Network-based Receivable Financing

Ilaria Bordino Francesco Gullo
UniCredit, R&D Department — Rome, Italy {ilaria.bordino, francesco.gullo}@unicredit.eu

ABSTRACT

Receivable financing – the process whereby cash is advanced to firms against receivables their customers have yet to pay – is a well-established funding source for businesses. In this paper we present a novel, collaborative approach to receivable financing: unlike existing centralized approaches where the financing company handles each request individually, our approach employs a network perspective where money flow is triggered among customers themselves. The main contribution of this work is to provide a principled formulation of the network-based receivable-settlement strategy, and show how to effectively achieve all algorithmic challenges posed by the design of this strategy. Extensive experiments on real receivable data attest the effectiveness of the proposed methods.

ACM Reference Format:
Ilaria Bordino Francesco Gullo. 2018. Network-based Receivable Financing. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3269206.3272017

1 INTRODUCTION

A *receivable* is a debt owed to a company by its customers for goods or services that have been delivered or used but not yet paid for.

Receivable Financing (RF) is a service for creditors to fund cash flow by selling accounts receivables to a *funder* or *financing company*. The funder anticipates (a proportion of) the receivable amount to the creditor, deducting a percentage as a fee for the service.

Receivable financing mainly exists to shorten the waiting times of receivable payment. These typically range from 30 to 120 days, which means that businesses face a nervous wait as they cannot effectively plan ahead without knowing when their next payment is coming in. Cashing anticipated payment for a receivable gives a business instant access to a lump sum of capital, which significantly eases the cash-flow issues associated with receivables. Another pro is that funders typically manage credit control too, meaning that creditors no longer need to chase up debtors for receivable payment.

RF has traditionally been provided by banks and financial institutions. The last few years have however witnessed the emergence of RF digital platforms, e.g., companies such as the US *BlueVine*, *Fundbox*, and *C2FO*, or the British *MarketInvoice*. While digital funders are rapidly growing, they are still far from saturating the huge potential of the commercial credit market.

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3272017

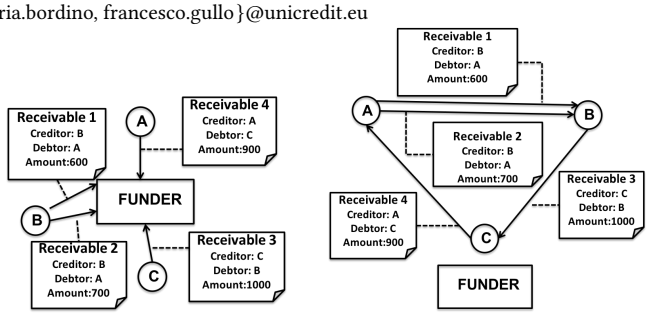

Figure 1: Client/server (left) vs. network-based (right) receivable financing. In the former scenario the funder handles each request by itself, which means anticipating a total amount of 3200. The network-based scenario enables customers to pay each other. Assuming account balances for A, B, C of 0, 300, 0, respectively, Receivables 2, 3, 4 are settled without involving the funder, which has thus to anticipate the amount of Receivable 1 only (600). This means more liquidity and reduced risk for the funder, and a larger saving for the customers (as network-based receivable financing requires smaller fees).

A novel, network-based perspective. The approach used by existing funders is *client-server*: the funder acts as a centralized server that handles each request for a receivable to be funded individually.

A major limitation of the centralized strategy lies in its incapability of dealing with simultaneous requests from customers of the same financing service. In other words, client-server RF completely disregards the fact that receivables for which the financing company is requested constitute a *network where the same customer may act as a creditor or a debtor* of different receivables. In this paper we propose a novel approach to receivable financing where this network perspective is profitably exploited *to trigger a money flow among customers themselves*. In this setting the funder is able to identify a proper subset of receivables such that the involved customers are autonomously able to pay each other, thus activating a *network-based settlement of receivables* that brings substantial benefits to both the funder and the customers (Figure 1).

The pros of network-based RF for the funder are availability of *more liquidity* (one of the main sources of income for financing companies), as it no longer has to provide cash anticipation for each request, and *reduced risk of exposure* to payment delays or credit losses. Such benefits in turn allow the funder to request *smaller fees* for the receivables that are handled through the network-based strategy: this way the network-based approach turns out to be appealing to the customers too. A further advantage for customers is the reduced time and effort in establishing the service, due to less bureaucracy and lighter risk-assessment procedures by the funder.

A crucial aspect is to make customers aware that an implicit *do-ut-des* principle regulates the approach, in which customers accept to pay receivables that are selected by the system, and for which they play as debtors, *possibly earlier than they would otherwise do*. Customers are free to deny their consensus to earlier payments. However, they are discouraged to opt for this choice: as explained in more detail in Section 2, our network-based RF approach implements an ad-hoc mechanism to guarantee that in-advance payment facilitates the arrival of their turn to act as creditors.

Contributions and roadmap. In this paper we present a novel, network-based approach to receivable financing. To the best of our knowledge, this network perspective has never been employed so far in RF services. The main focus of this work is on the algorithmic challenges posed by the design of this novel receivable-settlement methodology. We believe our work is a well-suited example of how a real-world problem from a specific application domain (i.e., finance) requires non-trivial algorithmic and theoretical effort to be effectively solved in practice.

Our main contributions can be summarized as follows:

- 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 derive interesting theoretical conditions to bound the objective-function value of a set of solutions to the problem, and exploit them to design an exact *branch-and-bound* algorithm (Section 4.1).
- We devise a more efficient algorithm, based on a further optimization problem, and its characterization in terms of **NP**-hardness and connection with Knapsack-like problems (Section 4.2).
- As our ultimate proposal to handle real-world problem instances, we design a *hybrid* algorithm that profitably combines the principles of both the exact algorithm and the faster one (Section 4.3).
- We present an extensive evaluation on a real dataset of receivables. Results demonstrate the effectiveness of the proposed methods in practice (Section 5).

Section 6 overviews related work, and Section 7 draws conclusions.

2 PRELIMINARIES

We consider a scenario where a funder has a set of customers who submit receivables requesting for cash anticipation. Our funder adopts a *network-based* approach for receivable settlement: i.e, the funder does not handle each request individually, but attempts (on a daily basis) to automatically identify a subset of receivables such that the involved customers can autonomously pay each other.[1]

Let $\mathcal{U}$ be the set of customers and $\mathcal{R}$ the set of receivables that have been submitted by the customers.

Receivables. A receivable $R \in \mathcal{R}$ has the following attributes:

- $amount(R) \in \mathbb{R}$: amount of the receivable;
- $creditor(R) \in \mathcal{U}$: customer being the payee of the receivable;
- $debtor(R) \in \mathcal{U}$: customer being the payer of the receivable;
- $insertdate(R)$: date the receivable was added to 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.

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

Customers. Customers are assigned a dedicated account by the funder, which is used to pay passive receivables or get paid for active receivables. Moreover, customers may perform *deposits*/*withdrawals* on/from their account. Such operations trigger *external* money flows, which do not derive from receivable settlement. The desideratum to keep the system in a collaborative equilibrium is that, if a customer withdraws money from her account, this operation should not increase the customer's marginal availability to cash in more receivables through the RF service. Conversely, if a customer deposits money on her account, this operation should obviously increase her ability to pay more receivables. To take into account these principles, we keep track of two different balances in the account of customer u, and require such balances to be limited by a *floor* and a *cap* (which are set on a customer basis during sign up). As a result, every customer $u \in \mathcal{U}$ is assigned the 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 lifetime;
- $bl_a(u) \in \mathbb{R}$: *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 $bl_r(u) \leq cap(u)$ at any time 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 are allowed, meaning that some overdraft is tolerated.

Network-based RF in action. The execution flow of our service is as follows. The creditor submits a receivable R to the service, setting $life(R)$, i.e., the number of days the receivable can last in the system. If that period expires with no settlement, the creditor gets the receivables back, possibly to resort to other financing services.

After insertion, the system asks $debtor(R)$ for confirmation, i.e., the consensus to pay the receivable anytime between $insertdate(R)$ and $duedate(R)$, through the money in her account. A specific mechanism is employed to maintain the desired equilibrium where customers autonomously pay each other as far as possible: the debtor is encouraged to accept paying 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. A customer can deny confirmation for a receivable if her current economic situation does not comply with in-advance payments.

Once $debtor(R)$ has given her consensus, R is added to the set $\mathcal{R}$ of current receivables. The system attempts to settle R during the period $[insertdate(R), \min\{insertdate(R) + life(R), duedate(R)\}]$, according to a proper strategy (see Sections 3-4). If no settlement happens, the receivable is returned to the creditor; otherwise, $amount(R)$ is transferred from $debtor(R)$'s account to $creditor(R)$'s account.[2]

[1] We assume a *single-funder scenario*: receivables and customers come from the same funder. Funders have no visibility on receivables and customers of other companies.

