Advancing Receivable Financing via a Network-based Approach

Ilaria Bordino, Francesco Gullo, and Giacomo Legnaro

Abstract—Receivable financing is the process whereby cash is advanced to firms against receivables their customers have yet to pay: a receivable can be sold to a funder, which immediately gives the firm cash in return for a small percentage of the receivable amount as a fee. Receivable financing has been traditionally handled in a centralized way, where every request is processed by the funder individually and independently of one another.

In this work we propose a novel, network-based approach to receivable financing, which enables customers of the same funder to autonomously pay each other as much as possible, and gives benefits to both the funder (reduced cash anticipation and exposure risk) and its customers (smaller fees and lightweight service establishment). Our main contribution consists in providing a principled formulation of the network-based receivable-settlement strategy, and showing how to achieve all algorithmic challenges posed by the design of this strategy. We formulate network-based receivable financing as a novel combinatorial-optimization problem on a multigraph of receivables. We show that the problem is $\mathbf{NP}$-hard, and devise an exact branch-and-bound algorithm, as well as algorithms to efficiently find effective approximate solutions. Our more efficient algorithms are based on cycle enumeration and selection, and exploit a theoretical characterization in terms of a knapsack problem, as well as a refining strategy that properly adds paths between cycles. We also investigate the real-world issue of avoiding temporary violations of the problem constraints, and design methods for handling it. An extensive experimental evaluation is performed on real receivable data. Results attest the good performance of our methods.

Index Terms—receivable financing, graph theory, combinatorial optimization, network flow.

1 Introduction

The term *receivable* indicates a debt owed to a creditor, which the debtor has not yet paid for.

Receivable Financing (RF) is a service that allows creditors to *sell* receivables to a *funder* or *financing company*. The funder, which is typically a bank, a financial institution, or a digital platform (e.g., *BlueVine*, *Fundbox*, *C2FO*, and *MarketInvoice*), anticipates (part of) the receivable amount, deducting a percentage as a fee. Creditors resort to RF to obtain cash anticipation, reducing the waiting times of receivable payment, which generally range from 30 to 120 days. Being forced to face such long waits may be challenging for businesses, as not knowing when their credits are coming in may affect their capability of planning ahead. Conversely, receiving early payments enables businesses to mitigate the cash-flow issues involving receivables. A further advantage is that funders typically offer credit control as well, relieving creditors from the burden of chasing up debtors.

A network-based perspective. Existing RF services employ an inherently *client-server* perspective: the funder, just like a centralized server, receives multiple funding requests, and processes each of them individually.

It is easy to observe that this strategy has an obvious limitation, i.e., it overlooks the fact that a set of receivables for which the RF service is simultaneously requested compose a *network where a customer may act as the creditor or the debtor* of different receivables. In this work we present a novel receivable-financing method, which leverages such a network perspective to enable an autonomous cash flow among customers. For instance, as shown in Figure 1, a network-based RF service might allow the funder to anticipate no money, while still having RF accomplished for all the receivables. A real scenario is clearly more complicated than the simple one depicted in the example. This means that the receivables that can be autonomously paid by the customers without involving the funder are typically only a subset of all the receivable which the RF service is requested for. The main goal in the design of network-based RF is therefore *to select the largest-amount subset of receivables for which autonomous payment is possible*.

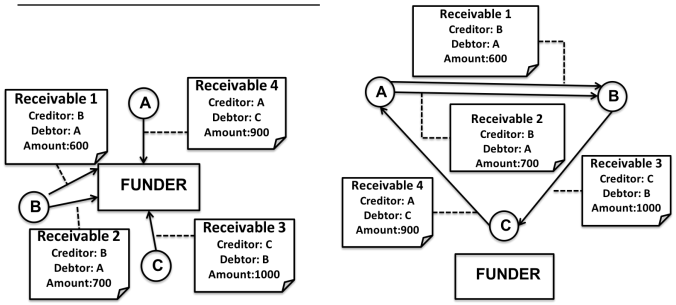


Fig. 1: Client/server (left) vs. network-based (right) receivable financing. In the first case, the funder handles each request individually, paying a total amount of 3200. In the network-based scenario, customers are allowed to pay each other. Assume A, B, C have 0, 1000, 0 on their accounts, respectively. Then, B has sufficient money to pay Receivable 3, which in turns makes C have enough money to pay Receivable 4, and so on, until all receivables have been paid. This way no money is anticipated by the funder.

Network-based RF provides noticeable advantages to both the funder and the customers. In fact, the funder can now entrust to the novel network-based settlement system the settlement of the receivable financing requests advanced by the customers who accept to enroll into it. The system takes care to enable autonomous payment of these receivables to the maximum extent possible. This means that the funder no longer needs to anticipate cash for each

• *I. Bordino and F. Gullo are with UniCredit, R&D Department, Italy. Email: {ilaria.bordino,francesco.gullo}@unicredit.eu.*
• *G. Legnaro is with Prometeia, Italy. Email: giacomo.legnaro@prometeia.com.*

request, achieving a *larger* availability of *liquidity*, and a *lower risk* of incurring into payment delays or credit losses. The increased cash availability in turn allows the funder to devise proper marketing strategies to attract more customers to try the novel settlement service. Specifically, the funder offers the customers the benefit of a *lower fee* on each incoming deliverable. A further advantage for the customer is the *easier access* to the receivable financing service (in terms of both time and effort), which is enabled by the employment of lighter bureaucracy and risk-assessment rules.

A crucial aspect for the network-based RF service to work properly is to regulate it with to a do-ut-des mechanism, where customers accept to pay selected receivables, for which they act as a debtor, earlier than they would do without the service. Our enrolling customer is warned that the system does not allow non-payments, and that when a receivable for which she plays as debtor is selected for settlement, its amount is automatically transferred to the creditor. However, the customer is informed that she also benefits from early payments, because the system ensures that they increase the chance for a debtor of subsequently acting as a creditor. Moreover, customers may preserve their freedom on how to handle payments, by choosing which requests to submit to the network-based system.

Challenges and contributions. In this work we tackle a real-world problem from a specific application domain, i.e., receivable financing, proposing a *novel* method, based on a *network-perspective* that – to the best of our knowledge – has never been employed for such a problem before. Effectively solving the network-based receivable settlement problem requires non-trivial algorithmic and theoretic work. This paper focuses on the algorithmic challenges that arise amid the design of a such a network-based approach.

We observe that the set of receivables that are available for settlement at any day can be naturally modeled as a *multigraph*, whose nodes correspond to customers, and an arc from u to v represents a receivable having u as a debtor and v as a creditor. Such a multigraph is *arc-weighted*, with weights corresponding to receivable amounts, and *node-attributed*, as every node is assigned its account balance(s), as well as a *floor* and a *cap*, which serve the purpose of limiting the balance(s) of a node to stay within a reasonable range. We formulate network-based receivable settlement as a novel combinatorial-optimization problem on a multigraph of receivables, whose objective is to select a subset of arcs so as to maximize the total amount of the selected arcs, while also satisfying two constraints: (i) the balance of every node meets the corresponding floor-cap constraints, and (ii) every node within the output solution is the creditor of at least one output receivable and the debtor of at least one output receivable. We show that the problem is **NP**-hard and devise both an exact algorithm and effective algorithms to find approximate solutions more efficiently. Our more efficient algorithms exploit the fact that a cycle of the input multigraph is guaranteed to satisfy one of the two problem constraints. For this reason, we formulate a further combinatorial-optimization problem, which identifies a subset of the cycles of the input multigraph, so as to maximize the total amount and keep the floor-cap constraints satisfied. Such a problem is shown to be **NP**-hard too, and our most efficient algorithms find approximate solutions to it by exploiting a knapsack-like characterization, as well as a refining strategy that properly adds paths between the selected cycles. Orthogonally, we focus on a practical issue that may arise while implementing a network-based RF service in real-world systems: to minimize system re-engineering, it might be required for the money transfers underlying the settled receivables to be executed one at a time and without violating the problem constraints, not even temporarily. We devise proper strategies to handle such a real-world issue, by either redefining floor constraints, or taking a subset of the settled receivables and properly ordering them.

To summarize, the main contributions of this work are:

- We devise a novel, network-based approach to receivable financing (Section 2).
- We provide a principled formulation of the network-based receivable settlement strategy in terms of a novel optimization problem, while also showing the **NP**-hardness of the problem (Section 3).
- We define a *branch-and-bound* algorithm to solve the proposed optimization problem exactly (Section 4.1).
- We devise a more efficient algorithm, based on the optimal selection of a subset of cycles (Section 4.2).
- We present a method to improve upon the cycle-selection-based algorithm by suitably adding paths between the selected cycles (Section 4.3). This algorithm and the branch-and-bound one are also combined into a further *hybrid* algorithm (Section 4.4).
- We handle the real-world scenario where temporary constraint violations are not allowed (Section 4.5).
- We present an extensive evaluation on a real dataset of receivables. Results demonstrate the effectiveness of the proposed methods in practice (Section 6).

In Section 5 we report implementation details, while Section 7 offers related work, and Section 8 draws conclusions.

