Heuristic methods for mutual debt reduction on B2B networks

L3 MPCI
November 22nd - December 17th 2021

Marie VELA-MENA

Internship supervisor : Nazim FATÈS
Institut national de recherche en sciences et technologies du numérique
(INRIA Nancy Grand-Est)

Acknowledgements

I would like to thank Nazim Fatès, my internship tutor, for his support during the whole internship and for his availability to answer all my questions. He gave me a lot of knowledge through all the discussions we had.

I thank Sylvain Contassot-Vivier for his availability to proofread my report and for the comments he gave me about my work.

I also thank Olivier Buffet, Irène Marcovici and Simon Perdrix for their remarks during the different presentations they were able to attend.

I would also like to thank the MOCQUA team, and more generally all the people of the laboratory that I could meet or with whom I could discuss during this internship, for their kindness and their warm welcome.

Contents

Introduction

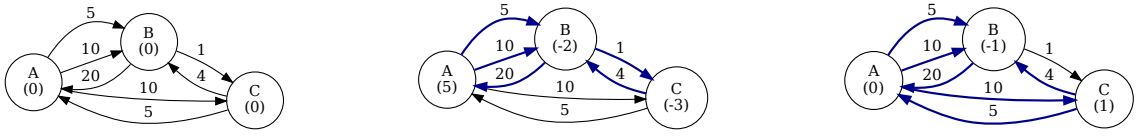
Nowadays, invoices of companies are usually settled after a certain amount of time. Indeed, to pay their invoices immediately, companies can use their available liquidity. However, some companies do not have enough liquidity and depend on other companies. They are waiting to be paid by companies to pay their own bills. So, some companies can be in difficulty during this period, difficulty which can lead to bankruptcy. Moreover, the accumulation of debts on global scale creates a systemic risk, which can lead to serious economic incidents [1].

To reduce this debt, a new method of financing companies was developed [2] by analyzing a payment network. A *payment network* is a set of interactions between economic actors. Here, the economic actors are companies, and the interactions take the form of debt and credit relationships. These relationships are invoices which can be settled by a payment.

We model this economic system as a graph: nodes are companies, and edges are invoices. For an edge, the source is the invoice issuer, the destination the invoice receiver and the weight the invoice amount. This graph representation has been the subject of recent studies such as the Sardex monetary network [3]. This system considers debts and credits as a form of money, and companies can, with a positive balance, buy with this available balance instead of real money.

This new method of financing companies is called the *integral mutual debts compensation*. It consists of injecting a sum of money from a financial center (bank, state) in order to reduce as much as possible, or even to eliminate the total debt. Companies involved make no profit or loss. The term *integral* here means that an invoice can only be settled entirely.

For example, given three companies A, B, C with invoices in euros, owed to each other for services invoiced but not yet paid (see Figure 1a). A solution of invoice settlement is where the blue invoices are settled (see Figures 1c and 1b). In the solution presented in Figure 1b, companies B and C has respectively a deficit of 2€ and 3€. So, by financing 2€ to B and 3€ to C, 40€ of invoices can be settled. In the solution of the Figure 1c, the company B has a deficit of 1€. So, by loaning 1€ to B, 54€ of invoices can be settled in this solution. Moreover, the company C ends with a net position positive of 1€. An invoice of 1€ from B to C is not settled.



(a) Initial graph

(b) Graph after integral mutual debts compensation

(c) Graph after another integral mutual debts compensation

Figure 1: Example of an integral mutual debts compensation: invoices in black are not yet settled, invoices in blue are settled, net position of the nodes in parentheses

Several solutions exist to settle invoices. The integral mutual debts compensation problem [2] consists of finding a subset of invoices that attains the best compromise between:

- maximizing the amount of debts contained in a given subset of invoices,
- minimizing the supply of liquidity required to clear this debt.