[2] For the sake of simplicity and without any loss of generality, we assume that the fee for the settlement of receivable R (corresponding to a small percentage of $amount(R)$) is paid by $creditor(R)$ to the funder of the RF service through a different channel.

3 PROBLEM DEFINITION

The key problem in the design of a network-based RF service is the definition of an effective strategy to properly select a subset of receivables to be settled. In our context we assume that receivable settlement works on a daily basis. It runs *offline at the end of any working day* t, taking as input the set of receivables that are *valid* at time t, i.e., $\mathcal{R}(t) = \{R \in \mathcal{R} \mid t \in [insertdate(R), \min\{insertdate(R) + life(R), duedate(R)\}]\}$. Such receivables describe a *multigraph*, where arcs correspond to receivables, and nodes correspond to the customers spanned by receivables. The multigraph at hand, termed *S-multigraph*, is *directed*, *weighted*, and *node-attributed*:

Definition 3.1 (S-multigraph). Given a set $\mathcal{R}(t)$ of receivables active at time t, the *S-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 $w : \mathcal{E} \to \mathbb{R}^+$ 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 $R \in \mathcal{R}(t)$ 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 S-multigraph $\mathcal{G}$, so as to *maximize the total amount* of the selected receivables. This objective is desirable from the point of view of both the funder and its customers. Indeed, the larger the amount of settled receivables, the larger the profit for the funder. Similarly, a larger receivable amount settled through the network-based RF service leads to larger savings for the customers, who would otherwise resort to more expensive services, such as traditional RF. The selected receivables should meet the following constraints for every customer u spanned by them: (1) the resulting $bl_r(u)$ and $bl_a(u)$ should remain consistent with $cap(u)$ 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) is motivated by the marketing choice of avoiding that a customer is only selected to be payer of receivables on a day. It is believed that showing the clients that any day they are selected to pay a receivable, they also cash at least another receivable as creditors, is essential to consolidate their satisfaction and engagement with the service. At the same time, disallowing a customer to be a payee-only serves the purpose of further guaranteeing the aforementioned *do-ut-des* principle.

The above principles are formalized into the following optimization problem, while Figure 2 depicts a (simple) problem instance.

Problem 1 (Max-profit Balanced Settlement). *Given an S-multigraph* $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$*, find a multisubset* $\mathcal{E}^*$ *of arcs so that*

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

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

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

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

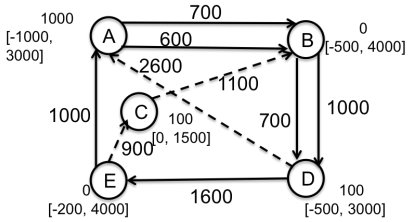

Figure 2: An instance of the Max-profit Balanced Settlement problem. Nodes are labeled with their balances (in this example $bl_r = bl_a$), and fl-cap ranges (square brackets). Arcs are assigned the amount of the corresponding receivables. The arcs depicted with full lines form the optimal solution.

Theorem 3.2. *Problem 1 is* NP*-hard.*

Proof. We reduce from the well-known NP-hard Subset Sum problem [11]: given a set S of positive real numbers and a further real number $B > \min\{x \mid x \in S\}$, find a subset $S^* \subseteq S$ such that the sum of the numbers in S^* is maximum and no more than B. Given an instance $\langle S = \{x_1, \ldots, x_k\}, B\rangle$ of Subset Sum, we construct a Max-profit Balanced Settlement instance composed of a multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ having two nodes, i.e., u and v, one arc from v to u with weight equal to some positive real number $\epsilon < \min\{x \mid x \in S\}$, and as many additional arcs from u to v as the numbers in S, with the weight on each arc e_i from u to v being equal to the corresponding number $x_i \in S$. Moreover, we let u and v have the following attributes: $bl_r(u) = bl_a(u) = 0$, $cap(u) = \epsilon$, $fl(u) = -\sum_{(u,v) \in \mathcal{E}} w(u,v)$, $bl_r(v) = bl_a(v) = \epsilon$, $cap(v) = B$, and $fl(v) = -\epsilon$. The optimal solution $\mathcal{E}^*$ for the Max-profit Balanced Settlement instance $\mathcal{G}$ possesses the following features:

- $\mathcal{E}^* \neq \emptyset$, as there exists at least one non-empty feasible solution whose objective function value is larger than the empty solution, e.g., the solution $\{e_{min}, (v,u)\}$, where $e_{min} = \arg\min_{i \in [1..k]} w(e_i)$ (as $w(v,u) = \epsilon < \min_{i \in [1..k]} w(e_i)$ by construction).
- Arc (v,u) will necessary be part of $\mathcal{E}^*$, otherwise Constraint (2) in Problem 1 would be violated.
- Apart from (v,u), $\mathcal{E}^*$ will contain all those arcs (u,v) that (*i*) fulfill Constraint (1) in Problem 1, and (*ii*) the sum of their weights is maximized. The constraints to be satisfied on node u are:
  $w(v,u) - \sum_{(u,v) \in \mathcal{E}^*} w(u,v) \in [fl(u) - bl_a(u), cap(u) - bl_r(u)]$,
  that is $\epsilon - \sum_{(u,v) \in \mathcal{E}^*} w(u,v) \in \left[-\sum_{(u,v) \in \mathcal{E}} w(u,v), \epsilon\right]$, which is always satisfied, as $\epsilon \in (0, \min_{(u,v) \in \mathcal{E}} w(u,v))$.
  The constraints to be satisfied on node v are instead:
  $\sum_{(u,v) \in \mathcal{E}^*} w(u,v) - w(v,u) \in [fl(v) - bl_a(v), cap(v) - bl_r(v)]$,
  that is $\sum_{(u,v) \in \mathcal{E}^*} w(u,v) - \epsilon \in [-2\epsilon, B - \epsilon]$. The constraint $\sum_{(u,v) \in \mathcal{E}^*} w(u,v) - \epsilon \geq -2\epsilon$ is always satisfied, as all $w(u,v)$ and ϵ are ≥ 0. The constraint $\sum_{(u,v) \in \mathcal{E}^*} w(u,v) - \epsilon \leq B - \epsilon$ corresponds to $\sum_{(u,v) \in \mathcal{E}^*} w(u,v) \leq B$, i.e., the Subset Sum constraint.

As a result, the optimal $\mathcal{E}^*$ of the constructed Max-profit Balanced Settlement instance contains arcs whose sum of weights is maximum and $\leq B$, which corresponds to the optimal solution to the original Subset Sum instance. The theorem follows. □

4 ALGORITHMS

4.1 Exact algorithm

The first proposed algorithm for Max-profit Balanced Settlement is a *branch-and-bound* exact algorithm, dubbed Settlement-bb.

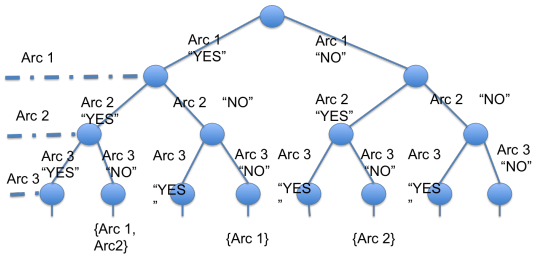

Figure 3: Tree-like representation of the Max-profit Balanced Settlement search space considered in the Settlement-bb algorithm.