An abridged version of this paper has been presented in [3]. New material includes novel algorithms (Sections 4.3 and 4.5), implementation details (Section 5), an extended empirical evaluation in Section 6 (with a new dataset, and novel algorithms), an expanded literature review (Section 7), and further insights and explanations throughout the paper.

2 Preliminaries

In this work we consider the following scenario. We focus on the customer basis $\mathcal{U}$ of a single funder (we assume the funder has no visibility on the customers of other funders). Customers in $\mathcal{U}$ submit RF requests to the funder. We denote by $\mathcal{R}$ the set of receivables submitted to the funder by its customers. The goal of the funder is to select a subset of $\mathcal{R}$ to be settled, by employing a *network-based* strategy, i.e., making the involved customers pay each other autonomously.

Receivables. Attributes of a receivable $R \in \mathcal{R}$ include:

- $amount(R) \in \mathbb{R}$: amount of the receivable;
- $creditor(R) \in \mathcal{U}$: payee of the receivable;
- $debtor(R) \in \mathcal{U}$: payer of the receivable;
- $indate(R)$: date the receivable entered the system;
- $duedate(R)$: date on which the payment falls due;
- $life(R) \in \mathbb{N}$: the maximum number of days the network-based RF service is allowed to try to settle the receivable.

R is said *active* for $creditor(R)$, and *passive* for $debtor(R)$.

TABLE 1: List of frequent symbols.

symbol	*definition*
$\mathcal{G}=(\mathcal{V},\mathcal{E},w)$	input R-multigraph
$bl_r(u)$	receivable balance of u's account ($u \in \mathcal{V}$)
$bl_a(u)$	actual balance of u's account ($u \in \mathcal{V}$)
$cap(u)$	upper bound on $bl_r(u)$ ($u \in \mathcal{V}$)
$fl(u)$	lower bound on $bl_a(u)$ ($u \in \mathcal{V}$)
$\mathcal{C}$	set of cycles in $\mathcal{G}$
$C \in \mathcal{C}$	a cycle in $\mathcal{C}$
$\mathcal{P}_{ij}$	paths from a node in cycle C_i to a node in cycle C_j
$\mathcal{P}_{max}$	max size of a path set in $\{\mathcal{P}_{ij}\}_{i,j}$
$L;\ L_p$	max length of a cycle in $\mathcal{C}$; max length of a path in $\mathcal{P}_{ij}$
$K;\ K_p$	size of $\mathcal{C}$'s subset; size of $\mathcal{P}_{ij}$'s subset (beam-search algorithms)
$\mathbf{CC}$	weakly connected components of $\mathcal{G}$
H	max size of a $\mathcal{G}$'s conn. component to run the exact algorithm on

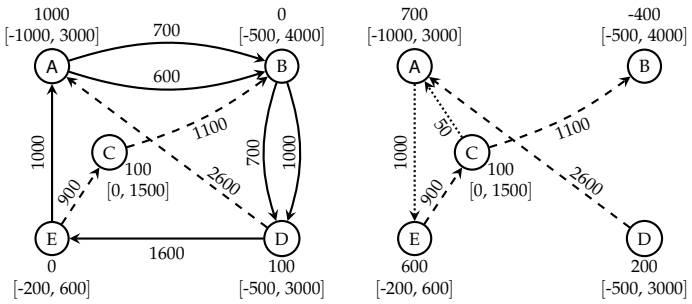


Fig. 2: An instance of MAX-PROFIT BALANCED SETTLEMENT (Problem 1). Nodes are assigned balances (in this example $bl_r = bl_a$), and $floor$-cap ranges (square brackets). Arcs are labeled with the amount of the corresponding receivables. Day 1: the arcs depicted with full lines form the optimal solution. No other arcs can be selected. In fact, adding (E,C) and (C,B) to the solution would violate C's and E's $floor$ constraint, while adding (D,A) would violate D's $floor$ constraint. Day 2: receivables present in the system corresponds to the ones not settled at Day 1, along with the two new ones depicted with dotted lines (i.e., (A,E) and (C,A)). The optimal solution at Day 2 is an empty set. In fact, the only way to have Problem 1's Constraint (2) satisfied would be selecting (A,E), (E,C), and (C,A) altogether. However, this is not possible as it would not comply with E's *cap* constraint.

Customers. An ad-hoc account is assigned from the funder to each of its customers. Such a dedicated account keeps track of the movements underlying the various RF requests of the customer. Customers may also *deposit/withdraw* money on/from their standard account: such deposits/withdrawals correspond to *external* money flows (as they do not come from receivable settlement), which should be properly regulated in order to maintain the overall collaborative equilibrium of the system. Specifically, the main requirement here is that withdrawals should not contribute to increase the operability of the customer within the network-based RF service, whereas the opposite should hold for deposits. To satisfy this requirement, every customer $u \in \mathcal{U}$ is assigned two different balances in her dedicated account. These balances are limited by a *floor* and a *cap* (that are set on a customer basis during sign up). Thus, a customer $u \in \mathcal{U}$ is assigned the following attributes:

- $bl_r(u) \in \mathbb{R}$: *receivable balance* of u's account, i.e., the sum of all receivables u has got paid minus the sum of all receivables u has paid through the RF service over the whole u's time of activity;
- $bl_a(u) \in \mathbb{R}$: *actual balance* of u's account, corresponding to the receivable balance $bl_r(u)$, increased by money from deposit operations and decreased by withdrawals;
- $cap(u) \in \mathbb{R}$: upper bound on the receivable balance of u's account; requiring $bl_r(u) \leq cap(u)$ avoids unbalanced situations where a customer utilizes the service only to get money without paying passive receivables;
- $fl(u) \in \mathbb{R}$: lower bound on the actual balance of u's account; typically, $fl(u) = 0$, but in some cases negative values may be allowed.

Network-based RF in action. A receivable R is submitted by $creditor(R)$ to the netwok-based RF service. While submitting R, the creditor also sets $life(R)$, i.e., the maximum number of days R can stay in the system: if R has not been settled during that period, the creditor gets it back (and she may require to sell it to different financing services).

Once R has been added to the system, $debtor(R)$ is asked to confirm if she agrees with paying R anytime between $indate(R)$ and $duedate(R)$. If $debtor(R)$ gives her confirmation, it means that she accepts to pay R possibly before its $duedate$. This is a crucial aspect in the design of network-based RF. In this regard, a specific mechanism is employed to maintain the desired equilibrium where customers autonomously pay each other as far as possible: the debtor must accept to pay a receivable before its $duedate$ to *gain operability within the service, so as to get her (future) active receivables settled more easily*. Indeed, according to the constraint $bl_r(u) \leq cap(u)$, the more the receivables paid by u through the network-based RF service, the further $bl_r(u)$ remains from $cap(u)$ and the higher the chance for u to have her active receivables paid.

After confirmation by $debtor(R)$, R becomes part of the set $\mathcal{R}$ of receivables that the system will attempt to settle (according to the method(s) presented in Sections 3-4). If the system is not able to settle R during the period $[indate(R), \min\{indate(R) + life(R), duedate(R)\}]$, $creditor(R)$ gets the receivable back. Otherwise, $amount(R)$ is withdrawn from the $debtor(R)$'s account and put to the $creditor(R)$'s account. Without loss of generality, we assume that the settlement fee of receivable R is paid by $creditor(R)$ to the funder of the RF service through a different channel. As better explained in Section 3, a receivable R can be selected for settlement only if it complies with the economic conditions of $debtor(R)$'s (and $creditor(R)$'s) account (and some other global constraints are satisfied, see Problem 1). This prevents the system from being affected by insolvencies.

3 PROBLEM DEFINITION

A network-based RF service requires the design of a proper strategy to select a subset of receivables to be settled. Here we assume receivable settlement works on a daily basis, running *offline at the end of a working day* t, and taking as input receivables that are *valid* at time t, i.e., $\mathcal{R}(t) = \{R \in \mathcal{R} \mid t \in [indate(R),\ \min\{indate(R) + life(R), duedate(R)\}]\}$. This scenario induces a *multigraph*, where arcs correspond to receivables, and nodes to customers. This multigraph, termed ***R-multigraph***, is *directed*, *weighted*, and *node-attributed*:

Definition 1 (R-multigraph). Given a set $\mathcal{R}(t)$ of receivables active at time t, the ***R-multigraph*** induced by $\mathcal{R}(t)$ is a triple $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, where $\mathcal{V}$ is a set of *nodes*, $\mathcal{E}$ is a *multiset* of *ordered* pairs of nodes, i.e., *arcs*, and $w : \mathcal{E} \rightarrow$

$\mathbb{R}^+$ is a function assigning (positive real) weights to arcs. Each arc $(u,v) \in \mathcal{E}$ models the case "u pays v", i.e., it corresponds to a receivable R where $u = debtor(R)$, $v = creditor(R)$, and $w(u,v) = amount(R)$. Each node $v \in \mathcal{V}$ is assigned attributes $bl_r(u)$, $bl_a(u)$, $cap(u)$, and $fl(u)$.

