Network-based Receivable Financing

I. Bordino F. Gullo

UniCredit
R&D Department
Rome, Italy
{ilaria.bordino, francesco.gullo}@unicredit.eu

The 27th ACM International Conference on Information and Knowledge Management (CIKM 2018)
October 22-26, 2018
Turin, Italy

Application scenario: Traditional (client-server) receivable financing

- 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
  - e.g., **invoices**
- **Receivable Financing** (**RF**) is a service for creditors to fund cash flow by selling accounts receivables to a **funder** or **financing company**
  - Benefits for funder: **service fee**
  - Benefits for customers: **instant access to capital**, **no credit control**
- Existing funders adopt a **client-server** approach
  - **each request** for a receivable to be funded **is handled individually** by the funder

A novel, network-based approach to receivable financing

Major limitation of client-server receivable financing

It disregards the fact that receivables constitute a **network where the same customer may act as a creditor or a debtor** of different receivables

Proposal

A novel approach to receivable financing where a **network perspective** is profitably exploited **to trigger a money flow among customers themselves**

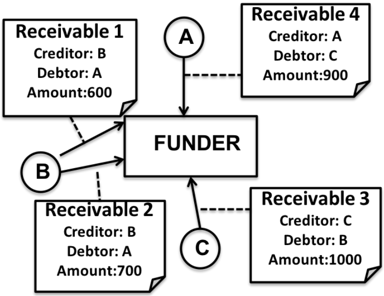
Pros for the funder:

- More liquidity
- Reduced risk of exposure

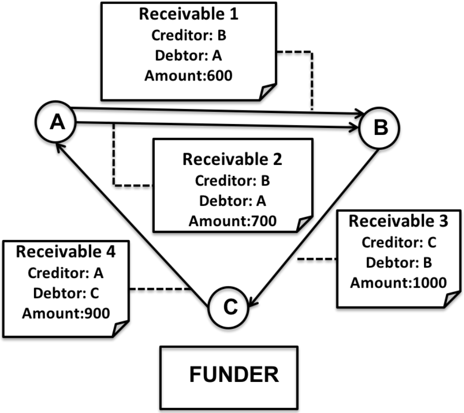
Pros for customers:

- Smaller fees
- Reduced time and effort in service establishment

A novel, network-based approach to receivable financing



(a) Client-server receivable financing



(b) Network-based receivable financing

Challenges and contributions

Main challenge

Given a **network of receivables**, identify a proper **subset of receivables** to be **settled** (i.e., for which the receivable-financing service is provided)

Contributions

- Formulation of network-based receivable settlement as a novel combinatorial-optimization problem
- Theoretical characterization of that problem
  - **NP**-hardness, bounds on the objective-function value of a set of solutions
- An exact branch-and-bound algorithm
- A more efficient algorithm
  - based on a relaxation of the original problem, and its theoretical characterization (**NP**-hardness and connection with KNAPSACK-like problems)
- A hybrid algorithm, as an ultimate proposal

Outline

- Introduction: motivation, challenges, contributions
- **Service overview**
- **Problem definition**
- **Algorithms**
  - **An exact algorithm**
  - **A more efficient algorithm**
  - **A hybrid algorithm**
- **Experiments**

Outline

- Introduction: motivation, challenges, contributions
- **Service overview**
- Problem definition
- Algorithms
  - An exact algorithm
  - A more efficient algorithm
  - A hybrid algorithm
- Experiments

Receivables

A **receivable** $R \in \mathcal{R}$ is an object with 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

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

Customers

Every **customer** $u \in \mathcal{U}$ is is assigned the following attributes:

- $bl_r(u) \in \mathbb{R}$: *receivable balance* of u's account
- $bl_a(u) \in \mathbb{R}$: *actual balance* of u's account
- $cap(u) \in \mathbb{R}$: 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

