the final *n* banks, so that $C'_I(r_1, \ldots, r_n) = (r_1, \ldots, r_n)$, i.e., (r_1, \ldots, r_n) is a fixed point of C'_I .

For defining our gadgets, we use our graphical notation for convenience. As stated, our gadgets each have one or two input banks, and one output bank, where the output bank is forced to have a clearing recovery rate that is obtained by applying an arithmetic operation on the recovery rate of the input banks, so as to simulate the arithmetic basis on which the algebraic circuit C'_{1} is built.

As a simple example of one of these gadgets we define a straightforward addition gadget, named g_+ , depicted in Fig. 4a and borrowed from [18,17]. This gadget directly accounts for the addition operation in the arithmetic basis. In our figures, input banks are denoted by black arrows incoming from the left, and output banks correspond to black arrows pointing out of the bank. The black arrows represent connections to other gadgets, and these connections are always realised by a debt contract of unit cost, and are always from the output node of a gadget to an input node of another gadget. Another slightly more complex gadget example is the *positive subtraction* gadget, displayed in Fig. 4b, taking two inputs r_1 , r_2 and producing as output the value max{0, $r_1 - r_2$ }. This gadget is in turn used to form the absolute difference gadget, which can be constructed by combining two positive subtraction gadgets with an addition gadget (as $|r_1 - r_2| = \max\{0, r_1 - r_2\} + \max\{0, r_2 - r_1\}$). It is also used in the construction of our multiplication gadget.

In the figures representing our gadgets, some of the nodes have been annotated with a formula in terms of the recovery rates of the input banks of the gadget, subject to the resulting values being in the interval [0, 1]. This can be seen for example in the output node of our addition gadget in Fig. 4a. Such a formula represents the value that a clearing recovery rate must satisfy for the respective node. It is straightforward to verify that the given formulas are correct for each of our gadgets.

Since all signals inside C'_{1} are guaranteed to be in [0, 1] for all input vectors, our financial system gadgets can readily be used and connected to each other to construct a financial system that corresponds to C'_{1} , i.e., such that the clearing recovery rates of the input and output banks of each of the gadgets must correspond to each of the signals inside the circuit C'_{1} .

The remaining gadgets are displayed in Fig. 11 to 17, in Appendix B. Our gadgets are all new and defined for our setting with singleton priority payments, with the exception of Figs. 11 and 12 where the gadgets in [17,18] work also in our setting and are depicted for completeness.

Besides gadgets for the necessary arithmetic operations, our reduction employs a *duplication gadget* g_{dup} that can be used to connect the output of a particular gadget to the input of more than one subsequent gadget. This gadget is displayed in Fig. 12. Gadget g_{dup} furthermore has the convenient property that it can be used for multiplication by a rational $c \in [0, 1]$: One of the output nodes has the recovery of the input node, while the other has this recovery rate multiplied by c. When choosing c = 1 the input recovery rate is effectively duplicated.

Besides this set of gadgets that correspond to the arithmetic base of the circuit C'_i , in order to translate the circuit C'_i appropriately into a financial system, we also need the ability to generate rational constants as inputs to the gadgets (which correspond to the rational constant nodes of C'_i). This is straightforward: Any rational recovery rate $c \in [0, 1]$ can be generated using a single node *i* with a single outgoing debt contract of unit cost, that can be pointed towards an input node of any gadget copy.

Some of our gadgets are composed of auxiliary gadgets. A notably non-trivial gadget is the multiplication gadget (Fig. 16) where the main challenge in its construction is to ensure that non-degeneracy holds and that the expressions for the intermediate recovery rates inside the gadget are all contained in [0, 1]. Finally, we use the algebraic argument in [17] to substitute the division gadget with a square root gadget. We provide the details of this argument and gadget in Appendix B.1.

With the set of gadgets we have now defined, we can proceed to interconnect copies of gadgets to create financial systems of which the clearing recovery rates of the input and output gadgets respect an arbitrary algebraic computations over the arithmetic basis $\{+, -, *, ..., n\}$, the absolute difference operator, and rational constants. In particular, we can connect n input nodes, say banks $\{1, ..., n\}$, to a network of gadgets that implements the algebraic circuit C'_I . We connect its n outputs, say banks (m - n + 1, ..., m) (where m is the number of banks in the resulting financial system) pairwise to the n input nodes, and we define the resulting financial system to be $\rho(I)$. It is clear that $\rho(I)$ can be constructed in polynomial time from C'_I , and since C'_I can be constructed in polynomial time from I, the financial system $\rho(I)$ takes polynomial time to compute.

Next, we show that clearing vectors of $\rho(I)$ correspond in a polynomial-time computable way (in this case: trivially) to fixed points of C'_I , thereby proving the correctness of the reduction and completing the proof. Let r be a clearing vector of $\rho(I)$. We show that (r_1, \ldots, r_n) is a fixed point of I. Due to the fact that r is clearing, and by the definition of the gadgets, we know that the assets flowing into nodes $(1, \ldots, n)$ are respectively (r_1, \ldots, r_n) , and come from nodes $(m - n + 1, \ldots, m)$. Hence, the recovery rates of nodes $(m - n + 1, \ldots, m)$ are equal to (r_1, \ldots, r_n) . By construction of our network, (i.e., by correspondence of our gadgets and the way they are connected to each other, following the structure of the algebraic circuit C'_I , it must then hold that $C'_I(r_1, \ldots, r_n) = (r_1, \ldots, r_n)$. Hence, (r_1, \ldots, r_n) is a fixed point of I.

This completes the proof, as the above is sufficient to establish FIXP-completeness. FIXP_a completeness of the strong approximation variant is immediate, since any strong ϵ -approximation of the recovery rate vector of R(I) corresponds in the same manner to a strong ϵ -approximate fixed point of C'_I .

In addition, it is straightforward to see that fixed points of C'_{l} correspond to recovery rates of $\rho(I)$, so that indeed $\rho(I)$ captures the complete set of fixed points of C'_{l} : Let x be a fixed point of C'_{l} . Now construct a vector of recovery rates for $\rho(I)$ by setting $(r_{1}, \ldots, r_{n}) = x$ and compute the remaining recovery rates of the nodes inside the gadgets in the natural way, as described by the arithmetic operations that each of the gadgets represents. Since x is a fixed point of C'_{l} , the recovery rates of banks $(m - n + 1, \ldots, m)$ will be set to x, which causes the recovery rates of banks $(1, \ldots, n)$ (and thereby also the entire financial system) to satisfy the clearing condition. \Box

4. Hardness of deciding the best priority profile

In a financial network, each node i can be assigned one of outdeg(i)! singleton liability priority lists. Consequently the number of candidate priority profiles for a system is exponentially large in terms of its input size. This suggests that selecting from priority profiles to attain a particular objective could pose a computational challenge. Next, we consider the setting where a financial authority is able to determine which priority list each bank should get assigned, and is interested in assigning these in such a way that a specific objective is optimised for. We show that this problem is NP-hard for a set of natural choices of objective functions:

- 1. Maximising the equity of a specific node (cf. Theorem 2);
- 2. Minimising the number of defaulting nodes (cf. Theorem 3);
- 3. Minimising the number of not fully paid liabilities (cf. Theorem 3);
- 4. Minimising the number of activated CDSes in the financial system (cf. Theorem 4);
- 5. Maximising the liquidity in the financial system (cf. Theorem 5).