Furthermore, we work on bigger graphs than these shown in the example and this problem is NP-complete: some instances of the problem are equivalent to solve the Knapsack problem [4]. The challenge of this internship is to find a good way to choose these invoices from the multitude of possible choices.

Given a percentage of the amount of debts to clear, the objective of this internship was to develop heuristics to minimize the supply of liquidity required. It means that we had to first elaborate heuristics to reach this amount of debts to clear. Then, heuristics which minimize the supply of liquidity required were developed.

To give an account of my internship experience, I will first present the context of the internship. After that, I will present heuristics which I have developed to reach a percentage of amount of debts to clear. These heuristics allow us to find a solution where the supply of liquidity is acceptable. Then, I will present the heuristics that help to minimize the supply of liquidity required given a solution previously obtained.

1 Background

1.1 Lab presentation

The Lorraine Research Laboratory in Computer Science and its Applications (LORIA) is a research unit common to INRIA, CNRS and the University of Lorraine. Its research activities are divided into five departments: Algorithms, Computation, Image and Geometry; Formal methods; Networks, Systems and Services; Natural language processing and Knowledge discovery; Complex systems, Artificial intelligence and Robotics.

The National Institute for Research in Digital Science and Technology (INRIA) is a research institution specialized in applied mathematics and computer science[1].

This internship took place in the MOCQUA team in the Formal methods department[2]. The goal of this team is to face the challenges which come from the emergence of new or future computational models.

Some of the topics explored during this internship can be connected with the team and the department. Indeed, the Formal methods department deals with the fundamental aspects of complexity and our problem is NP-complete. Dynamical systems are one of the main subjects of research of the team and of my internship supervisor. And the evolution of a configuration during simulated annealing used in my work can be seen as a dynamical system.

1.2 Definitions

To understand the integral mutual debts compensation problem, it is necessary to understand directed multi-graphs and to define some vocabulary.

1.2.1 Directed multi-graphs

A multi-digraph $G = (V, E)$ is a directed graph in which several directed edges may connect two given nodes.

It is defined by a set of nodes V, with $|V| = n$ and a multi-set of edges E, with $|E| = m$. An edge is a triplet in the form $u = (A, B, w)$, where $(A, B) \in V^2$ are respectively source and destination of the edge, and $w \in \mathbb{N}^*$ its weight. We denote by w_u the weight of the edge $u \in E$. We choose to not consider edges with a null weight.

Let $A \in V$ be a node. The incoming edges of A are $I(A) = \{u \in E | u = (., A, .)\}$ and the outgoing edges of A are $O(A) = \{u \in E | u = (A, .,)\}$.

1.2.2 Integral mutual debts compensation problem

Let us now present more formally our problem [2].

A configuration $\mathcal{C} \subset E$ is a set of settled invoices. It is the form of a solution of the problem. An edge added to a configuration $\mathcal{C}$ is described as a settled invoice.

The *settlement* $\Sigma(\mathcal{C})$ of a configuration $\mathcal{C}$ is the total value of debts included in the configuration. This is the sum of the weights of the edges in $\mathcal{C}$:

$$\Sigma(\mathcal{C}) = \sum_{u \in C} w_u.$$

Let $\mathcal{C}$ be a configuration, the *net position* $\pi_C(A)$ over $\mathcal{C}$ of a node $A \in V$ is the difference between the flow which enters A and the flow which leaves A:

$$\pi_C(A) = \sum_{u \in I(A) \cap \mathcal{C}} w_u - \sum_{v \in O(A) \cap \mathcal{C}} w_v.$$

The nodes which are not in $\mathcal{C}$ have a zero net position.

The *financing* $\Phi(\mathcal{C})$ of a configuration $\mathcal{C}$ is the quantity of liquidity that needs to be provided to clear the debt contained in $\mathcal{C}$. It is the sum of the negative net positions over $\mathcal{C}$: $\Phi(\mathcal{C}) = \sum_{A \in V} \max(0, -\pi_C(A))$.