The objective in our network-based settlement is to select a set of receivables, i.e., arcs of the R-multigraph $\mathcal{G}$, so as to *maximize the total amount*. This is a desideratum for both the funder and its customers. A larger amount of settled receivables brings more profit to the founder, and also larger savings for the customers, who – in the absence of this service – would be forced to pay for more expensive (traditional) alternatives. The identified receivables should satisfy two constraints for every customer u spanned by them: (1) the resulting $bl_r(u)$ and $bl_a(u)$ should be within $cap(u)$ and $fl(u)$ (i.e., $bl_r(u) \leq cap(u)$, $bl_a(u) \geq fl(u)$), and (2) u should be the payer of at least one selected receivable and the payee of at least another selected receivable. Constraint (2) arises due to a specific marketing choice, i.e., preventing a customer from only paying receivables in a given day. This is based on the idea that showing clients that any day they pay a receivable, they also receive at least one payment as creditors, may crucially help increase their appreciation and engagement with the service. Moreover, preventing a customer from only paying receivables serves the aim of ensuring the aforementioned *do-ut-des* principle.

The above desiderata are formalized into the following combinatorial-optimization problem, while Figure 2 depicts a (toy) problem instance.

***Problem 1* (MAX-PROFIT BALANCED SETTLEMENT).** Given an R-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, find

$$\mathcal{E}^* = \arg\max_{\hat{\mathcal{E}} \subseteq \mathcal{E}} \sum_{e \in \hat{\mathcal{E}}} w(e) \quad \text{subject to}$$

$$\Big(\sum_{(v,u) \in \hat{\mathcal{E}}} w(v,u) - \sum_{(u,v) \in \hat{\mathcal{E}}} w(u,v)\Big) \in [fl(u) - bl_a(u), cap(u) - bl_r(u)], \quad (1)$$

$$|\{(u,v) \mid (u,v) \in \hat{\mathcal{E}}\}| \geq 1, \text{ and } |\{(v,u) \mid (v,u) \in \hat{\mathcal{E}}\}| \geq 1, \quad (2)$$

$$\forall u \in \mathcal{V}(\hat{\mathcal{E}}) = \{u \in \mathcal{V} \mid (u,v) \in \hat{\mathcal{E}} \vee (v,u) \in \hat{\mathcal{E}}\}.$$

Theorem 1. Problem 1 is **NP**-hard.

Proof: (Sketch) We reduce from SUBSET SUM [11]: given a set S of positive real numbers and a further real number B larger than the minimum number in S, find a subset $S^* \subseteq S$ such that the sum of the numbers in S^* is maximum and no more than B. The idea of the reduction is to show that solving MAX-PROFIT BALANCED SETTLEMENT on a two-node R-multigraph with as many arcs between the two nodes as the number of input numbers in S corresponds to solving the original SUBSET SUM problem instance. A detailed proof is shown in [3]. □

4 ALGORITHMS

4.1 Exact algorithm

Our first proposal to solve the MAX-PROFIT BALANCED SETTLEMENT problem is a *branch-and-bound* exact algorithm, dubbed Settle-BB. Given a R-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, the search space of MAX-PROFIT BALANCED SETTLEMENT corresponds to the set $2^{\mathcal{E}}$ of all (multi)subsets of arcs of $\mathcal{G}$. In the proposed Settle-BB algorithm such a search space is represented as a *binary-tree* $\mathcal{T}$, with $|\mathcal{E}|+1$ levels, where every level (but the root) is logically assigned an arc $e \in \mathcal{E}$. Every level of the tree comes with a decision on whether the corresponding arc is part of the output or not. The assignment of arcs to the levels of $\mathcal{T}$ is decised based on a certain ordering of the arcs (e.g., by non-increasing amount, see Section 5). A path in $\mathcal{T}$, from the root to a leaf, corresponds to an admissible solution $\hat{\mathcal{E}} \in 2^{\mathcal{E}}$ to the problem (setting a decision for all arcs in $\mathcal{E}$). A tree-node other than the root or a leaf ($\mathcal{T}$'s nodes are termed "tree-nodes", to distinguish them from R-multigraph's nodes) represents a set of solutions, corresponding to all possible decisions for the arcs within the path from the root to that non-leaf node.

The search space $\mathcal{T}$ is visited by the proposed Settle-BB according to some strategy, e.g., BFS or DFS (see Section 5). While visiting the various tree-nodes, Settle-BB considers a *lower bound* and an *upper bound* on the set of solutions corresponding to the current tree-node. Such bounds are used in a traditional branch-and-bound fashion to prune the search space. For a detailed description of Settle-BB please see [3], [4].

4.2 Beam-search algorithm

Being MAX-PROFIT BALANCED SETTLEMENT **NP**-hard, the exact Settle-BB algorithm cannot handle large R-multigraphs. We thus design an alternative algorithm that finds approximate solutions and can run on larger instances. To this end, we exploit the idea of enumerating all cycles in the input multigraph, and selecting a subset of them while keeping the constraints satisfied. The intuition behind this strategy is twofold. First, a solution composed of a set of cycles always satisfies the other constraint of the problem, as every node of a cycle has at least one incoming arc and one outgoing arc. Also, cycle enumeration is a well-established problem, for which a variety of algorithms exist. As a trade-off between effectiveness, efficiency and simplicity, in this work we employ a variant of the Johnson's algorithm [10] that works on multigraphs [8].

For a principled cycle selection, we focus on the problem of seeking cycles that satisfy the constraints in Problem 1 and exhibit maximum total amount:

***Problem 2* (OPTIMAL CYCLE SELECTION).** Given an R-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ and a set $\mathcal{C}$ of cycles in $\mathcal{G}$, find a subset $\mathcal{C}^* \subseteq \mathcal{C}$ so that:

$$\mathcal{C}^* = \arg\max_{\hat{\mathcal{C}} \subseteq \mathcal{C}} \textstyle\sum_{e \in \mathcal{E}(\hat{\mathcal{C}})} w(e)$$

subject to $\mathcal{E}(\hat{\mathcal{C}}) = \bigcup_{C \in \hat{\mathcal{C}}} C$ meets Constraint (1) in Problem 1.

Theorem 2. Problem 2 is **NP**-hard.

Proof: (Sketch) We reduce from MAXIMUM INDEPENDENT SET [6], which asks for a maximum-sized subset of vertices in a graph no two of which are adjacent. Given a graph $G = (V, E)$ instance of MAXIMUM INDEPENDENT SET, the idea of the reduction is to construct an instance $\langle \mathcal{G}, \mathcal{C} \rangle$ of OPTIMAL CYCLE SELECTION where there is a cycle for every vertex in V, and those cycles satisfy certain conditions on the weights on their arcs depending on the two corresponding vertices are adjacent in G or not. Ultimately, it can be shown that the optimal cycle set found on the constructed OPTIMAL CYCLE SELECTION instance corresponds

Algorithm 1: Settle-BEAM

Input: R-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, integer K
Output: Multiset $\mathcal{E}^* \subseteq \mathcal{E}$
1: $\mathcal{E}^* \leftarrow \emptyset$
2: $\mathcal{C} \leftarrow$ cycles of $\mathcal{G}$ {[8]}
3: **while** $\mathcal{C} \neq \emptyset$ **do**
4: $\mathcal{C}' \leftarrow K$-sized subset of $\mathcal{C}$ by Greedy MAX COVER {[9]}
5: $\mathcal{C}'_2 \leftarrow \{\{C_i, C_j\} \,|\, C_i, C_j \in \mathcal{C}'\}$
6: **for all** $\{C_i, C_j\} \in \mathcal{C}'_2$ **do**
7: $C_{ij} \leftarrow C_i \cup C_j$
8: process all $C \in \mathcal{C}' \setminus \{C_i, C_j\}$ one by one, by non-increasing $\omega(\cdot)$ score (Eq.(3)); add C to C_{ij} if $C_{ij} \cup C \cup \mathcal{E}^*$ is feasible for Problem 2
9: $\mathcal{E}^* \leftarrow \mathcal{E}^* \cup \arg\max_{C_{ij} \in \mathcal{C}'_2} \sum_{e \in C_{ij}} w(e)$
10: $\mathcal{C} \leftarrow \mathcal{C} \setminus (\mathcal{C}' \cup \{C \in \mathcal{C} \mid C \cap \mathcal{E}^* = C\})$

to the solution to the original MAXIMUM INDEPENDENT SET instance. The detailed reduction can be found in [3]. □

As OPTIMAL CYCLE SELECTION is **NP**-hard, we design an effective approximate solution by combining KNAPSACK-solving methodologies with the principles of *beam search*. We start by observing that OPTIMAL CYCLE SELECTION is an instance of the MULTIDIMENSIONAL SET UNION KNAPSACK problem, which is defined as follows:

Problem 3* (MULTIDIMENSIONAL SET UNION KNAPSACK*). Let $U = \{x_1, \ldots, x_h\}$ be a universe of elements, $\mathcal{S} = \{S_1, \ldots, S_k\}$ be a set of items, where $S_i \subseteq U, \forall i \in [1..k]$, $p : \mathcal{S} \rightarrow \mathbb{R}$ be a profit function for items in $\mathcal{S}$, and $q : U \rightarrow \mathbb{R}^d$ be a d-dimensional cost function for elements in U. For any $\hat{\mathcal{S}} \subseteq \mathcal{S}$ define also: $U(\hat{\mathcal{S}}) = \bigcup_{S \in \hat{\mathcal{S}}} S$, $P(\hat{\mathcal{S}}) = \sum_{S \in \hat{\mathcal{S}}} p(S)$, and $\mathbf{Q}(\hat{\mathcal{S}}) = \sum_{x \in U(\hat{\mathcal{S}})} q(x)$. Given a d-dimensional vector $\mathbf{B} \in \mathbb{R}^d$, MULTIDIMENSIONAL SET UNION KNAPSACK finds $\mathcal{S}^* = \arg\max_{\hat{\mathcal{S}} \subseteq \mathcal{S}} P(\hat{\mathcal{S}})$ s.t. $\mathbf{Q}(\hat{\mathcal{S}}) \leq \mathbf{B}$.