We proceed to prove these statements in the following.

Theorem 2. Finding a priority list profile that maximises the equity of a specific node is NP-hard.

Proof. We prove the theorem via a reduction from KNAPSACK. Let us be given a knapsack of capacity *B* and a set $S = \{a_1, \ldots, a_n\}$ of objects, having profit profit(a_i) and size size(a_i). Without loss of generality, we assume that size(a_i), for all $a_i \in S$ as well as *B* are integer numbers. The aim is to identify a collection of objects that can fit inside the knapsack while maximising the overall profit.

We construct a financial network $\mathcal{F} = (N, e, c)$ as follows. We introduce a node 0 with external assets $e_i = B$ and for each object $a_j \in S$ we introduce a corresponding node j and let node 0 hold a debt contract towards each node j with notional $c_{0,j} = \text{size}(a_j)$. Moreover we introduce a node τ with $e_{\tau} = 0$, and a node T with $e_T = 0$, which we refer to as the *terminal node*. We add a debt contract of notional $\text{size}(a_j)$ from each node j to node τ . For each node j we moreover introduce a *j*-subnetwork, consisting of:

- two nodes y_j and x_j with $e_{y_j} = 1$, $e_{x_j} = 0$,
- a CDS contract (y_j, x_j, j) with notional $c_{y_j, x_j}^j = \max_{a_j \in S} \{size(a_j)\},\$
- a node z_j towards which x_j holds a debt contract of notional $c_{x_j,z_j} = 1$,
- a node k_j with $e_{k_j} = \text{profit}(a_j)$ and an outgoing CDS (k_j, T, x_j) with notional $\text{profit}(a_j)$.

The construction of the *j*-subnetwork is illustrated in Fig. 5.



Fig. 5. The *j*-subnetwork used in the proof of Theorem 2.

Assume an optimal solution to the original KNAPSACK instance and let OPT be the set of the objects a_i contained in it. We know that $\sum_{a_i \in OPT} \text{size}(a_i) \leq B$ and that $\sum_{a_i \in OPT} \text{profit}(a_i)$, is the maximum profit that can fit in the knapsack. We define the set $N = \{1, ..., n\}$, and $N_{OPT} = \{j \mid a_j \in OPT\}$ to be the set containing all nodes in \mathcal{F} that correspond to objects contained in the optimal solution. Fix the singleton liability priority list for node i to be $\mathcal{P}_i = (\{N_{\text{OPT}}\} \mid \{N \setminus N_{\text{OPT}}\})$, meaning that i first prioritises all creditors in N_{OPT} in an arbitrary order and afterwards all other creditors in $N \setminus N_{\text{OPT}}$ again in an arbitrary order. Next we prove that under this profile, node T receives its maximum total assets. Observe that $\forall j \in N_{\text{OPT}}$, $p_{0j} = \text{size}(a_j)$ since $\sum_{j \in N_{\text{OPT}}} c_{0j} = \sum_{a_i \in \text{OPT}} \text{size}(a_j) \le B = e_0$. Since every *j* node that corresponds to an object $a_j \in \text{OPT}$ receives $\text{size}(a_j)$, it can fully pay node τ , so $r_j = 1$. For all creditors $m \in N \setminus N_{OPT}$ it holds that $p_{0m} < \text{size}(a_m)$. So, $\forall j \in N_{OPT} : r_j = 1$ while $\forall m \in N \setminus N_{OPT} : r_m < 1$. Next we prove that for each $j \in N_{OPT}$, $p_{k_i,T} = \text{profit}(a_j)$ and for each $m \in N \setminus N_{OPT}$, $p_{k_m,T} = 0$. Take a $j \in N_{\text{OPT}}$, we know that $r_j = 1$, which implies that the CDS (y_j, x_j, j) is not activated thus $r_{x_j} = 0$ which in turn activates the CDS (k_i, T, x_i) where node k_i pays profit (a_i) to node T. On the other side, for a $m \in N \setminus N_{OPT}$, it holds that $r_m < 1$, which means that the CDS (y_m, x_m, m) is activated and generates a liability of $\max_{a_i \in S} \{\text{size}(a_i)\} \cdot (1 - r_m)$ for node y_m . We prove that this liability is at least 1. For an object $a_m \notin OPT$, r_m indicates the proportion of size (a_m) that fits in the available knapsack area unoccupied by the objects in OPT. Obviously for $a_m \notin OPT$, size $(a_m) > B - size(OPT)$, otherwise $a_m \in OPT$ and $r_m \cdot \text{size}(a_m) + \text{size}(\text{OPT}) = B$. Since by assumption B and $\text{size}(a_i)$ for all $a_i \in S$ are integers, it holds that $\forall a_m \notin \text{OPT}$. $r_m \leq (\max_{a_k \in S} \{ \text{size}(a_k) \} - 1) / (\max_{a_k \in S} \{ \text{size}(a_k) \} \}$, so the generated liability for y_m is:

$$l_{y_m, x_m}^m = \max_{a_k \in S} \{ \text{size}(a_k) \} (1 - r_m) \\ \ge \max_{a_k \in S} \{ \text{size}(a_k) \} \left(1 - \frac{\max_{a_k \in S} \{ \text{size}(a_k) \} - 1}{\max_{a_k \in S} \{ \text{size}(a_k) \}} \right) = 1$$

So eventually, $\forall m \in N \setminus N_{\text{OPT}}$, $p_{y_m,x_m} = 1$. Now $r_{x_m} = 1$ thus the CDS (k_m, T, x_m) is not activated meaning that $p_{k_m,T} = 0$. From the above observations we conclude that the equity of T is $\sum_{j \in N_{\text{OPT}}} \text{profit}(a_j)$: node T receives money from all nodes that correspond to objects contained in OPT. We claim that this is the maximum equity node T can receive. If there exists a higher equity for T, then this must be generated from another profile \mathcal{P}'_i that corresponds to a solution to the original KNAPSACK instance with higher profit than the optimal one which is a contradiction.

For the opposite direction assume \mathcal{P}_0 to be the profile of 0 that maximises *T*'s equity. Let $A = \{a_j \mid p_{0j} = \text{size}(a_j)\}$ be the set of objects that corresponds to creditor nodes that 0 can fully pay. Obviously *A* can be computed in polynomial time from \mathcal{P}_0 . We claim that *A* is an optimal solution to the original KNAPSACK instance. Assume that there exists another set *A'* such that $\sum_{a_j \in A'} \text{size}(a_j) \leq B$ and $\sum_{a_j \in A'} \text{profit}(a_j) > \sum_{a_j \in A} \text{profit}(a_j)$. Now node 0 could rearrange its priorities by prioritising all creditors *j* for which $a_j \in A'$. Doing so, 0 can fully pay all nodes *j* for which $a_j \in A'$ size $(a_j) \leq B = e_0$ and node *T* receives $\sum_{a_j \in A'} \text{profit}(a_j) > \sum_{a_j \in A} \text{profit}(a_j)$, a contradiction to the original assumption that $\sum_{a_j \in A} \text{profit}(a_j)$ is the maximum equity for node *T*. \Box

Theorem 3. Finding a priority list profile that minimises the number of defaulting banks is NP-hard. Finding a priority list profile that minimises the number of not fully paid liabilities is NP-hard.