Algorithm 1: Settlement-bb

Input: An S-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$
Output: A multiset $\mathcal{E}^* \subseteq \mathcal{E}$

1: $\mathcal{T}$:= tree-like representation of $2^{\mathcal{E}}$
2: $\mathcal{X} \leftarrow \{\text{root of } \mathcal{T}\}$, $LB_{max} \leftarrow 0$
3: **while** $\mathcal{X}$ contains some non-leaf tree-nodes **do**
4: $X \leftarrow$ extract (and remove) a non-leaf tree-node from $\mathcal{X}$
5: $UB_X \leftarrow$ upper bound on the solutions spanned by X {Alg. 3}
6: **if** $UB_X \geq LB_{max}$ **then**
7: $LB_X \leftarrow$ lower bound on the solutions spanned by X {Alg. 2}
8: **if** $LB_X = UB_X$ **then** $\mathcal{E}^* \leftarrow arcs(X)$ and stop the algorithm
9: $LB_{max} \leftarrow \max\{LB_{max}, LB_X\}$
10: add all X's children to $\mathcal{X}$
11: $\mathcal{L} \leftarrow \{\text{leaf } X \in \mathcal{X} \mid arcs(X) \text{ satisfy constraints of Problem 1}\}$
12: $\mathcal{E}^* \leftarrow \arg\max_{arcs(X): X \in \mathcal{L}} \sum_{e \in arcs(X)} w(e)$

Search space. Given an S-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 possible (multi)subsets of arcs. The Settlement-bb algorithm represents this search space as a *binary tree* $\mathcal{T}$ with $|\mathcal{E}|+1$ levels, where each level (except for the root one) logically represents an arc $e \in \mathcal{E}$ for which a decision has to be taken, i.e., include the arc or not in the output solution (Figure 3). Correspondence between arcs and levels comes from some ordering on the arcs (e.g., by non-increasing weight, as in Section 4.3). A path from the root to a leaf represents a complete individual solution $\hat{\mathcal{E}} \in 2^{\mathcal{E}}$ (where a decision has been taken for all arcs). A non-leaf tree-node[3] represents a set of solutions: those corresponding to all possible decisions for the arcs not in the path from the root to that non-leaf node.

Search-space exploration. The Settlement-bb algorithm explores the tree-like search space (according to some visiting strategy, e.g., bfs or dfs) by exploiting a *lower bound* and an *upper bound* on the set of all solutions identified by a non-leaf tree-node it has visited, and keeping track of the largest lower-bound among all the ones computed so far. Whenever the upper bound of a tree-node is smaller than the largest so-far lower bound, that node and the whole subtree rooted in it can safely be discarded. The visit stops when all tree-nodes have been visited or pruned, and the optimal solution $\mathcal{E}^*$ is selected among all survivor leaves, specifically as the one satisfying all the constraints of Max-profit Balanced Settlement and exhibiting the maximum objective-function value. Note that, during the visit of the search space, intermediate solutions whose partial decisions make them (temporarily) infeasible *are not discarded*, because such solutions may become feasible later on. The general scheme of Settlement-bb is outlined as Algorithm 1. A specific visiting strategy of the search space (e.g., bfs or dfs) can be implemented by properly defining the way of choosing the next tree-node to be processed (Line 4). A crucial point in Settlement-bb is the definition of lower bound and upper bound. We discuss this in the remainder of the subsection.

[3]We use the term "tree-node" to refer to the nodes of the tree-like search space (to distinguish them from the nodes of the input S-multigraph).

Algorithm 2: Settlement-bb-lb

Input: An S-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, two multisets $\mathcal{E}_X^+ \subseteq \mathcal{E}, \mathcal{E}_X^- \subseteq \mathcal{E}$
Output: A multiset $\hat{\mathcal{E}} \subseteq \mathcal{E} \setminus \mathcal{E}_X^-$

1: $\mathcal{C} \leftarrow$ cycles of multigraph $\mathcal{G}^- = (\mathcal{V}, \mathcal{E} \setminus \mathcal{E}_X^-, w)$ {cf. [8]}
2: $\hat{\mathcal{E}} \leftarrow \emptyset$, $\hat{\mathcal{C}} \leftarrow \emptyset$
3: **while** $\mathcal{C} \neq \emptyset \wedge \mathcal{E}_X^+ \not\subseteq \hat{\mathcal{E}}$ **do**
4: $\mathcal{C} \leftarrow \{C \in \mathcal{C} \mid \hat{\mathcal{E}} \cup C \text{ meets Constraint (1) of Problem 1}\}$
5: $C \leftarrow$ cycle in $\mathcal{C}$ minimizing $[|C \cap (\mathcal{E}_X^+ \setminus \hat{\mathcal{E}})| \times \sum_{e \in C \setminus \hat{\mathcal{E}}} w(e)]^{-1}$
6: $\hat{\mathcal{C}} \leftarrow \hat{\mathcal{C}} \cup \{C\}$, $\mathcal{C} \leftarrow \mathcal{C} \setminus \{C\}$, $\hat{\mathcal{E}} \leftarrow \hat{\mathcal{E}} \cup C$
7: **while** $\mathcal{C} \neq \emptyset$ **do**
8: $\mathcal{C} \leftarrow \{C \in \mathcal{C} \mid \hat{\mathcal{E}} \cup C \text{ meets Constraint (1) of Problem 1}\}$
9: $C \leftarrow$ cycle in $\mathcal{C}$ maximizing $\sum_{e \in C \setminus \hat{\mathcal{E}}} w(e)$
10: $\mathcal{C} \leftarrow \mathcal{C} \setminus \{C\}$, $\hat{\mathcal{E}} \leftarrow \hat{\mathcal{E}} \cup C$
11: **if** $\mathcal{E}_X^+ \not\subseteq \hat{\mathcal{E}}$ **then** $\hat{\mathcal{E}} \leftarrow \emptyset$

Lower bound. For a tree-node X at level i of $\mathcal{T}$, let $\mathcal{E}_X$ denote the arcs for which a decision has been taken, i.e., arcs corresponding to all levels from the root to level i. Let also $\mathcal{E}_X$ be partitioned into $\mathcal{E}_X^+$ and $\mathcal{E}_X^-$, i.e., arcs included and not included in the current (partial) solution. A lower bound on the solutions spanned by (the subtree rooted in) X can be defined by computing *any feasible solution* $\hat{\mathcal{E}}$ to Max-profit Balanced Settlement, subject to the additional constraint of containing all arcs in $\mathcal{E}_X^+$ and no arcs in $\mathcal{E}_X^-$.

To compute such a feasible solution, we aim at finding the set $\mathcal{C}$ of *cycles* of the multigraph induced by the arc set $\mathcal{E} \setminus \mathcal{E}_X^-$, and greedily selects cycles based on their amount, as long as they meet Constraint (1) of Max-profit Balanced Settlement. 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. Moreover, cycle enumeration is a well-known problem, for which a variety of algorithms exists. As a trade-off between effectiveness, efficiency and simplicity, we employ a variant of the classic Johnson's algorithm [10] that works on multigraphs [8].

More in detail, the algorithm at hand is dubbed Settlement-bb-lb and outlined as Algorithm 2. To guarantee the inclusion of the arcs $\mathcal{E}_X^+$, the greedy cycle-selection step (Lines 8–11) is preceded by a *covering* phase (Lines 3–7), whose goal is to find a first subset $\hat{\mathcal{C}} \subseteq \mathcal{C}$ of cycles that (*i*) cover all arcs in $\mathcal{E}_X^+$, i.e., $\mathcal{E}_X^+ \subseteq \bigcup_{C \in \hat{\mathcal{C}}} C$, (*ii*) maximize the total amount of the cycle set, i.e., $\sum_{e \in \bigcup_{C \in \hat{\mathcal{C}}} C} w(e)$, and (*iii*) remain feasible for Max-profit Balanced Settlement. This corresponds to a variant of the well-known Weighted Set Cover problem, where $\mathcal{E}_X^+$ represents the universe of elements, while the cycles in $\mathcal{C}$ represent the covering sets. The covering phase of Settlement-bb-lb is therefore tackled by adapting the classic greedy $(1 + \log |\mathcal{E}_X^+|)$-approximation algorithm [3] for Weighted Set Cover, which iteratively selects the set minimizing the ratio between the set cost and the number of uncovered elements within that set, until all elements have been covered. Our adaptation consists in (*i*) defining the cost of a set as the inverse of the amount of the corresponding cycle (computed by discarding arcs already part of the output solution), and (*ii*) checking whether the Max-profit Balanced Settlement constraints

are satisfied while selecting a cycle. In the event that not all arcs in $\mathcal{E}_X^+$ have been covered, the algorithm returns an empty set (and the lower bound used in Settlement-BB is set to zero).

Time complexity: The running time of Settlement-BB-LB is dominated by cycle enumeration (Line 2), as the number of cycles in a (multi)graph can be exponential. In our context this is however not blocking: 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. The complexity of the remaining steps is as follows. The covering phase (Lines 3–6) can be implemented by using a priority queue with logarithmic-time insertion/extraction. It comprises: (*i*) computing set-cover score and checking the problem constraints for all cycles ($O(L|\mathcal{C}|)$ time); (*ii*) adding/extracting cycles to/from the queue ($O(|\mathcal{C}|\log|\mathcal{C}|)$ time); (*iii*) once a cycle C has been processed, updating the score of all cycles sharing some edges with C ($O(L|\mathcal{C}|\log|\mathcal{C}|)$ time, as each cycle is updated at most L times, and, for every time, it should be removed from the queue and re-added with updated score). Hence, the covering phase takes $O(L|\mathcal{C}|\log|\mathcal{C}|)$ time. In greedy selection (Lines 7–11) cycles are processed one by one in non-increasing amount order, and added to the solution after checking (in $O(L)$ time) the problem constraints. This yields a $O(|\mathcal{C}|(\log|\mathcal{C}|+L))$ time.