Network-based receivable financing in action

1. Creditor submits a receivable R, setting $life(R)$
2. System asks $debtor(R)$ for confirmation
3. R is added to the set $\mathcal{R}$ of current receivables
4. System attempts to settle R during the period $[insertdate(R), \min\{insertdate(R) + life(R), duedate(R)\}]$
5. If no settlement happens, the receivable is returned to the creditor; otherwise, $amount(R)$ is transferred from $debtor(R)$ to $creditor(R)$

Do-ut-des principle

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** $\rightarrow$ due to the constraint $bl_r(u) \leq cap(u)$

Outline

- Introduction: motivation, challenges, contributions
- Service overview
- **Problem definition**
- Algorithms
  - An exact algorithm
  - A more efficient algorithm
  - A hybrid algorithm
- Experiments

Input: active receivables and S-multigraph

- Receivable settlement works on a daily basis, running **offline at the end of any working day** t
- Input: set $\mathcal{R}(t)$ of **valid** receivables at time t
- $\mathcal{R}(t)$ describes a **directed, weighted, node-attributed multigraph**

Definition (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 Max-profit Balanced Settlement problem

- **Objective**: maximize the total amount of selected receivables
  - desirable for both funder and customers
- **Constraints**:
  (1) Consistency with *fl-cap* range: $bl_r(u) \leq cap(u)$, $bl_a(u) \geq fl(u)$
  (2) Selected customers should be both payers and payees $\rightarrow$ **strategic marketing choice**

Problem (Max-profit Balanced Settlement)

Given an S-multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, find a multisubset $\mathcal{E}^*$ of arcs so that

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

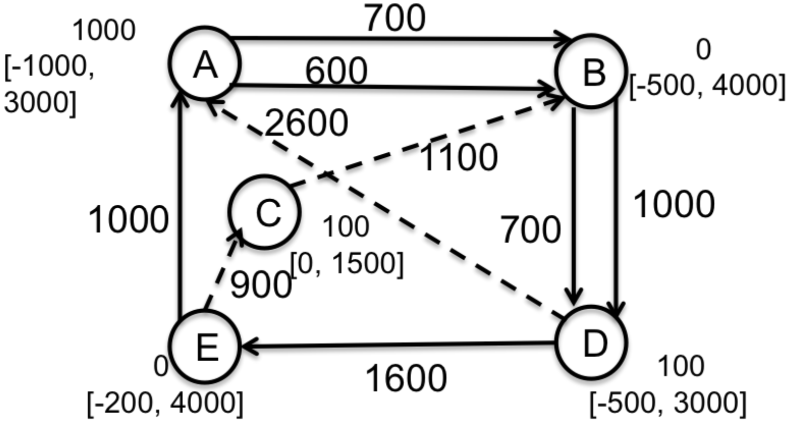
$$\left(\sum\nolimits_{(v,u) \in \hat{\mathcal{E}}} w(v,u) - \sum\nolimits_{(u,v) \in \hat{\mathcal{E}}} w(u,v)\right) \in [fl(u) - bl_a(u), cap(u) - bl_r(u)]\,, \tag{1}$$

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

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

Max-profit Balanced Settlement is **NP**-hard (reduction from Subset Sum)

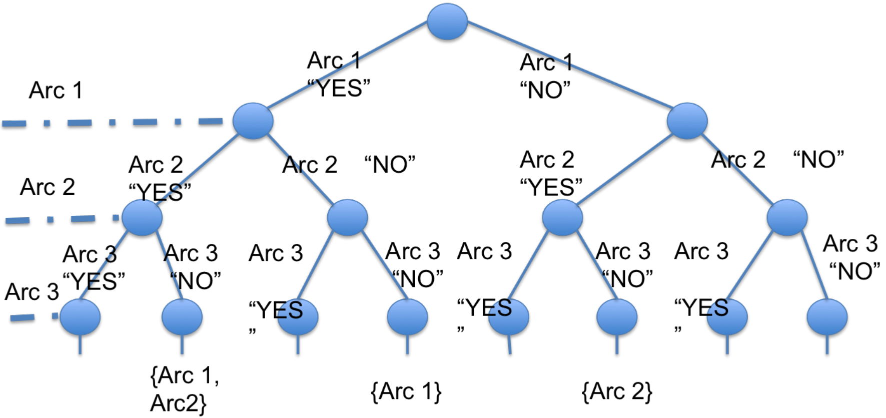
The Max-profit Balanced Settlement problem