Proof. We prove both statements of the theorem via a reduction from the satisfiability problem (SAT), where we are given a boolean formula in conjunctive normal form, and have to determine whether there is a truth assignment to the variables that renders the formula true. Let $F = \bigwedge_{i=1}^{n} C_i$ be a SAT instance, where C_1, \ldots, C_n are the clauses, and let V_F be the set of all variables that appear in F. We create a financial network from F as follows. For each variable $x \in V_F$ we construct a gadget that is referred to as the *x*-subnetwork. Each *x*-subnetwork consists of four nodes, labelled as $i_x, x, \neg x, j_x$, where $e_{i_x} = 1$ and $e_x = e_{\neg x} = j_x = 0$, and of four debt contracts, $\mathcal{DC} = \{(i_x, x), (i_x, \neg x), (x, j_x), (\neg x, j_x)\}$ all with contract notionals equal to one. Moreover, the constructed financial network has *n* further nodes, labelled as C_1, \ldots, C_n , that correspond to each clause in *F* with $e_{C_1} = \cdots = e_{C_n} = 0$, and one terminal node labelled as τ , towards which each C_i holds a debt contract of notional one, i.e. $c_{C_i,\tau} = 1$. Finally for each variable *x* of *F*, we construct two nodes labelled as k_x and $k_{\neg x}$ respectively, where e_{k_x} and $e_{k_{\neg x}}$ are equal to the number of occurrences of literals *x* and $\neg x$ in *F*, respectively. Whenever a literal *l* belongs to a



Fig. 6. The financial system corresponding to the formula $F = \{x \lor \neg y\} \land \{y \lor \neg y\}$. This construction is used to prove the statements of Theorem 3.

clause C_i we construct the CDS $(k_l, C_i, \neg l)$ with $c_{k_l, C_i}^{\neg l} = 1$. An example of a network induced from $F = (x \lor y) \land (y \lor \neg y)$ is given in Fig. 6.

We now map a truth assignment $T: V_F \mapsto \{\text{true, false}\}\ \text{to a priority list profile } \mathcal{P}_T\ \text{as follows. If } T(x) = \text{true, node } i_x$ prioritises node x, and otherwise it prioritises node $\neg x$. All k_l nodes posses enough external assets to fully pay their debts under any priority list, so we can take an arbitrary list for those nodes, and all remaining nodes have at most one liability. Conversely, from a priority list profile \mathcal{P} we induce the truth assignment $T_{\mathcal{P}}$ as follows: if i_x prioritises x then T(x) = true otherwise T(x) = false.

Let $T: V_F \mapsto \{\text{true, false}\}\$ be any truth assignment and consider the priority list profile \mathcal{P}_T . In each *x*-subnetwork, node i_x can fully pay only its first priority, thus in each *x*-subnetwork there exist two defaulting nodes regardless of the choice of priority list of i_x , meaning that the minimum number of defaulting nodes in the induced financial system is $2|V_F|$. Similarly, each *x*-subnetwork has two not fully paid liabilities regardless of the choice of priority lost of i_x . The only additional defaulting nodes might be the nodes C_1, \ldots, C_n , and the only additional not fully paid liabilities might be on the *n* debt contracts in which one of C_1, \ldots, C_n is the debtor. Let us inspect which of the latter set of nodes is defaulting and which of the latter liabilities are not fully paid under \mathcal{P}_T . If clause C_i is a clause that is not satisfied under T, then none of the CDSes involving node C_i are activated, and node C_i does not have any assets \mathcal{P}_T . Since C_i has to pay 1 to τ , node C_i will be in default, and C_i 's liability of 1 will not be paid. If clause C_i is a clause that is satisfied under T, then at least one CDS involving node C_i is activated, and since the reference bank in this CDS has recovery rate 0, node C_i will receive the CDS's full notional of 1, with which it can fully pay its liability of 1. Hence, in the latter case, node C_i is not in default.

Hence, for a truth assignment T, under \mathcal{P}_T , the number of banks in default and the number of not fully paid liabilities are both equal to $2|V_F|$ plus the number of unsatisfied clauses. Since we argued above that restricting to the profiles

$\{\mathcal{P}_{T} \mid T \text{ is a truth-assignment for } F\}$

is without loss of generality, from finding the profile of priority lists minimising the number of defaulting banks or minimising the number of not fully paid liabilities in the constructed financial network, one can infer whether the formula F is satisfiable, which proves our claim. \Box

Theorem 4. Finding a priority list profile that minimises the number of activated CDSes is NP-hard.

Proof. We reduce from the problem of minimising the number of defaulting nodes, proved NP-hard in Theorem 3. Let \mathcal{F} be a financial system and let *m* be the maximum number of CDS contracts that are issued upon the same reference bank. Consequently, there exists a node that is the reference bank in *m* CDSes and no other node is reference bank to more CDSes than that.

We construct a financial network \mathcal{F}' by expanding \mathcal{F} in the following way. For each node *i* appearing in \mathcal{F} , we add a number n_i of CDS contracts such that *i* is reference bank to exactly *m* CDSes. These contracts are of the form $(\alpha_i^1, \beta_i^1, i), \ldots, (\alpha_i^{n_i}, \beta_i^{n_i}, i)$, with notionals $c_{\alpha_i,\beta_i}^i = 1$, where α_i^j, β_i^j for $j \in [n_i]$, are newly introduced nodes with external assets $e_{\alpha_i} = 1$. We call the new contracts dummy CDSes. Eventually all nodes in \mathcal{F}' are reference banks to exactly *m* CDSes. Fig. 7 shows an example of this expansion.

By this construction, it is straightforward to see that if under any priority list profile a set of banks default in \mathcal{F} , then in \mathcal{F}' , for each of these banks *i*, exactly *m* distinct CDSes activate in which *i* is the reference bank. Since the priority lists of \mathcal{F} and \mathcal{F}' are in one-to-one correspondence with each other, we conclude that finding the priority list profile minimising the number of activated CDSes in \mathcal{F}' is equivalent to finding the priority list profile minimising the number of defaulting banks in \mathcal{F} . \Box



Fig. 7. The initial financial system \mathcal{F} and the constructed \mathcal{F}' used is the proof of Theorem 4.



Fig. 8. Transformation of an (i, j) contract that appears in \mathcal{F} to a τ_{ij} -gadget appearing in \mathcal{F}'_{τ} and \mathcal{F}' .



Fig. 9. Transformation of an (i, j, k) contract that appears in \mathcal{F} to a τ_{ii}^k -gadget appearing in \mathcal{F}'_{τ} and \mathcal{F}' .

Systemic Liquidity. In [19], the authors define the term systemic liquidity as the total amount of payments that are being transacted among the economic firms in the financial system under some clearing recovery rate vector. Given a financial system \mathcal{F} we use the notation $\mathcal{L}_{\mathcal{F}}^{\mathcal{P}}(r)$ to denote the systemic liquidity of the system under the priority profile \mathcal{P} and an assuming clearing recovery rate vector r and is defined as $\mathcal{L}_{\mathcal{F}}^{\mathcal{P}}(r) = \sum_{i \in N} \sum_{j \in N} p_{i,j}(r)$. Like the previous objectives it turns out that choosing the profile that maximises the systemic liquidity in a financial network is NP-hard.