Upper bound. The Settlement-BB upper bound lies on a relaxation of Max-profit Balanced Settlement where Constraint (2) is discarded and arcs are allowed to be selected *fractionally*:

Problem 2 (Relaxed Settlement). *Given an S-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, find $\{x_e \in [0,1]\}_{e\in\mathcal{E}}$ so as to*

$$\begin{array}{ll} \textit{Maximize} & \sum_{e\in\mathcal{E}} x_e w(e) \\ \textit{subject to} & \left(\sum_{e=(v,u)\in\mathcal{E}} x_e w(e) - \sum_{e=(u,v)\in\mathcal{E}} x_e w(e)\right) \\ & \qquad \in [fl(u)-bl_a(u), cap(u)-bl_r(u)], \ \ \forall u \in \mathcal{V} \end{array}$$

The desired upper bound relies on an interesting characterization of Relaxed Settlement as a network-flow problem. As a major result in this regard, we show that solving Relaxed Settlement on multigraph $\mathcal{G}$ is equivalent to solving the well-established Min-Cost Flow problem [1] on an ad-hoc modified version of $\mathcal{G}$. We start by recalling the Min-Cost Flow problem:

Problem 3 (Min-Cost Flow [1]). *Given a simple directed graph $G = (V, E)$, a cost function $c : E \to \mathbb{R}$, lower-bound and upper-bound functions $\lambda : E \to \mathbb{R}$, $\mu : E \to \mathbb{R}$, and a supply/demand function $b : V \to \mathbb{R}$, find a flow $f : E \to \mathbb{R}$ so as to*

$$\begin{array}{ll} \textit{Minimize} & \sum_{e\in E} c(e) f(e) \\ \textit{subject to} & \lambda(e) \le f(e) \le \mu(e), \ \ \forall e \in E \\ & \sum_{u:(v,u)\in E} f(v,u) - \sum_{u:(u,v)\in E} f(u,v) = b(u), \ \ \forall u \in V \end{array}$$

The modified version of $\mathcal{G}$ considered in this context is as follows:

Definition 4.1 (S-flow graph). The *S-flow graph* $\mathcal{G}_f = (\mathcal{V}_f, \mathcal{E}_f, w_f)$ of an S-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ is a simple weighted directed graph where:

- All arcs $(u, v) \in \mathcal{E}$ between the same pair of nodes are collapsed into a single one, and the weight $w_f(u, v)$ is set to $\sum_{(u,v)\in\mathcal{E}} w(u, v)$;
- $\mathcal{V}_f = \mathcal{V} \cup \{\tilde{s}, \tilde{t}\}$, i.e., the node set of $\mathcal{G}_f$ is composed of all nodes of $\mathcal{G}$ along with two dummy nodes $\tilde{s}$ and $\tilde{t}$;
- $\mathcal{E}_f = \mathcal{E} \cup \{(\tilde{s}, u) \mid u \in \mathcal{V}\} \cup \{(u, \tilde{t}) \mid u \in V\} \cup \{(\tilde{t}, \tilde{s})\}$, i.e., the arc set of $\mathcal{G}_f$ is composed of (*i*) all (collapsed) arcs of $\mathcal{G}$, (*ii*) for each node $u \in \mathcal{V}$, a dummy arc $(\tilde{s}, u)$ with weight $w_f(\tilde{s}, u) = bl_a(u) - fl(u)$ and a dummy arc $(u, \tilde{t})$ with weight $w_f(u, \tilde{t}) = cap(u) - bl_r(u)$, and (*iii*) a dummy arc $(\tilde{t}, \tilde{s})$ with weight $w_f(\tilde{t}, \tilde{s}) = \infty$.

Algorithm 3: Settlement-BB-UB

Input: An S-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, two multisets $\mathcal{E}_X^+ \subseteq \mathcal{E}, \mathcal{E}_X^- \subseteq \mathcal{E}$
Output: A real number UB_X
1: $\mathcal{G}^- := (\mathcal{V}, \mathcal{E} \setminus \mathcal{E}_X^-, w)$
2: $UB_X \leftarrow$ solve Min-Cost Flow applying Theor. 4.2 on $\mathcal{G}^-$ and forcing flow $f(e) = w(e)$, $\forall e \in \mathcal{E}_X^+$; return –1 if no admissible solution exists

The main result for the computation of the desired upper bound is stated in the next theorem and corollary:

Theorem 4.2. *Given an S-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, let $\mathcal{G}_f = (\mathcal{V}_f, \mathcal{E}_f, w_f)$ be the S-flow graph of $\mathcal{G}$. Let also cost, lower-bound, upper-bound and supply/demand functions $c : \mathcal{E}_f \to \mathbb{R}$, $\lambda : \mathcal{E}_f \to \mathbb{R}$, $\mu : \mathcal{E}_f \to \mathbb{R}$ and $b : \mathcal{V}_f \to \mathbb{R}$ be defined as:*

- $\lambda(e) = 0$, $\mu(e) = w_f$, $\forall e \in \mathcal{E}_f$;
- $c(\tilde{t}, \tilde{s}) = 0$, *and* $c(\tilde{s}, u) = c(u, \tilde{t}) = 0$, $\forall u \in \mathcal{V}_f$;
- $c(e) = -1$, $\forall e \in \mathcal{E}_f \cap \mathcal{E}$;
- $b(u) = 0$, $\forall u \in \mathcal{V}_f$.

It holds that solving Min-Cost Flow *on input $\langle \mathcal{G}_f, c, \lambda, \mu, b\rangle$ is equivalent to solving* Relaxed Settlement *on input $\mathcal{G}$.*

Proof. As $c(e) = -1$, if $e \in \mathcal{E}$, $c(e) = 0$, otherwise, and $\forall e \in \mathcal{E}$: $0 \le f(e) \le w(e)$, the objective function of Min-Cost Flow on $\mathcal{G}_f$ can be rewritten as $\min \sum_{e\in\mathcal{E}} -f(e)$, which is equivalent to the objective $\max \sum_{e\in\mathcal{E}} x_e w(e)$ ($x_e \in [0,1]$) of Relaxed Settlement on $\mathcal{G}$. For the cap-floor constraints, the conservation of flows ensures for any solution to Min-Cost Flow on $\mathcal{G}_f$:

$$\begin{array}{l} \forall u \in \mathcal{V}: \sum_{(v,u)\in\mathcal{E}} f(v,u) + f(\tilde{s},u) - \sum_{(u,v)\in\mathcal{E}} f(u,v) - f(u,\tilde{t}) = b(u) = 0 \\ \Leftrightarrow \forall u \in V: \sum_{(v,u)\in\mathcal{E}} f(v,u) - \sum_{(u,v)\in\mathcal{E}} f(u,v) = f(u,\tilde{t}) - f(\tilde{s},u). \end{array}$$

As $f(\tilde{s}, u) \in [f_{min}(\tilde{s}, u), f_{max}(\tilde{s}, u)] = [0, bl_a(u) - fl(u)]$ and $f(u, \tilde{t}) \in [f_{min}(u, \tilde{t}), f_{max}(u, \tilde{t})] = [0, cap(u) - bl_r(u)]$, then:

$$\begin{array}{l} \forall u \in \mathcal{V} : \sum_{(v,u)\in\mathcal{E}} f(v,u) - \sum_{(u,v)\in\mathcal{E}} f(u,v) \\ = f(u,\tilde{t}) - f(\tilde{s},u) \le f_{max}(u,\tilde{t}) - f_{min}(\tilde{s},u) = cap(u) - bl_r(u), \end{array}$$

$$\begin{array}{l} \text{and } \forall u \in \mathcal{V} : \sum_{(v,u)\in\mathcal{E}} f(v,u) - \sum_{(u,v)\in\mathcal{E}} f(u,v) \\ = f(u,\tilde{t}) - f(\tilde{s},u) \ge f_{min}(u,\tilde{t}) - f_{max}(\tilde{s},u) = fl(u) - bl_a(u). \end{array}$$

Overall, it is therefore guaranteed that $\forall u \in \mathcal{V}$:

$$\sum_{(v,u)\in\mathcal{E}} f(v,u) - \sum_{(u,v)\in\mathcal{E}} f(u,v) \in [fl(u) - bl_a(u), cap(u) - bl_r(u)],$$

i.e., the floor-cap constraints in Relaxed Settlement. □

Corollary 4.3. *Given an S-multigraph $\mathcal{G}$, the solution to* Max-profit Balanced Settlement *on $\mathcal{G}$ is upper-bounded by the solution to* Min-Cost Flow *on the input $\langle \mathcal{G}_f, c, \lambda, \mu, b\rangle$ of Theorem 4.2.*

Proof. Immediate as, according to Theorem 4.2, the solution to Min-cost Flow on input $\langle \mathcal{G}_f, c, \lambda, \mu, b\rangle$ corresponds to the optimal solution to a relaxed version of the Max-profit Balanced Settlement problem on the original S-multigraph $\mathcal{G}$. □

Corollary 4.3 is exploited for upper-bound computation as in Algorithm 3. To handle the arcs to be discarded ($\mathcal{E}^-$) and included ($\mathcal{E}^+$), Algorithm 3 removes all arcs $\mathcal{E}^-$ from the multigraph, and asks for a Min-Cost Flow solution where the flow on every arc $e \in E^+$ is forced to be $w(e)$. If no solution satisfying such a requirement exists, no admissible solution to Max-profit Balanced Settlement exists in the entire subtree rooted in the target tree-node X.