Motivated by this connection with MULTIDIMENSIONAL SET UNION KNAPSACK, we devise an algorithm for OPTIMAL CYCLE SELECTION inspired by Arulselvan's algorithm [2], which achieves a $1 - e^{-1/f_{max}}$ approximation guarantee for MULTIDIMENSIONAL SET UNION KNAPSACK, where f_{max} is the maximum number of items in which an element is present. Arulselvan's algorithm first computes all subsets of 2 items whose weighted union is within the budget B. Next, it expands each subset with items S_i added one by one in decreasing order of an ad-hoc-defined score, as long as the budget constraint B is satisfied. The score employed for selecting an item is directly proportional to its profit, and inversely proportional to its frequency within the entire item set $\mathcal{S}$. The algorithm outputs the augmented subset yielding the highest profit.

Our ultimate Settle-BEAM algorithm (Algorithm 1) adapts Arulselvan's algorithm to our context, introducing the following modifications. (*i*) We extend the algorithm so as to handle a MULTIDIMENSIONAL SET UNION KNAPSACK problem instance derived from the input OPTIMAL CYCLE SELECTION instance as stated in Theorem 4.5 in [3] (trivial extension). (*ii*) We define the score of a cycle $C \in \mathcal{C}$ as:

$$\omega(C) = \frac{\sum_{e \in C} w(e)}{\sum_{e \in C} \frac{w(e)}{f(e)}}, \text{ where } f(e) = |\{C \in \mathcal{C} : e \in C\}|, \quad (3)$$

This score considers the total cycle amount, penalized by a term accounting for the frequency of an arc within the cycle set $\mathcal{C}$. The intuition is that frequent arcs must be penalized as they contribute less to the objective function, which is defined on the *union* of the arcs of all selected cycles (see Problem 2). Finally, (*iii*) we employ beam search to mitigate the burden of pairwise cycle enumeration and augmentation. We first select a subset $\mathcal{C}' \subseteq \mathcal{C}$ of cycles of size $K \leq |\mathcal{C}|$ whose union arc set exhibits the maximum amount: we solve this step by observing that it is basically an instance of (weighted) MAX COVER [9], where arcs correspond to elements and cycles to sets, and consequently adopting the classic greedy $(1 - \frac{1}{e})$-approximation algorithm for MAX COVER. We then use $\mathcal{C}'$ for pairwise cycle computation and augmentation. We repeat the procedure (by selecting a further K-sized subset of $\mathcal{C} \setminus \mathcal{C}'$) until $\mathcal{C}$ has become empty.

The running time of Settle-BEAM is dominated by cycle enumeration (Line 2), as the number of cycles in a (multi)graph can be exponential. This is however not blocking for us: as it is unlikely that the problem constraints are satisfied on long cycles, we employ a simple yet effective workaround of detecting cycles up to a certain size L. This way, the time complexity of Settle-BEAM is $\mathcal{O}(L\,K^2\,|\mathcal{C}|)$. For a more detailed complexity analysis please see [3], [4].

Algorithm 2: Settle-PATH

Input: R-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, two integers K, K_p
Output: Multiset $\mathcal{E}^* \subseteq \mathcal{E}$
1: $\mathcal{C}^* \leftarrow$ cycles of $\mathcal{G}$ selected by Algorithm 1
2: $\mathcal{E}^* \leftarrow \bigcup_{C \in \mathcal{C}^*} C$
3: **for all** $C_i, C_j \in \mathcal{C}^*$ **do**
4: $\mathcal{P}_{ij} \leftarrow$ paths from a node in C_i to a node in C_j {[8]}
5: $\mathcal{P}^*_{ij} \leftarrow$ paths selected by Algorithm 1 on input $\langle \mathcal{G}, K_p \rangle$ and setting $\mathcal{C} := \mathcal{P}_{ij}$ at Line 2
6: $\mathcal{E}^* \leftarrow \mathcal{E}^* \cup \bigcup_{P \in \mathcal{P}^*_{ij}} P$

4.3 Adding paths between cycles

An interesting insight on cycle-based solutions to MAX-PROFIT BALANCED SETTLEMENT is that they can be improved by *looking for paths connecting two selected cycles*. An approach based on this observation is particularly appealing as adding a path between two cycles keeps the cycle constraint of our problem (i.e., Constraint (2) in Problem 1) satisfied, therefore one has only to focus on the floor-cap constraint while checking admissibility of paths.

Within this view, we propose a refinement of the Settle-BEAM algorithm where, for every pair C_i, C_j of selected cycles, we (*i*) compute paths $\mathcal{P}_{ij}$ from a node in C_i to a node in C_j, and (*ii*) take a subset $\mathcal{P}^*_{ij} \subseteq \mathcal{P}_{ij}$ such that the total amount $\sum_{e \in P, P \in \mathcal{P}^*_{ij}} w(e)$ is as large as possible and adding $\mathcal{P}^*_{ij}$ to the initial MAX-PROFIT BALANCED SETTLEMENT solution still meets the cap-floor constraints. We dub this algorithm Settle-PATH and outline it as Algorithm 2. To compute a path set $\mathcal{P}_{ij}$, we employ a simple variant of the (multigraph version of the) well-known Johnson's algorithm [8], where we make the underlying backtracking procedure start from nodes in C_i and yield a solution (i.e., a path) every time it encounters a node in C_j. The selection of $\mathcal{P}^*_{ij} \subseteq \mathcal{P}_{ij}$ can be accomplished by running either a standard greedy method (as in Algorithm 2 in the first version of our work [3]), or Algorithm 1 (with the modification that the set $\mathcal{C}$ at Line 2 corresponds to $\mathcal{P}_{ij}$).

Algorithm 3: Settle-H/Settle-H-PATH

Input: R-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, three integers H, K, K_p
Output: Multiset $\mathcal{E}^* \subseteq \mathcal{E}$

$\mathcal{E}^* \leftarrow \emptyset$, $\mathbf{CC} \leftarrow$ weakly connected components of $\mathcal{G}$
for all $G \in \mathbf{CC}$ s.t. $|arcs(G)| \leq H$ **do**
  $\mathcal{E}^* \leftarrow \mathcal{E}^* \cup$ Settle-BB on input G {Section 4.1}
for all $G \in \mathbf{CC}$ s.t. $|arcs(G)| > H$ **do**
  $\mathcal{E}^* \leftarrow \mathcal{E}^* \cup$ Settle-BEAM or Settle-PATH on input $\langle G, K, K_p \rangle$ {Algorithms 1–2}

The running time of Settle-PATH is expected to be dominated by the various path-enumeration steps. As far as the remaining steps, the worst case corresponds to employing Algorithm 1 for both cycle- and path-selection, which leads to an overall time complexity of $\mathcal{O}(LK^2|\mathcal{C}| + |\mathcal{C}^*|^2 L_p K_p^2 |\mathcal{P}_{max}|)$, where L_p and K_p are two integers playing the same roles as L and K, respectively, in Algorithm 1 when run on path sets, and $\mathcal{P}_{max} = \arg\max_{\mathcal{P} \in \{\mathcal{P}_{ij}\}_{i,j}} |\mathcal{P}|$.

4.4 Hybrid algorithm

The MAX-PROFIT BALANCED SETTLEMENT problem can be solved by running any of the algorithms presented in Sections 4.1–4.3 on each weakly connected component of the input R-multigraph *separately*, and then taking the union of all partial solutions. Based on this straightforward observation, we ultimately propose a *hybrid* algorithm, which runs the exact Settle-BB algorithm on the smaller connected components and the Settle-BEAM or Settle-PATH algorithm on the remaining ones. Our hybrid algorithm is outlined as Algorithm 3: we term it either Settle-H or Settle-H-PATH, depending on whether it is equipped with Settle-BEAM or Settle-PATH, respectively.