Theorem 5. Finding a priority list profile that maximises the systemic liquidity is NP-hard.

Proof. We prove the theorem by a reduction from the problem in Theorem 3. Given a financial system \mathcal{F} we construct a modified financial system \mathcal{F}' according to the following procedure:

- For each debt contract $(i, j) \in \mathcal{DC}_{\mathcal{F}}$ with contract notional $c_{i,j}$: Erase the debt contract (i, j). Add a new node denoted as τ_{ij} with $e_{\tau_{ij}} = 0$ and two new debt contracts (i, τ_{ij}) and $(\tau_{ij,j})$ each with contract notional $c_{i,\tau_{ij}} = c_{\tau_{ij},j} = c_{i,j}$. Finally add two new nodes α_{ij} and β_{ij} with $e_{\alpha_{ij}} = 2c_{i,j}$ and construct the CDS contract $(\alpha_{ij}, \beta_{ij}, \tau_{ij})$ with contract notional $c_{\alpha_{ij},j} = c_{i,j}$. Finally add two new nodes α_{ij} and β_{ij} with $e_{\alpha_{ij}} = 2c_{i,j}$ and construct the CDS contract $(\alpha_{ij}, \beta_{ij}, \tau_{ij})$ with contract notional $c_{\alpha_{ij},\beta_{ij}}^{\tau_{ij}} = c_{i,j}$. We refer to this construction as the τ_{ij} -gadget. It is illustrated in Fig. 8.
- For each CDS contract $(i, j, k) \in CDS_F$ with contract notional $c_{i,j}^k$: Erase the CDS (i, j, k). Add a new node denoted as τ_{ij}^k with $e_{\tau_{ij}^k} = 0$ and construct the CDS contract (i, τ_{ij}^k, k) with contract notional $c_{i,\tau_{ij}^k}^k = c_{ij}^k$ and the debt contract (τ_{ij}^k, j) with contract notional $c_{\tau_{ij},j} = c_{i,j}^k$. Finally we add two new nodes α_{ij}^k and β_{ij}^k with $e_{\alpha_{ij}^k} = 2c_{i,j}^k$ and construct the CDS $(\alpha_{ij}^k, \beta_{ij}^k, \tau_{ij}^k)$ with contract notional $c_{\tau_{ij},j}^{\tau_{ij}^k} = 2c_{i,j}^k$. We refer to this construction as the τ_{ij}^k -gadget. It is illustrated in Fig. 9.
- For each node $k \in N_{\mathcal{F}}$: We add five new nodes $\chi_k, \psi_k, \zeta_k, \alpha_k, \beta_k$ with $e_{\chi_k} = 1/(2 | N_{\mathcal{F}} |)$, $e_{\alpha_k} = 2$ and construct the CDS (χ_k, ψ_k, k) with $c_{\chi_k, \psi_k}^k = \infty$,⁵ the debt contract (ψ_k, ζ_k) with $c_{\psi_k, \zeta_k} = 1/(2 | N_{\mathcal{F}} |)$ and the CDS $(\alpha_k, \beta_k, \psi_k)$ with $c_{\alpha_k, \beta_k}^{\psi_k} = 2$. We refer to this construction as *k*-gadget. It is illustrated in Fig. 10.

⁵ Here we assume that a contract may have ∞ notional in the sense that whenever the debtor defaults it must submit all of its remaining assets through this contract. A similar assumption is made in [25,24].



Fig. 10. Addition of the *k*-gadget in \mathcal{F}' for every node *k* of \mathcal{F} .

For any priority profile \mathcal{P} of \mathcal{F} we define its 'natural extension' priority profile denoted as \mathcal{P}^{ext} of \mathcal{F}' as follows: For a node $i \in N_F$ having a singleton liability priority list \mathcal{P}_i we substitute each priority that corresponds to some debt contract $(i, j) \in \mathcal{DC}_{\mathcal{F}}$ with the debt contract $(i, \tau_{ij}) \in \mathcal{DC}_{\mathcal{F}'}$ and each priority that corresponds to some CDS contract $(i, j, k) \in \mathcal{DC}_{\mathcal{F}}$ CDS_F with the CDS contract $(i, \tau_{ii}^k, k) \in CDS_{F'}$. For all other nodes appearing in the constructed gadgets, the priority lists are uniquely defined since they have only one outgoing edge. For example, if some node i pays in \mathcal{F} according to $\mathcal{P}_i = ((i, j) \mid (i, j, k))$, then it pays in \mathcal{F}' according to $\mathcal{P}_i^{\text{ext}} = ((i, \tau_{ij}) \mid (i, \tau_{ij}^k, k))$. Next we prove three useful claims that we use in our reduction.

Claim 1. The 'natural extension' priority profile does not affect the recovery rate of nodes belonging in $N_{\mathcal{F}} \cap N_{\mathcal{F}'}$. Namely, if node i has recovery rate r_i under \mathcal{P} in \mathcal{F} , then it has recovery rate r_i under \mathcal{P}^{ext} in \mathcal{F}' .

Proof. The amount of money transferred via a liability (i, j) in \mathcal{F} under \mathcal{P} are transferred from τ_{ij} to node j under \mathcal{P}^{ext} in \mathcal{F}' , since node τ_{ij} has an incoming payment of at most $c_{i,j}$, a liability of $c_{i,j}$ and zero external assets. Similarly the amount of money that are being transferred via a CDS contract (i, j, k) in \mathcal{F} under \mathcal{P} are transferred from τ_{ij}^k to node j in \mathcal{F}' under \mathcal{P}^{ext} because τ_{ij}^k receives a payment of at most $(1 - r_k)c_{i,j}^k$, has a liability of $c_{i,j}^k$ and zero external assets. Also by the way we constructed \mathcal{F}' the liabilities of all nodes appearing in \mathcal{F} are unchanged. That means that the 'natural extension' profile \mathcal{P}^{ext} of a profile \mathcal{P} does not affect the assets and liabilities of nodes in $N_{\mathcal{F}} \cap N_{\mathcal{F}'}$, thus their recovery rate is the same. \Box

Next we denote by \mathcal{F}'_{τ} the financial system that is constructed from \mathcal{F} only by adding the τ_{ij} -gadget and τ^k_{ij} -gadget. We denote by \mathcal{P}_{τ} the 'natural extension' of any priority profile \mathcal{P} of \mathcal{F} in \mathcal{F}_{τ} . In the following claim we prove that the liquidity of \mathcal{F}'_{τ} is the same under any priority profile and any clearing recovery rate vector.

Claim 2. For each priority profile \mathcal{P} and each clearing recovery rate vector r, it holds that $\mathcal{L}_{\mathcal{F}'}^{\mathcal{P}}(r) = 2\left(\sum_{i,j\in N_{\mathcal{F}}} c_{i,j} + \sum_{i,j,k\in N_{\mathcal{F}}} c_{i,j}^k\right)$

Proof. Assume a priority profile \mathcal{P} of \mathcal{F} and let \mathcal{P}_{τ} be its 'natural extension' profile in \mathcal{F}_{τ} . First we will prove that each $(i, j) \in \mathcal{DC}_{\mathcal{F}}$ generates liquidity of $2c_{i, j}$ in $(\mathcal{F}_{\tau}, \mathcal{P}_{\tau})$. Assume $l \leq c_{i, j}$ to be the amount of money transferred via some debt contract (i, j) in $(\mathcal{F}, \mathcal{P})$. We distinguish two cases.