Given $\mathcal{C}$ a configuration, the *amplification factor* $\alpha(\mathcal{C})$ is the ratio of the settlement to the financing: $\alpha(\mathcal{C}) = \Sigma(\mathcal{C})/\Phi(\mathcal{C})$. We denote that $\alpha(\emptyset) = 0$. The amplification factor represents the quality of a configuration.

The *inclusion factor* $\iota(\mathcal{C})$ of a configuration $\mathcal{C}$ is the ratio of the settlement to the total value of debts in the graph: $\iota(\mathcal{C}) = \Sigma(\mathcal{C})/\Sigma(E)$.

[1] `www.inria.fr/fr/centre-inria-nancy-grand-est`

[2] `www.loria.fr/fr/la-recherche/les-equipes/mocqua/`

The interval that interested us for the inclusion factor is $[0.25, 0.5]$. Economically, this interval is more interesting. Some research has already been done [2]. A graphic of the amplification factor as a function of the inclusion factor exists which we can rely on (cf. Figure 2). Given an inclusion factor, an acceptable amplification factor for a configuration is close (approx. 0.2 of difference for a smaller value) or greater than the value in the graphic. This graphic tells us that the amplification factor decreases with the increase of the inclusion factor. Therefore, for heuristics which increase the inclusion factor, the challenge is that the amplification factor does not decrease too much with the increase of the inclusion factor.

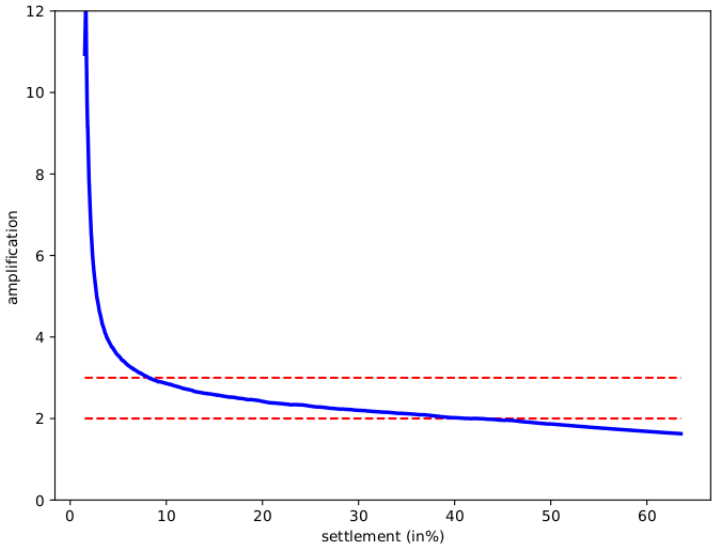


Figure 2: Evolution of the amplification factor as a function of the inclusion factor (settlement). The two red lines highlight the amplification factor values 2 and 3. [2]

1.3 Test graphs presentation

For tests, different graphs were used. We used real data and generated data.

These real data were provided by the Infocert company, which does electronic invoicing. It is a set of invoices between Italian companies. The graph `graphPME` is made up of invoices between small and medium-sized enterprises, which are Infocert clients, during January 2019.

Generated data were used too. Tests graphs were generated by another intern, Arthur ROUSSEAU, to model the economy [5]. The algorithm of graph generation is based on Markov chains. The distribution law for the weights of the edges follows a power law.

Some statistics of the graphs used for tests are presented in Table 1.

Graph	Number of nodes	Number of edges	Average weight of edges (in euro cents)
Gen_100	96	333	114 159
Gen_1000	868	3 325	123 633
Gen_5000	4 137	16 680	121 370
Gen_20000	16 244	66 382	114 629
Gen_40000	32 808	133 497	116 470
graphPME	38 334	145 540	117 676

Table 1: Number of nodes and edges for each graph used for tests

1.4 Heuristics implementation

During this internship, the language for the computer programming was Python 3.8.

All heuristics, which will be presented in this internship report, were implemented [6]. We choose to code them by using classes.