4.5 Temporary constraint violation

Given a set $\mathcal{E}^*$ of receivables selected for settlement, the eventual step of our network-based RF service is, for every receivable $R \in \mathcal{E}^*$, to transfer $amount(R)$ from $debtor(R)$'s account to $creditor(R)$'s account. A typical desideratum from big (financial) institutions is that these money transfers happen by exploiting procedures and software solutions already in place within the institution, in order to minimize the effort in system re-engineering. For instance, our partner institution required to interpret money transfers as a sequence of standard bank transfers. To accomplish this, money transfers have to typically be executed one at a time, meaning that *some constraints in Problem 1 might be temporarily violated*. As an example, look at Figure 2 and assume the money transfer corresponding to the receivable from D to E is the first one to be executed: this would lead to the temporary violation of D's floor constraint. In real scenarios constraint violation might not be allowed, not even temporarily. In banks, for instance, a floor-constraint violation corresponds to an overdraft on a bank account, which, even if it lasts a few milliseconds only, would anyway cause that account being charged. Another motivation may be simply that existing systems do not accept any form of temporary violation.

Motivated by the above, here we devise solutions to the temporary-constraint-violation issue. In particular, we focus on floor constraints only, as temporary violation of cap constraints or cycle constraints is not really critical. We propose two solutions: the first one is based on a clever redefinition of floor values, while the second one consists in properly selecting and ordering a subset of the arcs that were originally identified for settlement. The details of both solutions are reported next.

Algorithm 4: Redefine-Floors

Input: R-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, two integers H, K
Output: List $\mathcal{E}^* \subseteq \mathcal{E}$

$\mathcal{E}^* \leftarrow []$
repeat
  $\widehat{\mathcal{E}} \leftarrow$ a solution to Problem 1 on input $\langle \mathcal{G}, H, K \rangle$, with floors set as $fl(u) = ub_{in}(u), \forall u \in \mathcal{V}$ {Equation (5)}
  order $\widehat{\mathcal{E}}$ in some way and append it to $\mathcal{E}^*$
  remove all arcs in $\widehat{\mathcal{E}}$ from $\mathcal{G}$
  for all $u \in \mathcal{V}(\widehat{\mathcal{E}})$ **do**
    $\Delta(u) \leftarrow \sum_{v:(v,u)\in\widehat{\mathcal{E}}} w(v,u) - \sum_{v:(u,v)\in\widehat{\mathcal{E}}} w(u,v)$
    $bl_r(u) \leftarrow bl_r(u) + \Delta(u)$, $bl_a(u) \leftarrow bl_a(u) + \Delta(u)$
until $\mathcal{E} = \emptyset \vee \widehat{\mathcal{E}} = \emptyset$

Redefining floor values. The first proposal to handle temporary constraint violation is based on the following result: properly redefining floor values leads to output solutions to MAX-PROFIT BALANCED SETTLEMENT containing only receivables that do not violate the original floor constraints, *regardless of the ordering with which the corresponding money transfers are executed*. Specifically, given a set $\mathcal{E}^* \subseteq \mathcal{E}$ of arcs in a MAX-PROFIT BALANCED SETTLEMENT solution, for every node $u \in \mathcal{V}(\mathcal{E}^*)$, let $in(u, \mathcal{E}^*) = \sum_{v:(v,u)\in\mathcal{E}^*} w(v,u)$ and $out(u, \mathcal{E}^*) = \sum_{v:(u,v)\in\mathcal{E}^*} w(u,v)$ denote the total amount received and paid by u within $\mathcal{E}^*$, respectively. Let also $ub_{in}(u)$ denote an upper bound on the amount $in(u, \mathcal{E}^*)$ that u can receive in any possible solution $\mathcal{E}^*$. Our first observation is that, by setting $fl(u) = ub_{in}(u), \forall u \in \mathcal{V}(\mathcal{E}^*)$, no customer u will ever pay a total amount larger than her current $bl_a(u)$ balance availability $bl_a(u)$ in any MAX-PROFIT BALANCED SETTLEMENT solution, i.e., it would be guaranteed that, for all solutions $\mathcal{E}^*$ and all $u \in \mathcal{V}(\mathcal{E}^*)$, $bl_a(u) \geq out(u, \mathcal{E}^*)$. In fact, for all solutions $\mathcal{E}^*$ it holds that:

$$\begin{aligned} & in(u,\mathcal{E}^*) - out(u,\mathcal{E}^*) + bl_a(u) \;\geq\; fl(u) = ub_{in}(u) \\ \Leftrightarrow\;\; & bl_a(u) - out(u,\mathcal{E}^*) \;\geq\; ub_{in}(u) - in(u,\mathcal{E}^*) \;\geq\; 0 \\ \Rightarrow\;\; & bl_a(u) \;\geq\; out(u,\mathcal{E}^*). \end{aligned} \tag{4}$$

Now, the key question is how to define $ub_{in}(u)$. A simple way would be setting it to the u's in-degree. Another option consists in defining it based on the cap constraint:

$$\begin{aligned} & cap(u) \;\geq\; in(u,\mathcal{E}^*) - out(u,\mathcal{E}^*) + bl_r(u) \\ \Rightarrow\;\; & cap(u) \;\geq\; in(u,\mathcal{E}^*) - bl_a(u) + bl_r(u) \quad \{\text{Equation (4)}\} \\ \Leftrightarrow\;\; & in(u) \;\leq\; cap(u) + bl_a(u) - bl_r(u). \end{aligned}$$

Combining the two options, we ultimately define

$$ub_{in}(u) = \min\Big\{\textstyle\sum_{v:(v,u)\in\mathcal{E}} w(v,u),\, cap(u) + bl_a(u) - bl_r(u)\Big\}. \tag{5}$$

To summarize, our first strategy to handle temporary constraint violation consists in computing a solution $\mathcal{E}^*$ to MAX-PROFIT BALANCED SETTLEMENT with a floor constraint $fl(u) = ub_{in}(u)$ (Equation (5)), for all u. Based on the above arguments, $\mathcal{E}^*$ guarantees no temporary floor-constraint violation, independently of the execution order of the corresponding money transfers. As a refinement, one can run this method iteratively, i.e., removing the arcs

Algorithm 5: Select-and-Order

Input: R-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, two integers H, K
Output: List $\mathcal{E}^* \subseteq \mathcal{E}$

1: $\mathcal{E}^* \leftarrow [\,]$; $i \leftarrow 1$
2: **repeat**
3: $\quad \mathcal{E}^+ \leftarrow \emptyset$
4: $\quad$ **for all** $u \in \mathcal{V}(\mathcal{E})$ **do**
5: $\quad\quad \mathcal{E}_u \leftarrow$ a subset of $\{(u,v) \mid (u,v) \in \mathcal{E}\}$ s.t. $bl_a(u) - \sum_{e \in \mathcal{E}_u} w(e) \geq fl(u)$, $\forall (u,v) \in \mathcal{E}_u : w(u,v) + bl_r(v) \leq cap(v)$, and $\sum_{e \in \mathcal{E}_u} w(e)$ is maximized
6: $\quad\quad \mathcal{E}^+ \leftarrow \mathcal{E}^+ \cup \mathcal{E}_u$
7: $\quad$ set $ts(e) = i$, $\forall e \in \mathcal{E}^+$
8: $\quad$ order $\mathcal{E}^+$ in some way and append it to $\mathcal{E}^*$
9: $\quad$ **for all** $u \in \mathcal{V}(\mathcal{E}^+)$ **do**
10: $\quad\quad \Delta(u) \leftarrow \sum_{v:(v,u) \in \mathcal{E}^+} w(v,u) - \sum_{v:(u,v) \in \mathcal{E}^+} w(u,v)$
11: $\quad\quad bl_r(u) \leftarrow bl_r(u) + \Delta(u)$, $bl_a(u) \leftarrow bl_a(u) + \Delta(u)$
12: $\quad \mathcal{E} \leftarrow \mathcal{E} \setminus \mathcal{E}^+$; $i \leftarrow i + 1$
13: **until** $\mathcal{E}^+ = \emptyset$
14: $\mathcal{E}^- \leftarrow \emptyset$
15: **repeat**
16: $\quad \mathcal{E}^- \leftarrow \mathcal{E}^* \setminus \{$arcs of the (1,1)-D-core of $\mathcal{E}^*\}$ $\{[7]\}$
17: $\quad$ **for all** $(u,v) \in \mathcal{E}^-$ **do**
18: $\quad\quad \mathcal{E}^-_{uv} \leftarrow \{(u,v)\}$; $X \leftarrow \{(u,v)\}$
19: $\quad\quad$ **while** $X \neq \emptyset$ **do**
20: $\quad\quad\quad \widehat{X} \leftarrow \emptyset$
21: $\quad\quad\quad$ **for all** $(x,y) \in X$ **do**
22: $\quad\quad\quad\quad \widehat{X} \leftarrow \widehat{X} \cup \{(y,z) \in \mathcal{E}^* \mid ts(y,z) = ts(x,y) + 1\}$
23: $\quad\quad\quad X \leftarrow \widehat{X}$; $\mathcal{E}^-_{uv} \leftarrow \mathcal{E}^-_{uv} \cup X$
24: $\quad\quad$ remove all arcs in $\mathcal{E}^-_{uv}$ from $\mathcal{E}^*$
25: **until** $\mathcal{E}^- = \emptyset$

yielded at any iteration and computing a new solution on the remaining graph. The overall solution is the union of all partial solutions, and the arcs are ordered based on the iterations: money transfers corresponding to arcs computed in earlier iterations should be executed before the ones in later iterations (while arc ordering within the same iteration does not matter). This method is outlined as Algorithm 4.