- 1. $l < c_{i,j}$. The recovery rate for node τ_{ij} is $r_{\tau_{ij}} = l/c_{i,j} < 1$. Thus the CDS $(\alpha_{ij}, \beta_{ij}, \tau_{ij})$ is activated and node α_{ij} owes $2(1 - r_{\tau_{ij}})c_{i,j} = 2c_{i,j} - 2l$ to node β_{ij} which can fully pay off. Edges (i, τ_{ij}) and (τ_{ij}, j) generate a liquidity of 2l thus the generated liquidity in that case is $2c_{i,j} - 2l + 2l = 2c_{i,j}$.
- 2. $l = c_{i,j}$. The recovery rate of node τ_{ij} is $r_{\tau_{ij}} = 1$. Thus the $(\alpha_{ij}, \beta_{ij}, \tau_{ij})$ is not activated. The generated liquidity in that case arises only from edges (i, τ_{ij}) and (τ_{ij}, j) and is equal to $2c_{i,j}$.

Next we will prove that any CDS contract $(i, j, k) \in CDS_F$ generates a liquidity of $2c_{i,j}^k$ in $(\mathcal{F}_{\tau}, \mathcal{P}_{\tau})$. Assume a clearing recovery rate vector r and let $l \leq (1 - r_k)c_{i,j}^k$ to be the amount of money transferred via some CDS (i, j, k) in $(\mathcal{F}, \mathcal{P})$. We distinguish three cases.

- 1. $l = (1 r_k)c_{i,j}^k$. The recovery rate for node τ_{ij} is $r_{\tau_{ij}} = l/c_{i,j}^k = 1 r_k < 1$. Thus the CDS $(\alpha_{ij}^k, \beta_{ij}^k, \tau_{ij}^k)$ is activated and node α_{ij}^k owes $2(1 r_{\tau_{ij}^k})c_{i,j}^k = 2r_k c_{i,j}^k$, which can fully pay off. Since edges $(i, \tau_{i,j})$ and (τ_{ij}, j) generate a liquidity of $2(1 - r_k)c_{i,j}^k \text{ the total generated liquidity is } 2r_kc_{i,j}^k + 2c_{i,j}^k - 2r_kc_{i,j}^k = 2c_{i,j}^k.$ 2. $l < (1 - r_k)c_{i,j}^k$. Again $r_{\tau_{i,j}} = l/c_{i,j}^k < 1$. Thus $(\alpha_{ij}^k, \beta_{ij}^k, \tau_{ij}^k)$ is activated and α_{ij}^k owes $2c_{i,j}^k(1 - (l/c_{i,j}^k)) = 2c_{i,j}^k - 2l$ to β_{ij}^k .
- which can fully pay. The total generated liquidity is $2l + 2c_{i,j}^k 2l = 2c_{i,j}^k$.
- 3. $r_k = 1$. From Claim 1, we know that the recovery rate of any node does not change under \mathcal{P}^{ext} . It is not hard to see that the same holds under \mathcal{P}_{τ} . If $r_k = 1$ then $r_{\tau_{ij}^k} = 0$ and the (i, τ_{ij}^k, k) is inactive. So node α_{ij}^k owes $2c_{i,j}^k$ to β_{ij}^k , which can fully pay and the generated liquidity is $2c_{i,j}^k$. \Box

Notice that \mathcal{F}' is actually \mathcal{F}_{τ} with the addition of the *k*-gadgets, and since we proved that the systemic liquidity of \mathcal{F}' is always the same under any priority profile it must be the case that the liquidity if \mathcal{F}' depends on the generated liquidity in the *k*-gadgets.

Claim 3. The generated liquidity from a k-gadget under any clearing recovery rate vector is $1/(|N_{\mathcal{F}}|)$ whenever $r_k < 1$ and 2 whenever $r_k = 1$.

Proof. Assume some node *k* and let a clearing recovery rate vector with $r_k = 1$ in \mathcal{F} under some profile \mathcal{P} . From Claim 1 we get that $r_k = 1$ in \mathcal{F}' under \mathcal{P}^{ext} . Thus the CDS (χ_k, ψ_k, k) is inactive which means that $r_{\psi_k} = 0$ and node α_k owes 2 to node β_k which it can fully pay off and the generated liquidity in that case is 2. If $r_k < 1$ then the CDS (χ_k, ψ_k, k) is activated and since $c_{\chi_k, \psi_k}^k = \infty$ node χ_k must submit all of its assets to ψ_k . Now ψ_k can fully pay off ζ_k thus the ($\alpha_k, \beta_k, \psi_k$) is inactive and the generated liquidity is $1/|N_{\mathcal{F}}|$. \Box

Next we provide the reduction based on the above three claims. Assume \mathcal{P} is the priority profile of \mathcal{F} under which the number of defaulting nodes is minimised and let λ be that number. We construct the financial system \mathcal{F}' as described above and consider the priority profile \mathcal{P}^{ext} . We will prove that under \mathcal{P}^{ext} the liquidity in \mathcal{F}' is maximised. From Claim 1 we know that the recovery rate of each node in \mathcal{F} under \mathcal{P}^{ext} is unchanged thus if a node is in default in \mathcal{F} under \mathcal{P} for some clearing recovery rate vector then it is in default in \mathcal{F}' under \mathcal{P}^{ext} . That means that there exist exactly λ *k*-gadgets, which by Claim 3 generate liquidity $\lambda / |N_{\mathcal{F}}|$, which (taking into consideration Claim 2) means that

$$\mathcal{L}_{\mathcal{F}'}^{\mathcal{P}^{\text{ext}}}(r) = 2\left(\sum_{i,j\in N_{\mathcal{F}}} c_{i,j} + \sum_{i,j,k\in N_{\mathcal{F}}} c_{i,j}^{k}\right) + \frac{\lambda}{\mid N_{\mathcal{F}}\mid} + 2(\mid N_{\mathcal{F}}\mid -\lambda).$$

Assume there exists another profile Q^{ext} such that $\mathcal{L}_{\mathcal{F}'}^{Q^{\text{ext}}}(r) > \mathcal{L}_{\mathcal{F}'}^{\mathcal{P}^{\text{ext}}}(r)$. This means that there must exist another λ' such that

$$2\left(\sum_{i,j\in N_{\mathcal{F}}}c_{i,j} + \sum_{i,j,k\in N_{\mathcal{F}}}c_{i,j}^{k}\right) + \frac{\lambda'}{|N_{\mathcal{F}}|} + 2(|N_{\mathcal{F}}| - \lambda') > 2\left(\sum_{i,j\in N_{\mathcal{F}}}c_{i,j} + \sum_{i,j,k\in N_{\mathcal{F}}}c_{i,j}^{k}\right) + \frac{\lambda}{|N_{\mathcal{F}}|} + 2(|N_{\mathcal{F}}| - \lambda),$$

which is true if and only if $\lambda' < \lambda$. This means that the natural extension of the priority profile that minimises the number of defaulting nodes in \mathcal{F} maximises the systemic liquidity in \mathcal{F}' and vice versa. \Box

5. Conclusions and future work

Financial networks have emerged as a fertile research area in both computational complexity and algorithmic game theory. It is paramount to understand systemic risk in finance from both these perspectives. In this paper, we join both streams of work by settling questions around the computational complexity of systemic risk for priority payments, a scheme so far only studied from the game-theoretic point of view. In an interesting parallel with the state of the art for proportional payments, we prove that computing clearing recovery rates is FIXP-complete whereas it is NP-hard to compute the priority lists optimising several measures of financial health of the system.