The entities classes which represent nodes, edges and graphs are inherited from previous classes developed by another intern Yosyp MYKHAILIV. A heritage was used to separate our works and the methods and functions which were added. The other classes do not have any common point with his work.

To ease the use of these heuristics, a main function with a configuration file was created. This configuration file contain variables which are read by the function. The file is a `.env` file and we use `dotenv` [7] and `os` libraries from Python to read it.

2 Heuristics to increase the inclusion factor

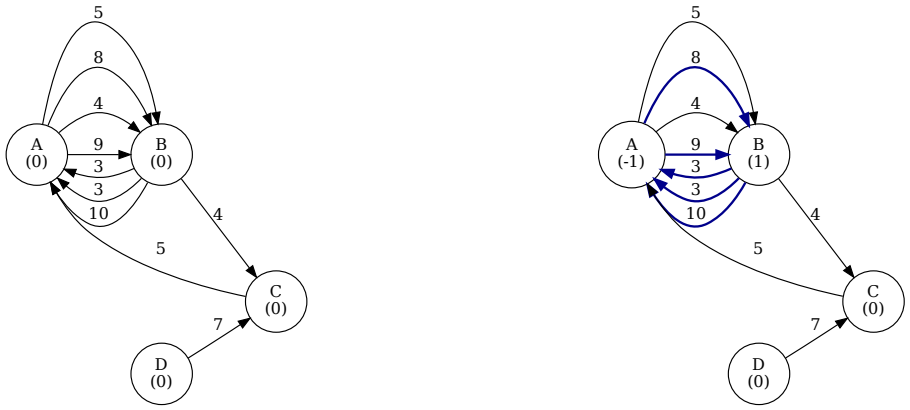
During this internship, heuristics to increase the inclusion factor were developed. These heuristics were developed to find a configuration which has an acceptable amplification factor. The combination of those heuristics allows us to have a configuration which reaches a desired inclusion factor with an acceptable amplification factor.

2.1 Bilateral settlement

We define the *bilateral settlement heuristic* as a heuristic that searches for bilateral settlement to reduce debts. This heuristic is applied on a graph where the configuration is empty. It is used to start the configuration with a high amplification factor, to then propagate this configuration.

A configuration $\mathcal{C}_{RB}$ is a *bilateral settlement* if there exists $A, B \in V$, $A \neq B$ such as $\mathcal{C}_{RB} \subseteq (I(A) \cap O(B)) \cup (I(B) \cap O(A))$, $\mathcal{C}_{RB} \cap I(A) \cap O(B) \neq \emptyset$, and $\mathcal{C}_{RB} \cap I(B) \cap O(A) \neq \emptyset$. We say that $\mathcal{C}_{RB}$ is a bilateral settlement between A and B.

For example, we have a simple graph G with companies A, B, C, D (see Figure 3a) and debts in euros, a bilateral settlement between A and B can be found. By loaning of 1€ to A, 33€ of invoices can be settled (cf. Figure 3b).



(a) Initial graph

(b) Graph with a bilateral settlement

Figure 3: Example of a graph with a bilateral settlement: invoices in black are not in the configuration, invoices in blue are in the configurations, net position of the nodes in parentheses

We define $\alpha_{\min}$ as the minimal amplification factor for a bilateral settlement to be considered. Indeed, some bilateral settlements can be unbalanced, with a low amplification factor. The condition on the amplification factor provokes a smaller decrease with the inclusion factor than if we had no condition at all.

The problem is: given a graph G and an admission parameter $\alpha_{\min} \in \mathbb{R}^+$, determine the best eligible bilateral settlement for each bilateral settlement.

$\mathcal{C}_{RB}$ is the best eligible bilateral settlement between A and B nodes of V if $\alpha(\mathcal{C}_{RB}) \geq \alpha_{\min}$ and for all $\mathcal{C}'_{RB}$ between A and B such as $\alpha(\mathcal{C}'_{RB}) \geq \alpha_{\min}$, we have $\iota(\mathcal{C}_{RB}) \geq \iota(\mathcal{C}'_{RB})$.