Taking a subset of arcs and properly ordering them. Our second solution to overcome temporary constraint violation consists in *selecting a subset* of the arcs that were originally identified for settlement, and *ordering* them, so as to guarantee that (i) executing the corresponding money transfers according to that ordering is free from constraint violations, and (ii) the arc subset still satisfies MAX-PROFIT BALANCED SETTLEMENT's constraints. While doing so, the objective remains *maximizing the total amount* of the arcs in the subset.

More specifically, the algorithm at hand (Algorithm 5) consists of two phases, i.e., arc selection (Lines 2–13) and arc removal (Lines 14–25). Arc selection aims at identifying, for every node u, the best (in terms of overall amount) subset $\mathcal{E}_u$ of arcs outgoing from u whose total amount complies with u's floor constraint, and such that no arc $(u,v) \in \mathcal{E}_u$ leads to the violation of v's cap constraint (Line 5). This corresponds to (a simple variant of) the SUBSET SUM problem, which is known to be **NP**-hard [11]. For nodes with small-sized out-neighborhood, the problem might even be solved optimally, by brute-force. Alternatively, for nodes with larger out-neighborhoods, an approximated solution can be computed by adopting some existing approximation algorithms for SUBSET SUM. Once such $\mathcal{E}_u$ subsets, for all $u \in \mathcal{V}(\mathcal{E})$, have been identified, all arcs within such $\mathcal{E}_u$ sets are appended to the output $\mathcal{E}^*$ list, with their timestamp set as equal to the current i (Lines 7–8), and the balances of the corresponding nodes are updated (Lines 9–11). As the selection of some arcs during a certain iteration may increase the balance of some nodes, and, thus, enable the selection of further arcs in the next iterations, the arc-selection phase is repeated until no new arc is selected in the current iteration.

The (temporary) list of arcs that has been built during arc selection guarantees that the floor-cap constraints of every selected node are still satisfied. Moreover, if the corresponding money transfers are executed following the ordering of the list, it is also ensured that no floor constraint will ever be violated. On the other hand, the current solution may violate the second constraint of the MAX-PROFIT BALANCED SETTLEMENT problem, i.e., the one that every node within the solution has at least one incoming arc and at least one outgoing arc. Hence, the algorithm has also an arc-removal phase, where those arcs $\mathcal{E}^-$ that are not part of the (1,1)-D-core [7] of the subgraph induced by the current solution are discarded (Line 16), and arcs that have been selected "thanks to" such discarded arcs are (iteratively) removed too (Lines 17–24). Arc removal is iteratively repeated until an empty $\mathcal{E}^-$ set has been built step, as removing arcs in one iteration may cause further constraint violations.

5 IMPLEMENTATION

Here we provide some insights for a correct and/or efficient implementation of the proposed algorithms.

As a first observation, all algorithms can benefit from a preprocessing step, where nodes with no incoming or outgoing arcs are filtered out of the input R-multigraph. In fact, the removal of those nodes is safe as they certainly violate Constraint (2) in our MAX-PROFIT BALANCED SETTLEMENT problem. Such a filtering may be exploited recursively, as the removal of those nodes may cause further nodes to have no incoming/outgoing arcs. Ultimately, the overall procedure corresponds to extracting what in the literature is referred to as the $(1,1)$-*D-core* [7] of the input R-multigraph.

Another general preprocessing consists in preventively filtering nodes based on their values of bl_a and bl_r. Specifically, given an R-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ and a node $u \in \mathcal{V}$, the difference in $bl_r(u)$ induced by any solution to MAX-PROFIT BALANCED SETTLEMENT is lower-bounded and upper-bounded by $LB(u) = \min_{(v,u) \in \mathcal{E}} w(v,u) - \sum_{(u,v) \in \mathcal{E}} w(u,v)$ and $UB(u) = \sum_{(v,u) \in \mathcal{E}} w(v,u) - \min_{(u,v) \in \mathcal{E}} w(u,v)$, respectively. Therefore, a node $u \in \mathcal{V}$ cannot be part of any solution to MAX-PROFIT BALANCED SETTLEMENT (and, thus, it can safely be discarded) if $UB(u) \leq fl(u) - bl_a(u)$, or $LB(u) \geq cap(u) - bl_r(u)$

As far as the search-space exploration in Settle-BB (Section 4.1), we process the arcs in non-increasing amount order. The reason is that arcs with larger amount are likely to contribute more to the optimal solution. In terms of visiting strategy, we experimented with both BFS and DFS, observing no substantial difference between the two.

Finally, in Settle-BEAM (Algorithm 1) we compute the set $\mathcal{C}'_2$ of (admissible) cycle pairs (Line 5) as follows. Let $\mathcal{C}'$ (Line 4) be partitioned into $\mathcal{C}'_{adm}$ (admissible cycles) and $\mathcal{C}'_{\neg adm}$ (non-admissible cycles). For all $C \in \mathcal{C}'$, let $\mathcal{C}'_{adm}(C) = \{C' \in \mathcal{C}'_+ \setminus \{C\} \mid C \cap C' \neq \emptyset\}$ and $\mathcal{C}'_{\neg adm}(C) = \{C' \in \mathcal{C}'_{\neg adm} \setminus \{C\} \mid C \cap C' \neq \emptyset\}$. The set $\mathcal{C}'_2$ is computed as (starting with $\mathcal{C}'_2 = \emptyset$):

TABLE 2: Size of the input R-multigraphs extracted from the real dataset considered in the experiments (algorithm: Settle-H).

dataset	span	size	Worst scenario cap $< \infty$ avg	Worst scenario cap $< \infty$ max	Worst scenario cap $= \infty$ avg	Worst scenario cap $= \infty$ max	Normal scenario cap $< \infty$ avg	Normal scenario cap $< \infty$ max	Normal scenario cap $= \infty$ avg	Normal scenario cap $= \infty$ max	Best scenario cap $< \infty$ avg	Best scenario cap $< \infty$ max	Best scenario cap $= \infty$ avg	Best scenario cap $= \infty$ max
Log_{17}	2017/18	#nodes	50 563	77 323	50 548	77 280	77 134	105 671	77 082	105 627	97 888	121 274	97 789	121 244
		#arcs	51 708	132 944	51 600	132 786	92 560	215 913	92 025	215 078	132 564	278 246	131 325	276 497

for all $C \in \mathcal{C}'_{adm}$ **do**
  for all $C' \in \mathcal{C}'_{adm}(C) \cup \mathcal{C}'_{\neg adm}(C)$ s.t. $C \cup C'$ is admissible **do**
    $\mathcal{C}'_2 \leftarrow \mathcal{C}'_2 \cup \{\{C, C'\}\}$
  for all $C' \in \mathcal{C}'_{adm} \setminus \mathcal{C}'_{adm}(C)$ s.t. $C \cup C'$ is admissible **do**
    $\mathcal{C}'_2 \leftarrow \mathcal{C}'_2 \cup \{\{C, C'\}\}$
for all $C \in \mathcal{C}'_{\neg adm}$ **do**
  for all $C' \in \mathcal{C}'_{adm}(C) \cup \mathcal{C}'_{\neg adm}(C)$ s.t. $C \cup C'$ is admissible **do**
    $\mathcal{C}'_2 \leftarrow \mathcal{C}'_2 \cup \{\{C, C'\}\}$

The rationale is that, for any $C \in \mathcal{C}'_{adm}$, there is no need to consider any $C' \in \mathcal{C}'_{\neg adm} \setminus \mathcal{C}'_{\neg adm}(C)$, as an admissible cycle C cannot be admissible if coupled with a non-admissible cycle that does not overlap with C. Similarly, for any $C \in \mathcal{C}'_{\neg adm}$, there is no need to consider any $C' \in \mathcal{C}'_{adm} \setminus \mathcal{C}'_{adm}(C)$ or any $C' \in \mathcal{C}'_{\neg adm} \setminus \mathcal{C}'_{\neg adm}(C)$, as a non-admissible cycle C cannot become admissible if coupled with any other cycle that does not overlap with C.

6 Experiments

In the first version of our work [3] we experimented with a real dataset provided by UniCredit – a noteworthy European bank – consisting of a random sample of a receivable log, spanning one year in 2015/16. Those experiments assessed the performance of the proposed Settle-H algorithm, with respect to the other simpler methods we devised, i.e., Settle-BB-LB and Settle-BEAM. The results reported in our first paper show that Settle-H outperforms both competitors in terms of total amount of the receivables selected for settlement. We also assessed the scalability of Settle-H on segments of data of increasing size, up to 90 days.