Our work paves the way for studying payment schemes in financial networks in more detail. Is there a payment scheme for financial networks with derivatives that makes the computation of systemic risk easy and/or that induces "nice" equilibria? We also wonder the extent to which the flexibility of working with financial networks can lead to a deeper understanding of FIXP; e.g., are there payment schemes that can be proved complete for variants of FIXP defined upon a different basis?

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Full details of the pre-processing step in the proof of Theorem 1

We here present the full details as presented in [18] to make this paper self-contained.

Let $I \in \Pi$ be an instance, let $F_I : [0, 1]^n \to [0, 1]^n$ be *I*'s associated fixed point function, and let C_I be the algebraic circuit corresponding to F_I . As a pre-processing step, we convert C_I to an equivalent alternative circuit C'_I that satisfies that all

the signals propagated by all gates in C'_{I} and all the used rational constants in C'_{I} are contained in the interval [0, 1]. The transformed circuit C'_{I} may contain two additional types of gates: Division gates and gates that computes the absolute value of the difference of two operands. We will refer to the latter type of gate as an *absolute difference gate*. The circuit C'_{I} will not contain any subtraction gates, and will not contain max and min gates either.

The transformation procedure for C_I follows the same approach of the transformation given in Theorem 4.3 of [8] where the 3-Player Nash equilibrium problem is proved FIXP-complete, and borrows some important ideas from there. Nonetheless, there are important differences in our transformation, starting with the fact that we use a different set of types of gates in our circuit.

First, transforming C_I into a circuit with only non-negative rational constants is trivial, since we can introduce subtraction gates combined with the rational constant 0 (which we will get rid of later on in the transformation process).

As a next step, we remove all min and max gates and replace them with addition, subtraction, multiplication, and absolute difference gates through the identities $\max\{a, b\} = (1/2) \cdot ((a+b) + |a-b|)$ and $\min\{a, b\} = (1/2) \cdot ((a+b) - |a-b|)$. Let the resulting circuit be C''_{I} .

The third step in the transformation is to move all the subtraction gates to the top of the circuit, right before the output gates, which results in exactly one subtraction gate being present for each output variable. This step is realised by representing each (potentially negative) signal *s* in $C_1^{"}$ that is propagated between two arithmetic gates, by two separate signals s^+ , $s^- \ge 0$, which represent the positive and negative parts of *s* respectively, and are guaranteed to have values such that $s^+ - s^- = s$: Each arithmetic gate g_i in the circuit is replaced (sequentially, from the bottom of the circuit upward) by gates that operate on the separate positive and negative parts of the original input signals as follows.

Suppose g_i is a node outputting a constant rational value c > 0, then we may replace g_i by two constant nodes, one which outputs c, the positive part of the resulting separated signal, and one which outputs 0, representing the negative part of the signal.

Suppose next that g_i is an addition gate operating originally on the outputs s and t of two other gates. Then, we use the observation that for the two signals s and t, separated into positive and negative parts s^+ , s^- , t^+ , t^- , it holds that $(s^+ - s^-) + (t^+ - t^-) = (s^+ + t^+) - (s^- + t^-)$. Therefore, g_i can be replaced by two addition gates g_i^+ and g_i^- , where g_i^+ operates on signals s^+ and t^+ , and g_i^- operates on gates s^- and t^- .

If g_i is a multiplication gate, note that for two inputs s and t, separated into positive and negative parts s^+ , s^- , t^+ , t^- , it holds that $(s^+ - s^-) \cdot (t^+ - t^-) = (s^+t^+ + s^-t^-) - (s^+t^- + s^-t^+)$, so we may similarly replace each multiplication gate with four multiplication gates and two addition gates accordingly.

When g_i is an absolute difference gate, we note that for two inputs s and t, separated into positive and negative parts s^+, s^-, t^+, t^- , it holds that $|(s^+ - s^-) - (t^+ - t^-)| = |(s^+ + t^-) - (s^- + t^+)| = 0$ so that we can replace g_i with two addition gates, one absolute difference gate, and one constant gate outputting 0 (representing the negative part of the resulting signal).

Lastly, when g_i is a subtraction gate in C''_i , we remove it and connect the gates that provide the inputs to g_i appropriately to the subsequent gates that g_i points to. The final subtraction gates introduced in the top layer of the circuit are directly connected to the output gates, and subtract the positive and negative parts of the final output signal, so as to generate the correct *n* outputs. Since the function F_I maps $[0, 1]^n$ to itself, and the resulting circuit is equivalent to the original circuit C_I , we now have a circuit where all signals in the circuit are non-negative, and where we have introduced the absolute difference operation as an additional type of gate into the circuit. Next, we perform a straightforward step that eliminates the remaining *m* subtraction gates altogether, by replacing them with absolute difference gates. This gives us an equivalent circuit, as we know that the result of the *n* subtractions at the top of the circuit is positive for all possible inputs. We call this resulting circuit without subtraction gates C''_I .

this resulting circuit without subtraction gates C_1''' . The last step is to transform the circuit C_1''' into a circuit where all signals inside the circuit are in [0, 1] for all possible inputs to the circuit. To do so, in case there are rational constants exceeding 1 present in the circuit, we first multiply all the rational constants by the inverse $c \in [0, 1]$ of the largest rational constant in the circuit. We add the appropriate gates at the start of the circuit that multiply all input gates by c, and we add division gates at the end of the circuit that divide the final signal by c, before being propagated to the output gates. Furthermore, for each multiplication gate, we add a division gate that divides its result by c. This results in a circuit where all rational constant nodes are in [0, 1], and all internal signals of the original circuit are essentially scaled by a factor of c. The resulting circuit is thus equivalent to C_1''' .

Now that we have that all the rational constants are in [0, 1], our final step is transforming the circuit further in such a way that all signals inside the circuit are contained in [0, 1] as well. Observe that the magnitude of any signal in the circuit is at most doubly exponential in the size of C_1'' (this can be attained when a certain signal s > 1 goes through a chain of successive squaring operations, using multiplication gates). We thus perform the following final transformation to enforce that all signals stay in [0, 1]: Let *d* be a number, bounded by a polynomial in |I|, such that all signals in the circuit are strictly smaller than 2^{2^d} for every input in $[0, 1]^n$. We add to the start of the circuit an auxiliary circuit *T* that computes $t = 1/2^{2^d}$ by successively squaring the constant 1/2 a number of *d* times, and we use *T* right after each input node and right after each rational constant node in order to multiply every input signal and every rational constant node by *t*. For

⁶ Note that the division by *c* that we introduce right after every multiplication gate is needed because multiplying two scaled signals $c \cdot s$ and $c \cdot s'$ results in $c^2 ss'$, hence dividing this result by *c* results in the desired output signal css'.

each multiplication gate, we add a division gate such that its result is divided by t. Lastly, at the end of the circuit, we add a division gate just before each output gate, that divides the final signal by t. We now observe that each signal in the new circuit has effectively been multiplied by a factor of t, with exception of the signals directly propagated by the input gates, and directly entering the output gates, which still retain their original values.