In this case the returned upper bound is -1, and the subtree rooted in X is pruned by the Settlement-BB algorithm. We solve MIN-COST FLOW with the well-established Cost Scaling algorithm [6], which has $O(|\mathcal{E}|\,(|\mathcal{V}|\log|\mathcal{V}|)\,\log(|\mathcal{V}|\,w_{max}))$ time complexity, where $w_{max} = \max_{e\in\mathcal{E}} w(e)$. This also corresponds to the time complexity of the entire upper-bound computation method.

4.2 Beam-search algorithm

Being MAX-PROFIT BALANCED SETTLEMENT **NP**-hard, the exact Settlement-BB algorithm cannot handle large S-multigraphs. We thus design an alternative algorithm that finds approximated solutions and can run on larger instances. For the reasons explained in Section 4.1, we exploit again the idea of enumerating cycles and properly selecting them while keeping the constraints satisfied.

We base cycle selection on interesting theoretical findings, formalizing the OPTIMAL CYCLE SELECTION problem and characterizing it as a KNAPSACK problem. We ultimately devise an algorithm, dubbed Settlement-BEAM, combining KNAPSACK-solving methodologies with the principles of *beam search*. The details follow.

Optimal cycle selection. For a principled cycle selection, we start from seeking the cycles that satisfy the MAX-PROFIT BALANCED SETTLEMENT constraints and exhibit maximum total amount:

PROBLEM 4 (OPTIMAL CYCLE SELECTION). *Given an S-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:*

$$\mathcal{C}^* = \arg\max_{\hat{\mathcal{C}}\subseteq\mathcal{C}} \sum_{e\in\mathcal{E}(\hat{\mathcal{C}})} w(e)$$
$$\text{subject to} \quad \mathcal{E}(\hat{\mathcal{C}}) = \bigcup_{C\in\hat{\mathcal{C}}} C \text{ meets Constraint (1) in Problem 1.}$$

THEOREM 4.4. *Problem 4 is* **NP***-hard.*

PROOF. We reduce from MAXIMUM INDEPENDENT SET [4], 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, we construct an instance $\langle\mathcal{G},\mathcal{C}\rangle$ of OPTIMAL CYCLE SELECTION as follows. For a vertex $u\in V$, let $N(u)=\{v \mid (u,v)\in E\}$, $d_u = |N(u)|$, $d_{max} = \max_{u\in V} d_u$. Without loss of generality, we assume that $\forall u\in V: d_u > 0$. First, we let the node set $\mathcal{V}$ of $\mathcal{G}$ contain: (*i*) a pair of nodes $\langle u', u''\rangle$ for every vertex $u\in V$, (*ii*) a node uv for each $(u,v)\in E$. Let also $\mathcal{V}$ follow a global ordering. For each $u\in V$, we create a cycle $C_u = c_1\to c_2\to\cdots\to c_{d_u+2}\to c_1$ in $\mathcal{G}$, where $c_1 = u'$, $c_2 = u''$, and all other nodes $c_i: i>2$ correspond to nodes $\{uv\in\mathcal{V} \mid v\in N(u)\}$, ordered as picked above. The arc weights of C_u are defined as follows:

- $w(c_1,c_2) = 1+\frac{3}{2}[d_{max}(d_{max}+1) - d_u(d_u+1)]$,
- $w(c_i,c_{i+1}) = d_u + i - 2,\ \forall i\in[2..d_u+1]$,
- $w(c_{d_u+2},c_1) = 2d_u$.

Finally, we set

- $bl_r(x) = bl_a(x) = 0,\ cap(x) = +\infty,\ \forall x\in\mathcal{V}$,
- $fl(u') = fl(u'') = -\infty, \forall u', u''$,
- $fl(uv) = -1, \forall uv$,

and the input cycles $\mathcal{C}$ equal to $\{C_u \mid u\in V\}$. It holds that:

(a) $\mathcal{G}$ has $2|V|+|E|$ nodes and $\sum_{u\in V'}(d_u+2) = 2(|V|+|E|)$ arcs, taking polynomial space and construction time in the G's size.
(b) All cycles in $\mathcal{C}$ are arc-disjoint with respect to each other.
(c) Every cycle $C_u\in\mathcal{C}$ has the same total amount, which is equal to $1+\frac{3}{2}[d_{max}(d_{max}+1) - d_u(d_u+1)] + \sum_{i=0}^{d_u}(d_u+i) = 1+\frac{3}{2}[d_{max}(d_{max}+1) - d_u(d_u+1)] + \frac{3}{2}d_u(d_u+1) = 1+\frac{3}{2}d_{max}(d_{max}+1)$.
(d) Selecting any two cycles $C_u, C_v\in\mathcal{C}$ such that u and v are adjacent in G, violates the constraint on $fl(uv)$ (as $bl_a(uv)$ will result to be $bl_a(uv) = -2 < fl(uv) = -1$).

Based on (b) and (c), any cycle brings the same gain to the objective. Thus, solving OPTIMAL CYCLE SELECTION on $\langle\mathcal{G},\mathcal{C}\rangle$ corresponds to selecting the *maximum number* of cycles in $\mathcal{C}$ so that cap-floor constraints are met. Combined with (d), solving OPTIMAL CYCLE SELECTION on $\langle\mathcal{G},\mathcal{C}\rangle$ is equivalent to selecting the maximum number of vertices in G no two of which are adjacent. □

Characterization as a KNAPSACK problem. As OPTIMAL CYCLE SELECTION is **NP**-hard, we focus on designing effective approximated solutions. To this end, we show an intriguing connection with the following variant of the well-known KNAPSACK problem:

PROBLEM 5 (SET UNION KNAPSACK [7]). *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 $S_i\subseteq U,\ \forall i\in[1..k]$, $p:\mathcal{S}\to\mathbb{R}$ be a profit function for items in $\mathcal{S}$, and $q: U\to\mathbb{R}$ be a 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 $Q(\hat{\mathcal{S}}) = \sum_{x\in U(\hat{\mathcal{S}})} q(x)$. Given a real number $B\in\mathbb{R}$, SET UNION KNAPSACK finds $\mathcal{S}^* = \arg\max_{\hat{\mathcal{S}}\subseteq\mathcal{S}} P(\hat{\mathcal{S}})$ s.t. $Q(\hat{\mathcal{S}})\le B$.*

A simple variant of SET UNION KNAPSACK arises when both costs and budget constraint are d-dimensional:

PROBLEM 6 (MULTIDIMENSIONAL SET UNION KNAPSACK). *Given $U,\mathcal{S},p$ as in Problem 5, a d-dimensional cost function $q: U\to\mathbb{R}^d$, and a d-dimensional vector $\mathbf{B}\in\mathbb{R}^d$, find $\mathcal{S}^* = \arg\max_{\hat{\mathcal{S}}\subseteq\mathcal{S}} P(\hat{\mathcal{S}})$ s.t. $\mathbf{Q}(\hat{\mathcal{S}})\le\mathbf{B}$, where $\mathbf{Q}(\hat{\mathcal{S}}) = \sum_{x\in U(\hat{\mathcal{S}})} q(x)$.*

We observe that an instance of OPTIMAL CYCLE SELECTION can be transformed into an instance of MULTIDIMENSIONAL SET UNION KNAPSACK so that every feasible solution for the latter instance is also a feasible solution for the original OPTIMAL CYCLE SELECTION instance. We let the arcs $\mathcal{E}$ represent elements, cycles $\mathcal{C}$ represent items, and define $2|\mathcal{V}|$ costs/budgets for each element (arc), so as to match the cap-floor constraints on every node ($|\mathcal{V}|$ costs/budgets for cap- and floor-related constraints each). Formally:

THEOREM 4.5. *Given an S-multigraph $\mathcal{G}=(\mathcal{V},\mathcal{E},w)$ and a set $\mathcal{C}$ of cycles in $\mathcal{G}$, let $\langle U,\mathcal{S},p,q,\mathbf{B}\rangle$ be an instance of* MULTIDIMENSIONAL SET UNION KNAPSACK *defined as follows:*

- $U=\mathcal{E};\ \mathcal{S}=\mathcal{C};\ \forall C\in\mathcal{C}: p(C) = \sum_{e\in C} w(e)$;
- $\forall(u_i,u_j)\in\mathcal{E}: q(u_i,u_j) = [\mathbf{q}^+(u_i,u_j)\,\mathbf{q}^-(u_i,u_j)]\in\mathbb{R}^{2|\mathcal{V}|}$, *where*
  - $\mathbf{q}^+(u_i,u_j) = [q^+_k(u_i,u_j)]_{k=1}^{|\mathcal{V}|}$, *with* $q^+_i(u_i,u_j) = -w(u_i,u_j)$, $q^+_j(u_i,u_j) = w(u_i,u_j)$, *and* $q^+_k(u_i,u_j) = 0$, *for* $k\ne i,j$;
  - $\mathbf{q}^-(u_i,u_j) = [q^-_k(u_i,u_j)]_{k=1}^{|\mathcal{V}|}$, *with* $q^-_i(u_i,u_j) = w(u_i,u_j)$, $q^-_j(u_i,u_j) = -w(u_i,u_j)$, *and* $q^-_k(u_i,u_j) = 0$, *for* $k\ne i,j$;
- $\mathbf{B} := [\mathbf{B}^+\ \mathbf{B}^-]\in\mathbb{R}^{2|\mathcal{V}|}$, *where*
  - $\mathbf{B}^+ = [cap(u_1) - bl_r(u_1),\ldots,cap(u_n) - bl_r(u_n)]$;
  - $\mathbf{B}^- = [bl_a(u_1) - fl(u_1),\ldots,bl_a(u_n) - fl(u_n)]$.

It holds that any feasible solution for MULTIDIMENSIONAL SET UNION KNAPSACK *on input $\langle U,\mathcal{S},p,q,\mathbf{B}\rangle$ is a feasible solution for* OPTIMAL CYCLE SELECTION *on input $\langle\mathcal{G},\mathcal{C}\rangle$.*

Algorithm 4: Settlement-BEAM

Input: An S-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, an integer K
Output: A multiset $\mathcal{E}^* \subseteq \mathcal{E}$

1: $\mathcal{E}^* \leftarrow \emptyset$
2: $\mathcal{C} \leftarrow$ cycles of $\mathcal{G}$ {cf. [8]}
3: **while** $\mathcal{C} \neq \emptyset$ **do**
4: $\quad \mathcal{C}' \leftarrow K$-sized subset of $\mathcal{C}$ by Greedy MAX COVER {cf. [9]}
5: $\quad \mathcal{C}'_2 \leftarrow \{\{C_i, C_j\} \mid C_i, C_j \in \mathcal{C}'\}$
6: $\quad$ **for all** $\{C_i, C_j\} \in \mathcal{C}'_2$ **do**
7: $\quad\quad C_{ij} \leftarrow C_i \cup C_j$
8: $\quad\quad$ 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 Prob. 4
9: $\quad \mathcal{E}^* \leftarrow \mathcal{E}^* \cup \arg\max_{C_{ij} \in \mathcal{C}'_2} \sum_{e \in C_{ij}} w(e)$
10: $\quad \mathcal{C} \leftarrow \mathcal{C} \setminus (\mathcal{C}' \cup \{C \in \mathcal{C} \mid C \cap \mathcal{E}^* = C\})$

PROOF. (SKETCH) It suffices to show that the constraints of MULTIDIMENSIONAL SET UNION KNAPSACK on $\langle U, \mathcal{S}, p, q, \mathbf{B}\rangle$ correspond to cap-floor constraints of OPTIMAL CYCLE SELECTION on $\langle \mathcal{G}, \mathcal{C}\rangle$. This can be achieved by simple math on $\mathbf{q}^+, \mathbf{q}^-, \mathbf{B}^+, \mathbf{B}^-$. □

Putting it all together. Motivated by Theorem 4.5, we devise an algorithm to approximate OPTIMAL CYCLE SELECTION inspired by Arulselvan's algorithm for SET UNION KNAPSACK [2], which achieves a $1-e^{-1/f_{max}}$ approximation guarantee, where f_{max} is the maximum number of items in which an element is present. Arulselvan's algorithm considers all subsets of 2 items whose weighted union is within the budget B. Then, it augments each subset with items S_i added one by one in the decreasing order of an ad-hoc-defined score, as long as the inclusion of S_i complies with the budget constraint B. The score exploited for item processing is directly proportional to the profit of S_i and inversely proportional to the frequency of S_i's elements within the entire item set $\mathcal{S}$. The highest-profit one of such augmented subsets is returned as output.

The ultimate Settlement-BEAM algorithm (Algorithm 4) combines the ideas behind Arulselvan's algorithm with the *beam-search* paradigm and a couple of adaptations to make it suitable for our context. Particularly, the adaptations are as follows. (*i*) We extend Arulselvan's 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 (trivial extension). (*ii*) We define the score of a cycle $C \in \mathcal{C}$ as:

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

whose rationale is to consider the total cycle amount, penalized by a term accounting for the frequency of an arc within the whole cycle set $\mathcal{C}$. The idea is that frequent arcs contribute less to the objective function, which is defined on the *union* of the arcs of all selected cycles (see Problem 4). Finally, (*iii*) to overcome the expensive pairwise cycle enumeration and augmentation of *all* 2-sized cycle sets with *all* other cycles, we adopt a beam-search methodology to reduce the cycles to be considered. 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,[4] and use $\mathcal{C}'$ for both 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.

[4] To solve this step, we note that the problem at hand is an instance of (weighted) MAX COVER [9], where arcs correspond to elements and cycles correspond to sets. We hence employ the classic greedy $(1 - \frac{1}{e})$-approximation algorithm, which iteratively adds to the solution the cycle maximizing the sum of the weights of still uncovered arcs.

Algorithm 5: Settlement-HYBRID

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

1: $\mathcal{E}^* \leftarrow \emptyset$, CC $\leftarrow$ weakly connected components of $\mathcal{G}$
2: **for all** $G \in$ CC s.t. $|arcs(G)| \leq H$ **do**
3: $\quad \mathcal{E}^* \leftarrow \mathcal{E}^* \cup$ Settlement-BB on input G {Algorithm 1}
4: **for all** $G \in$ CC s.t. $|arcs(G)| > H$ **do**
5: $\quad \mathcal{E}^* \leftarrow \mathcal{E}^* \cup$ Settlement-BEAM on input $\langle G, K\rangle$ {Algorithm 4}

Time complexity. As far as cycle enumeration, the considerations made for Algorithm 2 (Section 4.1) remain valid here too. The complexity of the remaining main steps is as follows. Denoting by L the maximum size of a cycle in $\mathcal{C}$, the greedy MAX COVER algorithm on input $\mathcal{C}'$ (Line 4) takes $O(LK \log K)$ time. All feasible cycle pairs (Line 5) can be computed in $O(LK^2)$ time, while augmentation of all such pairs (Lines 6–8) takes $O(LK^3)$ time. All these steps are repeated $O(\frac{|\mathcal{C}|}{K})$ times. The overall time complexity is $O(LK^2|\mathcal{C}|)$.

4.3 Hybrid algorithm

As a straightforward observation, MAX-PROFIT BALANCED SETTLEMENT can be solved by running any of the algorithms described above on each weakly connected component of the input S-multigraph *separately*, and then taking the union of all partial solutions. This motivates us to devise a *hybrid* strategy which runs the exact Settlement-BB algorithm on the smaller connected components and the Settlement-BEAM algorithm on the remaining ones. This algorithm, termed Settlement-HYBRID, is the algorithm we ultimately propose in this work. We report its outline as Algorithm 5.

Implementation details. We improve the efficiency of all algorithms observing that a node with no incoming or outgoing arc can be filtered out of the input S-multigraph, as it violates Constraint (2) in Problem 1. Such a filtering may be exploited recursively, extracting the $(1, 1)$-*D-core* of the input S-multigraph [5]. Regarding the tree-like search space of Settlement-BB, we sort the arcs by non-increasing amount, as larger-amount arcs are more likely to contribute more to the optimal solution. Finally, we experimented with both BFS and DFS visiting strategies, without observing any substantial difference.