In this work we report *additional* experiments on a *novel*, more recent dataset, i.e., a sample of the receivable log of UniCredit, spanning one year in 2017/2018. The evaluation described in the remainder of this section expands our previous experiments in three directions: (i) first, we compare the proposed Settle-H algorithm to a baseline for network-based receivable financing that we suitably define. Next, we assess the impact of the novel algorithmic contributions hereby introduced, i.e., (ii) improving the solutions based on cycle selection by suitably adding paths between the selected cycles (comparing Settle-H to Settle-PATH), and (iii) avoiding temporary constraints violations (Algorithm 4).

Dataset. We tested the performance of our algorithms on a novel *real dataset* provided by UniCredit. The dataset, which we dub Log_{17}, consists of a random sample of the receivable log of the bank, spanning a time interval of one year, in 2017/2018. The sample roughly includes 5M receivables and 400K customers. Obviously, we worked on an anonymized version of the log: the dataset contains sensitive information – such as the identity of the customers, and other personal data – that cannot be publicly disclosed, and was also made inaccessible to us.

Customers' attributes. Following the settings employed in our previous experiments [3], we set customers' attributes by computing statistics on a training prefix of 3 months,

TABLE 3: Comparing best cycle-based algorithm (Settle-H) vs. RFB baseline in terms of (i) total amount (euros) of settled receivables, (ii) total number of settled receivables (**#R**), and (iii) number of distinct customers involved in at least a daily solution (**#C**).

quarter	scenario	Settle-H amount	Settle-H #R	Settle-H #C	RFB amount	RFB #R	RFB #C
Q_1	W	208 452 169	2045	853	75 871 408	8	6
Q_2		133 998 727	2482	933	28 759 273	4	3
Q_3	cap:	250 919 555	2445	1007	57 472 835	4	3
Q_4	$<\infty$	435 864 820	2678	1019	128 409 243	8	3
Q_1	N	100 052 350	3582	1578	0	0	0
Q_2		210 803 336	5054	1832	0	0	0
Q_3	cap:	181 024 199	5069	2083	0	0	0
Q_4	$<\infty$	405 407 829	5171	1998	68 940 677	4	3
Q_1	B	128 172 168	5369	2347	0	0	0
Q_2		223 972 720	7429	2740	0	0	0
Q_3	cap:	138 227 899	7934	3262	0	0	0
Q_4	$<\infty$	380 133 762	7786	2968	0	0	0
Q_1	W	250 919 555	4474	1504	75 871 408	8	6
Q_2		374 766 629	5194	1525	28 759 273	4	3
Q_3	cap:	300 021 017	5321	1603	57 472 835	4	3
Q_4	∞	516 434 249	6896	1645	128 409 243	8	3
Q_1	N	395 904 601	9373	3012	0	0	0
Q_2		555 117 105	11 712	3270	0	0	0
Q_3	cap:	471 313 548	14 342	3888	0	0	0
Q_4	∞	679 395 059	15 319	3581	189 533 414	26	15
Q_1	B	552 458 593	14 314	4566	0	0	0
Q_2		699 245 106	19 557	5519	0	0	0
Q_3	cap:	544 101 823	22 306	6078	0	0	0
Q_4	∞	907 115 932	23 606	5881	120 592 737	22	15

The initial actual balance $bl_a(u)$ of a customer u was set equal to the average absolute daily difference between the total amount of her passive receivables and the total amount of her active receivables, computed over the days when such a difference yielded a negative value. The rationale is that in those days the customer would have needed further liquidity in addition to that provided by incoming receivables, to finalize the payment of the receivables where she acted as the debtor. Based on the assumption that real customers would try a new service with a limited initial cash deposit, we also imposed an initial upper bound of $50K$ euros on the actual balance of customers.

We set the cap of a customer to be either (i) *finite*, and, specifically, equal to her average daily incoming amount in the training data (using the average cap of all customers as the default value for customers with no incoming payments in the training interval), or (ii) $cap = \infty$: here, accounts were allowed to grow arbitrarily.

Finally, we set $fl(u) = 0$, for each customer u. The heuristics we use are the result of several discussions with marketing experts of our partner institution.

Simulation. We defined 6 simulation settings:

1) *Finite CAP.* Let $fcap(u)$ be the finite value of the cap of a customer u computed as above. We considered 3 scenarios:
   - *Worst:* $life(R) = 5, \forall R \in \mathcal{R}; cap(u) = fcap(u), \forall u \in \mathcal{U}$;
   - *Normal:* $life(R) = 10, \forall R \in \mathcal{R}; cap(u) = 2fcap(u), \forall u \in \mathcal{U}$;
   - *Best:* $life(R) = 15, \forall R \in \mathcal{R}; cap(u) = 3fcap(u), \forall u \in \mathcal{U}$;
2) *CAP* $= \infty$. We considered *Worst*, *Normal*, and *Best* scenarios here too, with $life$ values equal to the corresponding finite-CAP cases, but we set $cap(u) = \infty$, $\forall u \in \mathcal{U}$.

Such settings identify different sets of valid receivables for a day, and yield different multigraphs. Table 2 reports on the sizes of the multigraphs extracted from the Log_{17} dataset.

TABLE 4: Evaluating path selection: comparing the best cycle-based algorithm, Settle-H, with the two path-based algorithms.

cap	sce-nario	quar-ter	Settle-H time (s)	Settle-H amount	Settle-H-PATH-G time (s)	Settle-H-PATH-G amount	Settle-H-PATH-G %gain vs Settle-H	Settle-H-PATH-S time (s)	Settle-H-PATH-S amount	Settle-H-PATH-S %gain vs Settle-H-PATH-G	Settle-H-PATH-S %gain vs Settle-H
$<\infty$	W	Q_1	4	208 452 169	4	206 122 369	−1	54	208 658 217	1	0.1
		Q_2	6	133 998 727	6	142 880 339	7	38	139 157 842	−3	3.9
		Q_3	5	250 919 555	4	251 236 654	0	18	251 689 885	0	0.3
		Q_4	6	435 864 820	6	436 866 648	0	54	437 305 782	0	0.3
	N	Q_1	8	100 052 350	8	171 073 852	71	375	130 166 286	−24	30.1
		Q_2	95	210 803 336	47	224 572 822	7	525	222 848 364	−1	5.7
		Q_3	13	181 024 199	14	187 757 266	4	429	215 370 572	15	19.0
		Q_4	23	405 407 829	38	425 474 537	5	448	413 760 885	−3	2.1
	B	Q_1	50	128 172 168	47	139 304 792	9	576	140 389 844	1	9.5
		Q_2	215	223 972 720	185	238 760 149	7	721	245 781 771	3	9.7
		Q_3	66	138 227 899	34	154 369 240	12	677	158 211 843	2	14.5
		Q_4	192	380 133 762	109	395 718 595	4	1150	392 670 886	−1	3.3
∞	W	Q_1	20	374 766 629	7	401 557 792	7	256	407 383 061	1	8.7
		Q_2	169	226 076 200	122	250 268 430	11	330	264 854 214	6	17.2
		Q_3	17	300 021 017	17	317 522 710	6	300	316 476 954	0	5.5
		Q_4	98	516 434 249	101	540 613 736	5	214	543 879 376	1	5.3
	N	Q_1	246	395 904 601	162	434 561 172	10	1761	458 001 083	5	15.7
		Qs_2	968	555 117 105	739	622 621 171	12	2023	624 403 545	0	12.5
		Q_3	573	471 313 548	505	516 884 575	10	1766	604 594 618	17	28.3
		Q_4	884	679 395 059	694	713 902 252	5	2279	727 907 790	2	7.1
	B	Q_1	804	552 458 593	634	602 544 685	9	3004	538 053 935	−11	−2.6
		Q_2	1604	699 245 106	1170	819 639 994	17	4822	727 206 932	−11	4
		Q_3	971	544 101 823	814	647 240 125	19	3430	685 719 252	6	26
		Q_4	1609	907 115 932	943	970 525 923	7	3360	963 608 594	−1	6.2

Assessment criteria. We measure the performance of the algorithms in terms of the total amount of the receivables selected for settlement. This metric provides direct evidence of the benefits for both the funder and the customers: the greater the amount, the less liquidity the funder has to anticipate, and the smaller the fees for customers.

Parameters. Unless otherwise specified, all experiments refer to $L = 15$ (maximum length of a cycle), $L_p = 15$ (maximum length of a path between cycles), $H = 20$ (size of a connected component to be handled with the exact Settle-BB algorithm), and $K = K_p = 1\,000$ (size of the subset of cycles to be used in every iteration of the Settle-BEAM algorithm and the Settle-PATH algorithm, respectively). These values were chosen *experimentally*, i.e., by verifying the limits that our implementation could handle, with a good tradeoff between effectiveness and efficiency.

Testing environment. All algorithms were implemented in Scala (v. 2.12). Experiments were run on an i9 Intel 7900x 3.3GHz, 128GB RAM machine.