This completes our pre-processing steps on the algebraic circuit, and we denote the resulting circuit by C'_I . The circuit C'_I has the desired properties that we are looking for: All signals in the circuit are in [0, 1] regardless of the input. Furthermore, the applied transformation ensures that C'_I is equivalent to the original C_I and thus represents F_I . The circuit can be obtained in a polynomial number of computation steps from C_I and is thus polynomial-time computable from the instance I. The circuit C'_I consists of $\{+, *, /\}$ -gates, as well as absolute difference gates that compute the absolute value of the difference of two inputs. Lastly, there are rational constant nodes in the circuit where all such constants are in [0, 1].

Appendix B. Financial system gadgets



Fig. 11. Inversion gadget g_{inv} : Computing 1 - r from r [17].



Fig. 12. Duplication (and multiplication-by-constant) gadget g_{dup} : This gadget outputs two values: r and cr, where $c \in [0, 1]$ is a rational constant. Choosing c = 1 yields a duplication gadget that takes the input recovery rate r and outputs r as the two recovery rates of the output nodes. We may also denote this gadget by $g_{c\cdot r}$, for $c \in [0, 1]$ in case this gadget is used to multiply the input by a constant c [17].



Fig. 13. Minimum gadget g_{\min} , computing $\min\{r_1, r_2\}$. In this gadget we assume node 6 pays according to the priority list $P_6 = ((6, 7, 4) | (6, 8))$. Node 6 prioritises the liability it holds towards node 7. This liability is generated by the CDS contract (6, 7, 4) and is equal to $l_{6,7}(r) = 1 - r_4$. The recovery rate of node 4 is equal to $1 - r_2$ since node 3 holds a debt of $1 - r_2$ against 4 and $e_3 = 1$. Consequently $l_{6,7}(r) = r_2$. Since node 6 receives a payment of r_1 from the second input bank and prioritises the contract (6, 7, 4), it submits to node 7 the min $\{r_1, r_2\}$. Node 8 will receive the remaining assets of node 6, which are calculated as max $\{0, r_1 - r_2\}$. It's worth noting that if we were to direct the output of g_{\min} to node 8 instead, we would generate the Positive Subtraction Gadget g_{pos-} shown in Fig. 4b.



Fig. 14. Maximum gadget g_{max} , computing $max\{r_1, r_2\}$. This is a compact representation where the nodes labelled with a subscripted *g* have to be replaced by copies of the respective gadgets, in order to obtain the full financial system defining the gadget. This gadget exploits the fact that $max\{a, b\} = 1 - min\{1 - a, 1 - b\}$ for $a, b \in [0, 1]$.



Fig. 15. Absolute difference gadget g_{abs} : This is a compact representation where the nodes labelled with a subscripted *g* have to be replaced by copies of the respective gadgets, in order to obtain the full financial system defining the gadget. The recovery rate of the output node is $|r_1 - r_2|$, where r_1 and r_2 are the recovery rates of the input nodes. The gadget is formed by first applying two g_{pos-} gates, the first on input (r_1, r_2) and the second on input (r_2, r_1) . The two positive subtraction gadgets g_{pos-} compute max $\{r_1 - r_2, 0\}$ and max $\{r_2 - r_1, 0\}$, after which these two maxima are added together using g_+ , resulting in the desired output $|r_1 - r_2|$.



Fig. 16. Multiplication gadget g_* : This gadget's construction is rather involved and makes use of various instances of multiplication-by-constant gadgets and positive subtraction gadgets. Some of the nodes have been annotated with expressions marked in red: These expressions are the recovery rates of some of the intermediate nodes in the gadget, in terms of the input nodes' recovery rates. We have included these expressions so that it is more straightforward for the reader to verify that the gadget's output node has the desired recovery rate. The function of nodes 1 to 15 is to compute $1/(1 + r_2)$, which is subsequently transformed into the expression $r_1/4 + r_1r_2/4 + 1/32 + (1/32)r_2$ in node 27. After node 27, it is then straightforward to extract the term r_1r_2 from the latter expression by making use of multiplication-by-constant gadgets and positive subtraction gadgets. Some of the constants appearing in this gadget (particularly the factor 1/4 and the external assets of 1/32 in node 20) have the function to keep the expressions (marked in red) small enough so that they remain in the interval [0, 1] and can thus take the form of a valid recovery rate. The presence of positive external assets at node 20 is also needed to ensure non-degeneracy, and this is also the reason for the presence of some of the debt contracts in this gadget.



Fig. 17. Square Root gadget $g_{\sqrt{r}}$: On input *r* the recovery rate of the output node is \sqrt{r} . The gadget works as follows, nodes 4 and 6 receive a payment of 1 - r from nodes 3 and 5 respectively and pass them on to nodes 7 and 10 respectively. From Fig. 1, we know that $r_7 = r_{10} = 1 - \sqrt{r}$, thus generating a liability of \sqrt{r} to node 13 which it can fully pay off, so that the terminal node receives a payment of \sqrt{r} .

B.1. A square root gadget for division

To obtain our division gadget, we will essentially get rid of division gates altogether in C'_I , and replace each of them by a set of alternative operations that achieve the same result. We thus make a few modifications to C'_I . This argument is originally given in [17] and only reported here for completeness.

The first modification is that we alter the pre-processing step to generate a value $t = 1/2^{2^d}$, which scales all the signals in the circuit so that each signal is in [0, 1] irrespective of the input vector. Here, *d* is a polynomial time computable value that satisfies that every signal in the original circuit C_I is at most 2^{2^d} . We then scaling factor to $1/(2^{1+2^d})$ instead of $1/2^{2^d}$. This can be done by adding one additional multiplication gate at the end of the pre-processing. Let the output of the modified pre-processing be t' = t/2. After this modification, it holds that all signals inside the circuit are in [0, 1/2].

We now make use of the fact that division gates are used in a very limited way in C'_{I} .

- First, division is used for dividing by c, where c is a given explicit constant in the original circuit C_I . For such divisions, we can simply use our constant multiplication gadget g_{dup} to multiply by 1/c, which is equivalent to dividing by c.
- Secondly, division is used to divide certain outputs of gates by t' (previously t), where $t' = 1/2^{1+2^d}$. This type of division happens in two cases.
 - 1. At every point in the circuit where two scaled signals t'a and t'b with $a, b \le 2^{2^d}$ are multiplied with each other, resulting in the value t't'ab. This value is divided by t' in order to generate the signal t'ab, i.e., a scaled version of the signal ab in the original circuit.
- 2. At the end of the circuit, where a scaled signal t'a is divided by t' to produce an output signal of the original circuit, which is in [0, 1].