Finding the best admissible bilateral settlement between two companies is an NP-complete problem. Plus, for n edges between two nodes, the number of combinations is approximately 2^n. In Appendix A.1 we present the number of combinations and how we limit this number.

2.1.1 Searching for bilateral settlements algorithm

The searching for bilateral settlements algorithm involves looking at the best admissible bilateral settlement for each pair of nodes which are strongly connected. In Appendix A.2, the algorithm and the way chosen to limit combinations are presented.

For each combinations of bilateral settlement between two given nodes, the amplification factor and the inclusion factor are computed. To avoid an error of overused memory during the study of all combinations, a generator function was implemented (see Appendix A.3).

2.1.2 Invoice settlement

Once all the best admissible bilateral settlements are found, they are added to the configuration.

After several experiments, the inclusion factor of the configuration with all the best admissible bilateral settlements is small: less than 0.02. Bilateral settlements are small in number among the graph. So keeping a high amplification factor is quite important for the next steps.

After that, several tests to determine the best $\alpha_{\min}$ given a graph G were done. For example, on the graph `graphPME`, the value of $\alpha_{\min}$ can be taken in the interval $[3, 5]$. Indeed, the inclusion factor is in the interval $[0.01, 0.02]$ and the final amplification factor is above 5 (see Figure 4). After the first step, the amplification factor needs to be high because for each step it decreases (cf. Figure 2).

The variations on curves in the graphic are due to adding edges. Indeed, for each edge added to the configuration, new inclusion and amplification factors are computed and this information is then written in a file. Given a bilateral settlement between A and B two nodes, the outgoing edges of A are first added, and then the incoming edges of A. So there is an imbalance during the addition of a bilateral settlement. These variations are greater when the amplification factor is high.

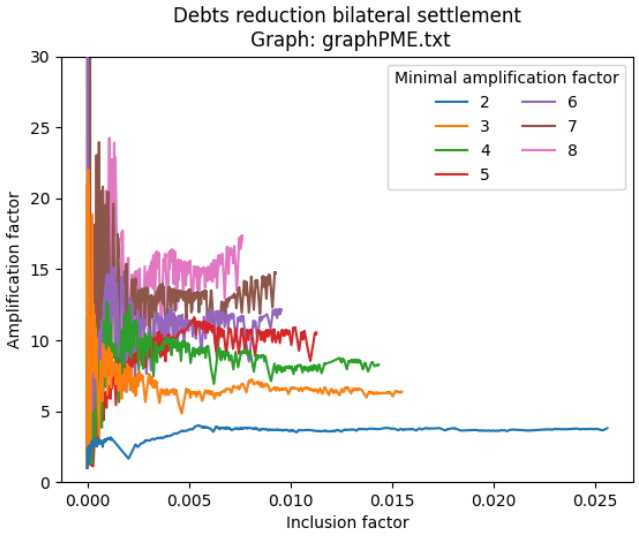


Figure 4: Amplification factor as a function of the inclusion factor during an invoice settlement on bilateral settlements for different $\alpha_{\min}$ and a $n_{\max} = 20$

Consequently, this heuristic is useful to start a configuration. Even if the inclusion factor is very low. In fact, at the end of this step, the amplification factor is high (greater than 5) and it is important for the next steps.

Open question : In our case, each bilateral settlement is taken alone. It means that a node can be part of several bilateral settlements. We choose that the net positions of nodes are not taken into account. If the debts are reduced in a certain order, some bilateral settlements can be admissible. What would be the inclusion factor and the amplification factor, if an order on finding and financing bilateral settlements was applied ?