Outline

- Introduction: motivation, challenges, contributions
- Service overview
- Problem definition
- **Algorithms**
  - **An exact algorithm**
  - A more efficient algorithm
  - A hybrid algorithm
- Experiments

The Settlement-BB algorithm: search space



- **Binary tree** $\mathcal{T}$ with $|\mathcal{E}|+1$ levels
- Levels (but the root) $\equiv$ arcs in $\mathcal{E}$
- Root-to-leaf paths $\equiv$ individual solutions $\hat{\mathcal{E}} \in 2^{\mathcal{E}}$
- Non-leaf tree node $\equiv$ set of solutions

The Settlement-BB algorithm: search-space exploration

Algorithm 1: Settlement-BB

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

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

- Standard branch-and-bound exploration
- **Crucial point**: definition of **lower bound** and **upper bound**

The Settlement-BB algorithm: lower bound

- For a tree-node X at level i of $\mathcal{T}$,
  $\mathcal{E}_X = \mathcal{E}_X^+ \cup \mathcal{E}_X^-$: **arcs for which a decision has been taken**
- **Lower bound** on the solutions spanned by X: **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** $\mathcal{E}_X^-$
- Find the set $\mathscr{C}$ of **cycles** of the multigraph induced by $\mathcal{E} \setminus \mathcal{E}_X^-$
- Greedily selects cycles based on their amount, as long as they meet the *fl-cap* problem constraints
  - other problem constraint always satisfied
  - cycle enumeration is a well-established problem (we use the classic Johnson's algorithm)
- Time complexity
  - dominated by cycle enumeration $\Rightarrow$ we look for cycle up to length L
  - the rest takes $\mathcal{O}(L\,|\mathscr{C}|\log|\mathscr{C}|)$ time

The Settlement-BB algorithm: lower-bound

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 $\hat{\mathcal{E}} \subseteq \mathcal{E} \setminus \mathcal{E}_X^-$

$\mathscr{C} \leftarrow$ cycles of multigraph $\mathcal{G}^- = (\mathcal{V}, \mathcal{E} \setminus \mathcal{E}_X^-, w)$ {Johnson's algorithm}
$\hat{\mathcal{E}} \leftarrow \emptyset$, $\hat{\mathscr{C}} \leftarrow \emptyset$
while $\mathscr{C} \neq \emptyset \wedge \mathcal{E}_X^+ \nsubseteq \hat{\mathcal{E}}$ **do**
  $\mathscr{C} \leftarrow \{C \in \mathscr{C} \mid \hat{\mathcal{E}} \cup C$ meets Constraint (1) of Max-profit Balanced Settlement$\}$
  $C \leftarrow$ cycle in $\mathscr{C}$ minimizing $[|C \cap (\mathcal{E}_X^+ \setminus \hat{\mathcal{E}})| \times \sum_{e \in C \setminus \hat{\mathcal{E}}} w(e)]^{-1}$
  $\hat{\mathscr{C}} \leftarrow \hat{\mathscr{C}} \cup \{C\}$, $\mathscr{C} \leftarrow \mathscr{C} \setminus \{C\}$, $\hat{\mathcal{E}} \leftarrow \hat{\mathcal{E}} \cup C$
while $\mathscr{C} \neq \emptyset$ **do**
  $\mathscr{C} \leftarrow \{C \in \mathscr{C} \mid \hat{\mathcal{E}} \cup C$ meets Constraint (1) of Max-profit Balanced Settlement$\}$
  $C \leftarrow$ cycle in $\mathscr{C}$ maximizing $\sum_{e \in C \setminus \hat{\mathcal{E}}} w(e)$
  $\mathscr{C} \leftarrow \mathscr{C} \setminus \{C\}$, $\hat{\mathcal{E}} \leftarrow \hat{\mathcal{E}} \cup C$
if $\mathcal{E}_X^+ \nsubseteq \hat{\mathcal{E}}$ **then** $\hat{\mathcal{E}} \leftarrow \emptyset$