In the first case, let x = t'ab and in the second case, let x = a, so that in both cases a number t'x is divided by t' to result in x, and in both cases it holds that $x \in [0, 1]$. We replace the division t'x/t' by a sequence of d gates that compute the square root of its input, followed by a multiplication by 2, followed by a sequence of d successive squaring multiplication gates, followed by a final multiplication by 2. This results in the correct value

$$((t'x)^{1/2^d} \cdot 2)^{2^d} \cdot 2 = (t'^{1/2^d}x^{1/2^d} \cdot 2)^{2^d} \cdot 2 = t'x \cdot 2^{2^d} \cdot 2 = \frac{1}{2}x \cdot 2 = x.$$

It is furthermore straightforward to verify that, due to the order in which we apply our arithmetic operations, all of the values throughout this computation stay in the interval [0, 1]. We note that the adapted scaling factor t' = t/2 is needed because of the multiplication by 2 that is executed after the successive square roots and before the successive squaring.

The resulting circuit has no division gates anymore, so we do not need a division gadget in our reduction. Instead, now we need a square root gadget, which is presented in Fig. 17.

References

- [1] D. Acemoglu, A. Ozdaglar, A. Tahbaz-Salehi, Systemic risk and stability in financial networks, Am. Econ. Rev. 105 (2) (2015) 564-608.
- [2] A. Bertoni, G. Mauri, N. Sabadini, A characterization of the class of functions computable in polynomial time on random access machines, in: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing (STOC 1981), Association for Computing Machinery, 1981, pp. 168–176.

- [3] N. Bertschinger, M. Hoefer, D. Schmand, Strategic payments in financial networks, in: 11th Innovations in Theoretical Computer Science Conference (ITCS 2020), in: LIPIcs, vol. 151, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 46.
- [4] A. Borodin, R. Fagin, J.E. Hopcroft, M. Tompa, Decreasing the nesting depth of expressions involving square roots, J. Symb. Comput. 1 (2) (1985) 169–188.
- [5] R. Cifuentes, G. Ferrucci, H.S. Shin, Liquidity risk and contagion, J. Eur. Econ. Assoc. 3 (2–3) (2005) 556–566.
- [6] L. Eisenberg, T.H. Noe, Systemic risk in financial systems, Manag. Sci. 47 (2) (2001) 236-249.
- [7] M. Elliott, B. Golub, M.O. Jackson, Financial networks and contagion, Am. Econ. Rev. 104 (10) (2014) 3115–3153.
- [8] Kousha Etessami, Mihalis Yannakakis, On the complexity of Nash equilibria and other fixed points, SIAM J. Comput. 39 (6) (2010) 2531–2597.
- [9] A. Filos-Ratsikas, Y. Giannakopoulos, A. Hollender, P. Lazos, D. Poças, On the complexity of equilibrium computation in first-price auctions, in: Proceedings of the 22nd ACM Conference on Economics and Computation (EC 2021), Association for Computing Machinery, 2021, pp. 454–476.
- [10] A. Filos-Ratsikas, K.A. Hansen, K. Høgh, A. Hollender, FIXP-membership via convex optimization: games, cakes, and markets, in: IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS 2021), IEEE Computer Society, 2022, pp. 827–838.
- [11] M.R. Garey, R.L. Graham, D.S. Johnson, Some NP-complete geometric problems, in: Proceedings of the Eighth Annual ACM Symposium on Theory of Computing (STOC 1976), Association for Computing Machinery, 1976, pp. 10–22.
- [12] P. Glasserman, H.P. Young, How likely is contagion in financial networks?, J. Bank. Finance 50 (2015) 383-399.
- [13] P.W. Goldberg, A. Hollender, The hairy ball problem is PPAD-complete, J. Comput. Syst. Sci. 122 (2021) 34-62.
- [14] S. Heise, R. Kühn, Derivatives and credit contagion in interconnected networks, Eur. Phys. J. B 85 (4) (2012) 1–19.
- [15] B. Hemenway, S. Khanna, Sensitivity and computational complexity in financial networks, Algorithmic Finance 5 (3-4) (2016) 95-110.
- [16] D. Hu, J.L. Zhao, Z. Hua, M.C.S. Wong, Network-based modeling and analysis of systemic risk in banking systems, Manag. Inf. Syst. Q. (2012) 1269-1291.
- [17] S.D. Ioannidis, B. de Keijzer, C. Ventre, Strong approximations and irrationality in financial networks with derivatives, in: 49th International Colloquium on Automata, Languages, and Programming (ICALP 2022), in: LIPIcs, vol. 229, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022, 76.
- [18] Stavros D. Ioannidis, Bart de Keijzer, Carmine Ventre, Strong approximations and irrationality in financial networks with financial derivatives, CoRR, arXiv:2109.06608 [abs], 2021.
- [19] P. Kanellopoulos, M. Kyropoulou, H. Zhou, Forgiving debt in financial network games, in: Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI 2022), International Joint Conferences on Artificial Intelligence Organization, 2022, pp. 335–341.
- [20] Panagiotis Kanellopoulos, Maria Kyropoulou, Hao Zhou, Financial network games, in: ICAIF, Association for Computing Machinery, 2021, 26.
- [21] C.H. Papadimitriou, The Euclidean traveling salesman problem is NP-complete, Theor. Comput. Sci. 4 (3) (1977) 237-244.
- [22] P.A. Papp, R. Wattenhofer, Network-aware strategies in financial systems, in: 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020), in: LIPIcs, vol. 168, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020, 91.
- [23] P.A. Papp, R. Wattenhofer, Debt swapping for risk mitigation in financial networks, in: Proceedings of the 22nd ACM Conference on Economics and Computation (EC 2021), Association for Computing Machinery, 2021, pp. 765–784.
- [24] P.A. Papp, R. Wattenhofer, Sequential defaulting in financial networks, in: 12th Innovations in Theoretical Computer Science Conference (ITCS 2021), in: LIPIcs, vol. 185, Schloss Dagstuhl-Leibniz-Zentrum f
 ür Informatik, 2021, 52.
- [25] P.A. Papp, R. Wattenhofer, Default ambiguity: finding the best solution to the clearing problem, in: Web and Internet Economics. WINE 2021, in: Lecture Notes in Computer Science, vol. 13112, Springer, 2022, pp. 391–409.
- [26] L.C.G. Rogers, L.A.M. Veraart, Failure and rescue in an interbank network, Manag. Sci. 59 (4) (2013) 882-898.
- [27] A. Schönhage, On the power of random access machines, in: 8th International Colloquium on Automata, Languages and Programming (ICALP 1979), vol. 71, Springer, 1979, pp. 520–529.
- [28] S. Schuldenzucker, S. Seuken, Portfolio compression in financial networks: incentives and systemic risk, in: Proceedings of the 21st ACM Conference on Economics and Computation (EC 2020), Association for Computing Machinery, 2020, p. 79.
- [29] S. Schuldenzucker, S. Seuken, S. Battiston, Clearing payments in financial networks with credit default swaps, in: Proceedings of the 2016 ACM Conference on Economics and Computation (EC 2016), Association for Computing Machinery, 2016, p. 759.
- [30] S. Schuldenzucker, S. Seuken, S. Battiston, Finding clearing payments in financial networks with credit default swaps is PPAD-complete, in: 8th Innovations in Theoretical Computer Science Conference (ITCS 2017), vol. 67, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017, 32.
- [31] P. Tiwari, A problem that is easier to solve on the unit-cost algebraic RAM, J. Complex. 8 (4) (1992) 393-397.
- [32] M. Yannakakis, Equilibria, fixed points, and complexity classes, in: STACS 2008, HAL, 2008, pp. 19-38.