5 EXPERIMENTS

We tested the performance of our algorithms on a random sample of a *real dataset* provided by UniCredit, a noteworthy European banking company. The sample consists of 5 413 375 receivables and 369 479 (anonymized) customers, spanning one year in 2015-16.

Customers' attributes. We set customers' attributes based on statistics computed on a training prefix of 3 months of data.

As far as the initial actual balance $bl_a(u)$ of each customer u, we computed the difference between the total amount of her passive receivables and the total amount of her active receivables; we considered the days were such a difference was negative, and ultimately set $bl_a(u)$ as the average of the absolute values of such negative daily differences. The rationale is that in those days the customer would have needed extra liquidity with respect to the amount cashable from incoming receivables, to handle the payment of the receivables in which she was listed as payer. Assuming that in reality customers would try this new service with a limited initial cash deposit, we also imposed a hard upper bound of $50K$ euros for the initial bl_a of every customer.

Table 1: Size of the input S-multigraphs.

	Worst Scenario						Normal Scenario						Best Scenario					
	CAP $< \infty$			CAP $= \infty$			CAP $< \infty$			CAP $= \infty$			CAP $< \infty$			CAP $= \infty$		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
#Nodes	16 070	40 810	58 390	16 060	40 810	58 380	27 890	62 890	82 930	27 890	62 870	82 900	42 330	80 260	97 400	42 320	80 230	97 380
#Arcs	14 450	41 720	69 270	14 430	41 670	69 180	28 680	76 490	116 000	28 640	76 310	115 600	50 600	111 100	149 300	50 520	110 700	148 500

As for the *cap*, we considered: (*i*) $cap < \infty$, and (*ii*) $cap = \infty$. In the former case each customer was assigned a finite *cap*, equal to the average amount she received in the training data. Customers who received no payments in the training interval were assigned a default *cap* equal to the average *cap* of all other customers. In the $cap = \infty$ setting we instead allowed all accounts to grow arbitrarily. Finally, we set $fl(u) = 0$, for each customer u.

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 explained 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) *Infinite CAP.* Here we also considered a *Worst*, a *Normal*, and a *Best* scenario with the same values of $life$ as in the corresponding finite-CAP case, but we set $\forall I \in \mathcal{I},\ \ cap(u) = \infty\ , \forall u \in \mathcal{U}$.

Such settings identify different sets of valid receivables for a day, thus yielding different input multigraphs. Table 1 reports graph sizes. More complex scenarios clearly lead to larger graphs. Conversely, as the *cap* goes from $<\infty$ to ∞ (in the same scenario), graphs get smaller. This is motivated as $cap < \infty$ represents a tighter constraint for receivable settlement, i.e., more receivables not settled any day which will be included in the input graph of the next day.

Algorithms. We compared our ultimate proposal, i.e., Settlement-HYBRID (Algorithm 5), against two baselines: the simple greedy-cycle-selection Settlement-BB-LB algorithm (Algorithm 2), and the beam-search Settlement-BEAM algorithm (Algorithm 4). Clearly, the exact Settlement-BB (Algorithm 1) could not afford the size of the real graphs involved in our experiments. However, we recall that it is part of Settlement-HYBRID, where it is employed to handle the smaller connected components. To get an idea of its performance, one can thus resort to the comparison Settlement-HYBRID vs. Settlement-BEAM. As for parameter setting, all experiments refer to $L = 15$ (all algorithms based on cycle enumeration), $H = 20$ (Settlement-BEAM), and $K = 1\,000$ (Settlement-BEAM and Settlement-HYBRID). Observe that we do not have external baselines to employ, as this is (to the best of our knowledge) the first attempt to exploit the receivable network to optimize RF services.

Assessment criteria. We assess the performance of the various algorithms by measuring the total amount of the receivables that each algorithm selects for automatic settlement. This metric provides direct evidence of the benefits gained by both the funder and the customers. In fact, the greater the amount, the less liquidity the funder is requested to anticipate, and the larger the saving for customers due to the reduced fees of the network-based service.

Testing environment. All algorithms were implemented in Scala (v. 2.12). Experiments in Table 2 were run on a commodity laptop equipped with an Intel Core i7 CPU at 2.8GHz and 16GB RAM. Experiments in Table 3 were run on a i9 Intel 7900x 3.3GHz, 128GB RAM machine, which we limited to 60GB in our tests.

General performance. Table 2 shows the results of our experiments. For each scenario, we split the dataset into 3-month periods, to have a better understanding of the performance on a quarterly basis, and speed up the evaluation by parallelizing the computations on different quarters. For each scenario/quarter pair, we report: total amount (euros) of the settled receivables, per-day running time (seconds) averaged across all days in the quarter, total number of settled receivables, and number of distinct customers involved in at least a daily solution. For Settlement-HYBRID we also report the percentage gain on the total amount with respect to the baselines.

Settlement-HYBRID outperforms both baselines in terms of settled amount. In the finite-*cap*, worst scenario, the gain of Settlement-HYBRID over Settlement-BB-LB is 150% on average, with a maximum of 460%. As for infinite *cap*, Settlement-HYBRID yields avg gain over Settlement-BB-LB of 70%, 28%, and 30.6% in the three scenarios. The superiority of Settlement-HYBRID is confirmed over the other baseline, Settlement-BEAM: the average gain is 139%, 45%, and 33% in the finite-*cap* scenarios, and 37%, 16%, and 9% in the infinite-*cap* cases, with maximum gain of 392%. *This attests the relevance of employing the exact algorithm even only on small components.* In one quarter Settlement-BEAM and Settlement-HYBRID perform the same: the corresponding graph has no small components where the exact solution improves upon Settlement-BEAM.

Concerning running times, the fastest method is the simplest one, i.e., Settlement-BB-LB: in the finite-*cap* cases it takes on average 10*s*. The proposed Settlement-HYBRID takes from 3.5*s* to 22*mins*, remaining perfectly compliant with the settings of the service being built, which computes solutions offline, at the end of each day. Settlement-HYBRID is comparable to Settlement-BEAM, due to the fact that the computation on large components dominates.

Scalability. To test the scalability of Settlement-HYBRID, we considered the finite-cap, normal scenario and randomly sampled segments of data corresponding to 5, 10, 15, 30, 60, and 90 days. We collapsed the set of receivables of each segment into one single S-multigraph, and ran Settlement-HYBRID on it, setting $L = 10$ and $K = 100$. The goal of this experiment was to assess the running time of Settlement-HYBRID on larger graphs. Table 3 reports the outcome of this experiment, showing, for each segment, size of the S-multigraph, settled amount, and running time (in seconds). Note that amounts in Table 2 are generally larger than those reported here. The two experiments are not comparable in terms of amount, as Table 2 reports amounts summed over all time instants of a quarter, each expanded to a window of 5, 10 or 15 days depending on the $life$ parameter. Running time is instead comparable up to the segments of 15 days, because the first table reports average per-day running time. Indeed, when up to 15 days are considered, the achieved times are consistent with those reported before. On a period of 30 days, the algorithm is still very efficient, taking slightly more than a minute. The running time increases on the two larger segments: the algorithm employs less than one hour on the two-month segment, and around 7.5 hours on the three-month one. Albeit the cost increase is remarkable, note that the network-based

Table 2: Experimental results: proposed Settlement-HYBRID algorithm vs. baselines.