The Settlement-BB algorithm: upper bound

Relaxation of MAX-PROFIT BALANCED SETTLEMENT where

- Constraint (2) is discarded
- Arcs are allowed to be selected **fractionally**

Problem (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*

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

The desired upper bound relies on an interesting characterization of the RELAXED SETTLEMENT problem as a **network-flow problem**:

> **Solving Relaxed Settlement on multigraph $\mathcal{G}$ is equivalent to solving MIN-COST FLOW on a modified version of $\mathcal{G}$**

The Settlement-BB algorithm: upper bound

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 Theorem 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

- We solve Min-Cost Flow with the well-established Cost Scaling algorithm (*Goldberg and Tarjan, Math. Oper. Res., 1990*)
  - $\mathcal{O}(|\mathcal{E}|\,(|\mathcal{V}| \log |\mathcal{V}|) \log(|\mathcal{V}|\, w_{max}))$ time complexity, where $w_{max} = \max_{e \in \mathcal{E}} w(e)$

The Settlement-BB algorithm: upper bound

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

Definition (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 $w_f(\tilde{t}, \tilde{s}) = \infty$.

The Settlement-BB algorithm: upper bound

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 $\mathcal{G}$.

Corollary

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.*

Outline

- Introduction: motivation, challenges, contributions
- Service overview
- Problem definition
- **Algorithms**
  - An exact algorithm
  - **A more efficient algorithm**
  - A hybrid algorithm
- Experiments

The Settlement-BEAM algorithm

- **Main idea**: enumerating cycles and properly selecting a subset
- We formulate the OPTIMAL CYCLE SELECTION problem
  - find a subset of cycles exhibiting the maximum total amount and satisfying the constraints of MAX-PROFIT BALANCED SETTLEMENT
- We theoretically characterize OPTIMAL CYCLE SELECTION
  - **NP**-hardness (reduction from MAXIMUM INDEPENDENT SET)
  - connection with KNAPSACK-like problems (i.e., SET UNION KNAPSACK)
- We devise our Settlement-BEAM inspired by the well-established Aruselvan's algorithm for SET UNION KNAPSACK
  - arcs $\equiv$ elements, cycles $\equiv$ items
  - extension to handle MULTIDIMENSIONAL SET UNION KNAPSACK
  - coupling it with a **beam-search** methodology

The Settlement-BEAM algorithm

Algorithm 4: Settlement-BEAM

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

$\mathcal{E}^* \leftarrow \emptyset$
$\mathscr{C} \leftarrow$ cycles of $\mathcal{G}$
while $\mathscr{C} \neq \emptyset$ **do**
  $\mathscr{C}' \leftarrow K$-sized subset of $\mathscr{C}$ by Greedy MAX COVER
  $\mathscr{C}'_2 \leftarrow \{\{C_i, C_j\} \mid C_i, C_j \in \mathscr{C}'\}$
  for all $\{C_i, C_j\} \in \mathscr{C}'_2$ **do**
    $C_{ij} \leftarrow C_i \cup C_j$
    process all $C \in \mathscr{C}' \setminus \{C_i, C_j\}$ 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 OPTIMAL CYCLE SELECTION
  $\mathcal{E}^* \leftarrow \mathcal{E}^* \cup \arg\max_{C_{ij} \in \mathscr{C}'_2} \sum_{e \in C_{ij}} w(e)$
  $\mathscr{C} \leftarrow \mathscr{C} \setminus (\mathscr{C}' \cup \{C \in \mathscr{C} \mid C \cap \mathcal{E}^* = C\})$

- $$\omega(C) = \frac{\sum_{e \in C} w(e)}{\sum_{e \in C} \frac{w(e)}{f(e)}}, \text{ where } f(e) = |\{C \in \mathscr{C} : e \in C\}| \quad (3)$$
- $\mathcal{O}(L\, K^2\, |\mathscr{C}|)$ time complexity