6.1 Results

Comparison against a baseline. Whilst we are not aware of any external method, as this is (to the best of our knowledge) the first attempt to exploit the receivable network to optimize RF, we define a possible baseline for the proposed algorithm(s) as follows. We start from a solution $\mathcal{E}^* = \mathcal{E}$ composed of all the arcs of the input multigraph, and, as long as $\mathcal{E}^*$ violates some Problem 1's constraints, we iteratively: (i) remove arcs from $\mathcal{E}^*$ in non-decreasing amount order, until Constraint (1) in Problem 1 is satisfied; (ii) extract the $(1,1)$-D-core from the subgraph induced by $\mathcal{E}^*$ (to satisfy Constraint (2) in Problem 1). In Table 3 we compare this baseline, dubbed RFB, and Settle-H. We split the Log_{17} dataset into 3-month periods, to have an understanding of the performance on a quarterly basis, and speed up the evaluation by parallelizing on different quarters. For each quarter, parameter-configuration, and algorithm, we report: total amount (euros) of the settled receivables, number of settled receivables (#R), and number of distinct customers involved in a daily solution (#C). The main observation here is that the baseline achieves a consistent loss (70%–100%) against Settle-H in all quarters and scenarios: this confirms the strength of the proposed algorithm.

TABLE 5: Evaluating method for avoiding temporary constraint violation (Algorithm: Settle-H; $cap < \infty$)

quarter	scenario	amount
Q_1	W	7 836 382
Q_2		10 821 713
Q_3		11 026 866
Q_4		18 073 953
Q_1	N	7 836 382
Q_2.		10 821 713
Q_3		11 026 866
Q_4		18 073 953
Q_1	B	7 836 382
Q_2.		10 821 713
Q_3		11 026 866
Q_4		18 073 953

Evaluating path selection. A further experiment we carried out was on the impact of enriching a cycle-based solution by adding paths between cycles. We picked Settle-H, which the previous assessment proved to be our best cycle-selection-based algorithm, and compared it to its path-selection version Settle-PATH. Specifically, we consider the two versions of Settle-PATH mentioned in Section 4.3: Settle-H-PATH-G, which performs a simple greedy path selection, and, Settle-H-PATH-S, which employs the more refined selection method in Algorithm 1. Both variants are based on the outline in Algorithm 2. The results of this experiment are in Table 4.

The main finding here is that our path-based algorithms outperform the best no-path method in all configurations (but one, where they however exhibit small losses, i.e., 1% and 2.6%). Settle-H-PATH-G achieves an average gain over Settle-H of, respectively, 1.5%, 21.8% and 8% in the finite-cap worst, normal, and best scenarios, and 7.3%, 9.3%, and 13% in the infinite-cap worst, normal, and best cases. The average gain of Settle-H-PATH-S over Settle-H is, respectively, 1.2%, 14.2%, and 9.2% in the finite-cap worst, normal, and best scenarios, and 9.2%, 15.9%, and 8.4% in the infinite-cap worst, normal, and best cases. *This attests the relevance of the idea of adding paths to the cycle-based solutions*.

The two path-selection strategies are comparable: Settle-

H-PATH-S wins in 12 configurations, with 5% avg gain, while Settle-H-PATH-G wins in 8 configurations, with 7% avg gain.

As for running time, Settle-H-PATH-G is comparable to Settle-H, and actually faster in 75% of cases. The motivation may be that adding paths to a daily solution causes the algorithm to work on smaller graphs in the next days. As expected, Settle-H-PATH-S is instead consistently (about one order of magnitude) slower than Settle-H-PATH-G and Settle-H, due to its more sophisticated path-selection strategy.

Avoiding temporary constraint violation. We tested the performance of Algorithm 4 for avoiding temporary constraint violation, employing the Settle-H algorithm and considering the finite-cap scenario. The results are reported in Table 5. We observe that, although the settled amount is expectedly less than its counterpart where temporary constraint violation is not addressed, this amount remains reasonably large, i.e., in the order of 1M/10M euros.

7 Related Work

The MAX-PROFIT BALANCED SETTLEMENT problem that we study in this work is a novel contribution of ours [3]. We have introduced network-based receivable settlement in the first version of our work. In the present version, we offer further algorithmic contributions, such as improving cycle-based solutions via proper selection of paths among identified cycles, and methods to adapt the original algorithm to real-world scenarios where temporary constraint violations is not permitted. To the best of our knowledge, *no previous work has ever adopted a similar, network-based formulation, neither for receivable financing, nor for other applications*.

There are some problems falling into the same broad application domain, while still being clearly different from ours. The main goal of these problems is to predict – based on historical data – whether a receivable will be paid, the date of the payment, and late-payment amounts. In this regard, Zeng *et al*. [14] devise (supervised) machine-learning models for invoice-payment outcomes, enabling customized actions tailored for invoices or customers. Kim *et al.* [12] focus on debt collection via call centers, proposing machine-learning models for late-payment prediction and customer-scoring rules to assess the payment likelihood and the amount of late payments. Tater *et al.* [13] propose ensemble methods to predict the status of an invoice being affected by other invoices that are simultaneously being processed. Cheon and Shi [5] devise a customer-attribute-based neural-network architecture for predicting the customers who will pay their (outstanding) invoices with high probability. Appel *et al.* [1] present a prototype developed for a multinational bank, aimed to support invoice-payment prediction.

It is apparent that all those works do not share any similarity with our network-based formulation of receivable financing: our goal is *to select a receivables according to a (novel) combinatorial-optimization problem* that enables money circulation among customers, while the above works *predict future outcomes on the payment* of receivables. Such prediction problems cannot even be somehow auxiliary for our setting. In fact, as discussed in Section 2, our network-based receivable-financing service *does not allow non-payments by design*. Thus, in our context, asking whether payments will be accomplished or not is a meaningless question.

8 Conclusion

We have presented network-based receivable financing. We extend [3], where we described a novel optimization problem on a multigraph of receivables, an exact algorithm, and more efficient, cycle-selection-based algorithms. Here, we present further algorithms, and adaptations to real-world scenarios where temporary constraint violations are not allowed. Experiments on real data attest the performance of our algorithms. In the future we plan to consider *dynamic* aspects, and other applications, e.g., receivable trading.

References

[1] A. P. Appel, V. Oliveira, B. Lima, G. L. Malfatti, V. F. de Santana, and R. de Paula. Optimize cash collection: Use machine learning to predicting invoice payment. *CoRR*, abs/1912.10828, 2019.

[2] A. Arulselvan. A note on the set union knapsack problem. *Discr. Appl. Math.*, 169(Supplement C):214–218, 2014.

[3] I. Bordino and F. Gullo. Network-based receivable financing. In *CIKM*, pages 2137–2145, 2018.

[4] I. Bordino, F. Gullo, and G. Legnaro. Advancing receivable financing via a network-based approach. *CoRR*, abs/2006.13738, 2020.

[5] M. L. F. Cheong and W. SHI. Customer level predictive modeling for accounts receivable to reduce intervention actions. In *ICDATA*, pages 23–29, 2018.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[7] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis. D-cores: Measuring collaboration of directed graphs based on degeneracy. In *IEEE ICDM*, pages 201–210, 2011.

[8] K. A. Hawick and H. James. Enumerating circuits and loops in graphs with self-arcs and multiple-arcs. In *FCS*, pages 14–20, 2008.

[9] D. S. Hochbaum. Approximation algorithms for NP-hard problems. chapter Approximating Covering and Packing Problems: Set Cover, Vertex Cover, Independent Set, and Related Problems, pages 94–143. PWS Publishing Co., 1997.

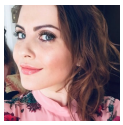
[10] D. B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM J. Comput.*, 4(1):77–84, 1975.

[11] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.

[12] J. Kim and P. Kang. Late payment prediction models for fair allocation of customer contact lists to call center agents. *Decision Support Systems*, 85:84–101, 2016.

[13] T. Tater, S. Dechu, S. Mani, and C. Maurya. Prediction of invoice payment status in account payable business process. In *Service-Oriented Computing*, pages 165–180, 2018.

[14] S. Zeng, P. Melville, C. A. Lang, I. M. Boier-Martin, and C. Murphy. Using predictive analysis to improve invoice-to-cash collection. In *KDD*, pages 1043–1050, 2008.



Ilaria Bordino is a researcher at UniCredit, R&D Department. She received her PhD in 2010, from Sapienza University of Rome and Pompeu Fabra University of Barcelona. Prior to joining UniCredit, she was a research scientist at Yahoo Labs, Barcelona. She has organized the MIDAS (MIning DAta for financial applicationS) Workshop @ECML-PKDD['16-'20].

Francesco Gullo is a researcher at UniCredit, R&D department. He received his PhD from the University of Calabria, in 2010. Before joining UniCredit, he spent 4 years in Yahoo Labs, Spain. He served as a workshop co-chair of ICDM'16, and organized several workshops/symposia (MIDAS @ECML-PKDD['16-'20]; MultiClust @SDM'14, KDD'13; 3Clust @PAKDD'12).

Giacomo Legnaro got his M.S. in Data Science from Sapienza University, Rome, in 2018. His master thesis was entitled *Efficient and effective streaming algorithms for network-based invoice factoring*. After his graduation, he joined Prometeia as a data scientist.