CAP	Scenario	Start Date	End Date	Settlement-BB-LB Amount	BB-LB Time (s)	BB-LB Receivables	BB-LB Clients	Settlement-BEAM Amount	BEAM Time (s)	BEAM Receivables	BEAM Clients	Settlement-HYBRID Amount	%Gain vs. S-BB-LB	%Gain vs. S-BEAM	HYBRID Time (s)	HYBRID Receivables	HYBRID Clients
<∞	worst	2015-07-01	2015-09-30	95 397 950	1.78	1827	859	79 014 431	3.93	2532	939	**146 703 722**	53.78	85.67	3.54	3330	1208
		2015-10-01	2015-12-31	121 680 562	1.80	1798	863	111 184 064	9.22	2222	891	**157 007 283**	29.03	41.21	9.45	3263	1248
		2016-01-01	2016-03-31	52 232 665	1.93	2066	987	59 767 431	14.39	2730	1034	**82 875 118**	58.67	38.66	14.47	3619	1318
		2016-04-01	2016-06-30	46 457 493	1.93	1862	950	52 872 191	16.49	2499	987	**260 196 811**	460.08	392.12	13.72	3296	1254
<∞	normal	2015-07-01	2015-09-30	162 215 076	2.73	4535	1986	146 759 007	99.09	6673	2269	**221 401 342**	36.49	50.86	75.18	7951	2685
		2015-10-01	2015-12-31	151 707 991	2.83	4305	1975	133 556 473	47.52	5906	2141	**168 556 675**	11.11	26.21	48.14	7080	2548
		2016-01-01	2016-03-31	143 547 779	3.01	5093	2200	158 408 830	76.15	7319	2464	**208 011 456**	44.91	31.31	113.72	8172	2730
		2016-04-01	2016-06-30	149 813 738	3.01	5137	2296	162 254 519	104.73	7459	2603	**277 568 569**	85.28	71.07	117.45	8551	2941
<∞	best	2015-07-01	2015-09-30	229 604 568	7.63	7443	3115	263 379 260	172.09	11 045	3566	**270 221 655**	17.69	2.60	158.36	12 004	3957
		2015-10-01	2015-12-31	195 171 304	4.00	7183	3108	208 162 233	133.60	10 364	3506	**251 353 161**	28.79	20.75	143.71	11 669	3917
		2016-01-01	2016-03-31	236 195 806	9.73	8397	3516	264 654 596	193.91	12 564	4000	**314 861 548**	33.31	18.97	205.36	13 486	4333
		2016-04-01	2016-06-30	210 056 244	8.92	9094	3801	234 928 905	212.45	13 184	4259	**449 862 686**	114.16	91.49	239.25	14 360	4620
∞	worst	2015-07-01	2015-09-30	194 399 317	2.56	2863	1275	215 249 395	110.43	5418	1535	**295 595 794**	52.06	37.33	143.39	6528	1872
		2015-10-01	2015-12-31	215 635 790	2.30	2939	1283	263 672 222	124.54	5285	1490	**319 804 764**	48.31	21.29	125.16	6724	1918
		2016-01-01	2016-03-31	206 616 203	12.06	3329	1371	262 778 883	343.81	6876	1718	**302 740 733**	46.52	15.21	354.13	8290	2116
		2016-04-01	2016-06-30	216 264 862	2.56	3224	1383	286 859 858	287.28	6034	1685	**504 402 450**	133.23	75.84	304.92	7241	1992
∞	normal	2015-07-01	2015-09-30	553 364 544	64.58	7131	2836	660 907 304	1168.95	15 323	3761	**779 733 449**	40.91	17.98	1006.17	17 082	4268
		2015-10-01	2015-12-31	643 722 123	6.67	6742	2736	663 873 349	618.98	14 570	3507	**784 314 578**	21.84	18.14	690.14	16 761	4133
		2016-01-01	2016-03-31	693 852 990	29.03	7999	3034	743 945 529	1159.27	17 390	4143	**827 346 450**	19.24	11.21	1329.80	19 544	4701
		2016-04-01	2016-06-30	751 368 135	30.81	8289	3189	855 932 063	757.85	17 666	4155	**987 866 224**	31.48	15.41	865.92	19 576	4718
∞	best	2015-07-01	2015-09-30	916 036 152	8.35	11 246	4231	1 110 498 564	866.00	23 055	5353	**1 172 926 462**	28.04	5.62	988.04	24 811	5917
		2015-10-01	2015-12-31	1 028 777 612	26.28	10 937	4159	1 275 000 083	842.56	23 235	5333	**1 565 296 054**	52.15	22.77	1182.68	25 469	5901
		2016-01-01	2016-03-31	1 329 747 599	54.87	13 271	4830	**1 489 713 871**	993.90	27 404	6130	**1 489 713 871**	12.03	0	1101.04	27 404	6130
		2016-04-01	2016-06-30	1 270 225 872	22.05	13 337	4835	1 524 865 674	904.29	26 177	5781	**1 657 784 888**	30.51	8.72	886.05	27 746	6239

Table 3: Scalability of the proposed Settlement-HYBRID algorithm.

Days	Nodes	Arcs	Amount	Time (s)	Days	Nodes	Arcs	Amount	Time (s)
5	15 983	14 466	185 959	1	10	41 088	43 244	873 317	4
15	68 183	85 454	3 471 960	17	30	106 167	183 570	16 151 068	65
60	143 989	377 635	38 063 145	3291	90	168 861	600 172	73 101 255	27 504

RF service is designed to function offline, at the end of each working day. In this setting even the larger times reported would be acceptable. Moreover, given that the service works on a daily basis, the realistic data sizes that Settlement-HYBRID is required to handle, are those of the previous experiment. Finally, the tested implementation is sequential. It can be improved by parallelizing the computation on the connected components of the input multigraph.

6 RELATED WORK

The MAX-PROFIT BALANCED SETTLEMENT problem that formalizes the receivable-settlement strategy behind the proposed network-based RF service is a novel one. To the best of our knowledge, no previous works adopt the same formulation, neither for receivable financing, nor for other applications. All the references that inspired our algorithmic solutions are already reported in Section 4 where appropriate. Here we mention for the interested reader a couple of problems that share some marginal similarity with our problem.

As testified by upper-bound derivation in the design of the exact algorithm, MAX-PROFIT BALANCED SETTLEMENT resembles a *network flow* problem [1]. Among the numerous variants of this problem, the one perhaps most related to ours is *Min-cost flow with minimum quantities* [13], which introduces a constraint for having minimum-flow quantities on the arcs of the network. In our context minimum quantities are required because a receivable can only be paid entirely. However, MAX-PROFIT BALANCED SETTLEMENT requires a different *conservation-flow* constraint (to properly handle cap and floor), and an additional constraint that each node in the solution has at least one incoming arc and one outgoing arc.

Another recent variant is the *Max-flow problem with disjunctive constraints* [12], which introduces binary constraints on using certain arc pairs in a solution. The problem is applied to event-participant arrangement optimization [14] in event-based social networks, such as *Meetup*. Compared to that problem, our MAX-PROFIT BALANCED SETTLEMENT requires different constraints on the arcs included in any feasible solution.

7 CONCLUSION

We have presented a novel, network-based approach to receivable financing. Our main contributions consist in a principled formulation and solution of such a novel service: we define and characterize a novel optimization problem on a network of receivables, and design both an exact algorithm and a more efficient algorithm to solve the problem. Experiments on real receivable data show that our algorithms work well in practice. In the future we plan to improve the methodology by incorporating *predictive* aspects, where recent history is considered instead of taking static decisions every day. We will also attempt to apply the lessons learned here to advance other financial services, e.g., receivable trading.

Acknowledgement. We are very thankful to S. Borghi, E. Cicco, S. Pascolutti, and L. G. Piccione for their valuable contribution to the design of the application scenario considered in this work.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc.
[2] A. Arulselvan. 2014. A note on the set union knapsack problem. *Discr. Appl. Math.* 169, Supplement C (2014), 214–218.
[3] V. Chvatal. 1979. A Greedy Heuristic for the Set-Covering Problem. *Math. Oper. Res.* 4, 3 (1979), 233–235.
[4] M. R. Garey and D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co.
[5] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis. 2011. D-cores: Measuring Collaboration of Directed Graphs Based on Degeneracy. In *IEEE ICDM.* 201–210.
[6] A.V. Goldberg and R.E. Tarjan. 1990. Finding minimum-cost circulations by successive approximation. *Math. Oper. Res.* 15 (1990), 430–466.
[7] O. Goldschmidt, D. Nehme, and Y. Gang. 1994. Note: On the set-union knapsack problem. *Nav. Res. Log.* 41 (1994), 833–842.
[8] K. A. Hawick and H. James. 2008. Enumerating Circuits and Loops in Graphs with Self-Arcs and Multiple-Arcs. In *FCS.* 14–20.
[9] D. S. Hochbaum. 1997. Approximation Algorithms for NP-hard Problems. PWS Publishing Co., Chapter Approximating Covering and Packing Problems: Set Cover, Vertex Cover, Independent Set, and Related Problems, 94–143.
[10] D. B. Johnson. 1975. Finding All the Elementary Circuits of a Directed Graph. *SIAM J. Comput.* 4, 1 (1975), 77–84.
[11] H. Kellerer, U. Pferschy, and D. Pisinger. 2004. *Knapsack problems.* Springer.
[12] U. Pferschy and J. Schauer. 2013. The maximum flow problem with disjunctive constraints. *J. Comb. Opt.* 26, 1 (2013), 109–119.
[13] H. G. Seedig. 2011. *Network Flow Optimization with Minimum Quantities.* Springer Berlin Heidelberg, 295–300.
[14] J. She, Y. Tong, L. Chen, and C. C. Cao. 2016. Conflict-Aware Event-Participant Arrangement and Its Variant for Online Setting. *TKDE* 28, 9 (2016), 2281–2295.