Outline

- Introduction: motivation, challenges, contributions
- Service overview
- Problem definition
- **Algorithms**
  - An exact algorithm
  - A more efficient algorithm
  - **A hybrid algorithm**
- Experiments

The Settlement-HYBRID algorithm

- Run Settlement-BB on the smaller connected components
- Run Settlement-BEAM on the larger connected components

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}

Implementation details:

- Extract the $(1,1)$-*D-core* of the input S-multigraph beforehand
- Tree-like search space of Settlement-BB:
  - sort arcs by non-decreasing amount
  - DFS vs. BFS: no evident difference

Outline

- Introduction: motivation, challenges, contributions
- Service overview
- Problem definition
- Algorithms
  - An exact algorithm
  - A more efficient algorithm
  - A hybrid algorithm
- **Experiments**

Experimental evaluation: settings

- Random sample of a **real dataset** provided by UniCredit, a noteworthy European banking company
  - 5 413 375 receivables; 369 479 (anonymized) customers; 1 year in 2015-16
- Customers' attributes set based on statistics computed on a training prefix of 3 months of data
- $fl(u) = 0$, for each customer u
- 6 simulation settings:
  - $cap < \infty$ vs. $cap = \infty$
  - "worst", "normal", "best" scenarios (defined by invoice lifetime and *cap*)
- $L = 15$ (all algorithms based on cycle enumeration), $H = 20$ (Settlement-BEAM), and $K = 1\,000$ (Settlement-BEAM and Settlement-HYBRID)

Experimental evaluation: general performance

CAP	Scenario	Period	Settlement-bb-lb Amount	Time (s)	Receivables	Clients
∞	normal	20150701-0930	553 364 544	64.58	7131	2836
		20151001-1231	643 722 123	6.67	6742	2736
		20160101-0331	693 852 990	29.03	7999	3034
		20160401-0630	751 368 135	30.81	8289	3189

CAP	Scenario	Period	Settlement-beam Amount	Time (s)	Receivables	Clients
∞	normal	20150701-0930	660 907 304	1168.95	15 323	3761
		20151001-1231	663 873 349	618.98	14 570	3507
		20160101-0331	743 945 529	1159.27	17 390	4143
		20160401-0630	855 932 063	757.85	17 666	4155

CAP	Scenario	Period	Settlement-hybrid Amount	%Gain vs. S-bb-lb	%Gain vs. S-beam	Time (s)	Receivables	Clients
∞	normal	20150701-0930	**779733K**	40.91	17.98	1006.17	17 082	4268
		20151001-1231	**784315K**	21.84	18.14	690.14	16 761	4133
		20160101-0331	**827346K**	19.24	11.21	1329.80	19 544	4701
		20160401-0630	**987866K**	31.48	15.41	865.92	19 576	4718

Experimental evaluation: scalability

Table: Scalability of the proposed Settlement-HYBRID algorithm

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

Conclusion

- We introduce a novel, network-based approach to receivable financing
- We provide 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
- Experiments on real receivable data show that our algorithms work well in practice

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

Thanks!

The MIN-COST FLOW problem

Problem (MIN-COST FLOW)

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*

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

The Set Union Knapsack problem

Problem (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 $q : U \rightarrow \mathbb{R}$ 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

$\mathcal{S}^* = \arg\max_{\hat{\mathcal{S}} \subseteq \mathcal{S}} P(\hat{\mathcal{S}}) \;\; s.t. \;\; Q(\hat{\mathcal{S}}) \leq B.$

The Multidimensional Set Union Knapsack problem

Problem (Multidimensional Set Union Knapsack)

Given U, $\mathcal{S}$, p as in Set Union Knapsack, *a d-dimensional cost function $q : U \rightarrow \mathbb{R}^d$, 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.* $\mathbf{Q}(\hat{\mathcal{S}}) \leq \mathbf{B}$, *where* $\mathbf{Q}(\hat{\mathcal{S}}) = \sum_{x \in U(\hat{\mathcal{S}})} q(x)$.