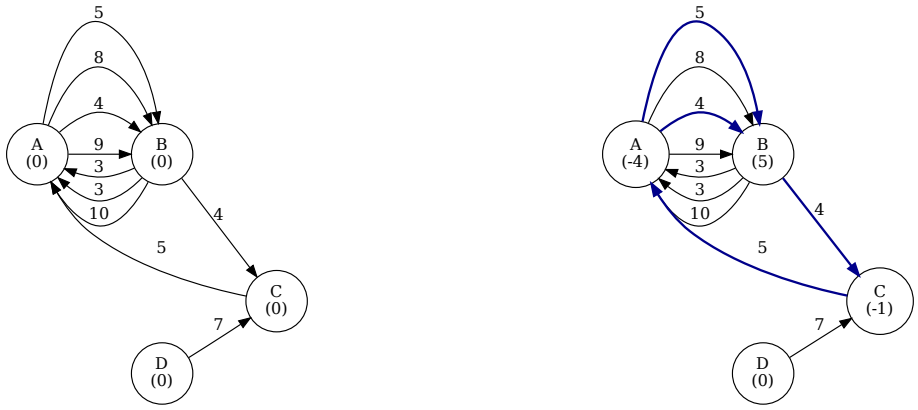
Bilateral settlements are not enough to have a configuration with a high inclusion factor. However, after this step we have a good quality configuration. For the next step, we developed a heuristic based on paths.

2.2 Paths

The *paths heuristic* is a heuristic that searches for paths in the graph to reduce debts along them. It is useful to increase the inclusion factor.

We define a *path* as an ordered finite sequence of edges of E. Two consecutive edges share a same node and an edge cannot be present twice or more in a path. Moreover, by following the direction of the edges, we can visit all the edges of the path. A path of length $s_{\max}$ is modeled by $(u_n)_{n\in\mathcal{N}}$ where $\mathcal{N} = \{0, 1, \ldots, s_{\max} - 1\}$. Given $n \in \mathcal{N} \setminus \{s_{\max} - 1\}$, if $u_n = (A, B, w_{u_n})$, $A \neq B$, then $u_{n+1} \in O(B)$. Furthermore $\forall i, j \in [\![0, s_{\max} - 1]\!], u_i \neq u_j$.

For example, we have a graph G with companies A, B, C, D (see Figure 5a). A path can be found from A to B, B to C, C to A and A to B (cf. Figure 5b).



(a) Initial graph

(b) Graph with a path: $u_0 = (A, B, 4)$, $u_1 = (B, C, 4)$, $u_2 = (C, A, 5)$, $u_3 = (A, B, 5)$

Figure 5: Example of a graph with a path: invoices in black are not in the configuration, invoices in blue are in the configuration, net position of the nodes in parentheses

We define $s_{\min} \geq 2$ as the minimal number of edges in a path.

The problem is: given a graph G and, admissions parameters $s_{\min}$ and the condition on the weight of two consecutive edges, determine a set of admissible paths.

An *admissible path* meets both conditions: the length of the path is above $s_{\min}$ and the weight of an edge is close to the weight of the consecutive edge. To measure the last condition, there are two possibilities:

- the ratio r with the weights: this ratio is calculated in way such that it is less than 1. For example, we consider two edges $u_1 = (A, B, w_{u_1})$ and $u_2 = (B, C, w_{u_2})$ where $A, B, C \in V$, $w_{u_1}, w_{u_2} \in \mathbb{N}^*$. The ratio r is: if $w_{u_1} >= w_{u_2}$, $r = w_{u_2}/w_{u_1}$ else $r = w_{u_1}/w_{u_2}$. The parameter $r_{\min} \in]0, 1]$ is the threshold value of the ratio for the edges to be in the path, i.e. $r \geq r_{\min}$.
- the absolute difference d of the weights. For example, we consider two consecutive edges $u_1 = (A, B, w_{u_1})$ and $u_2 = (B, C, w_{u_2})$ where $A, B, C \in V$ and $w_{u_1}, w_{u_2} \in \mathbb{N}^*$. The difference d is: $d = |w_{u_1} - w_{u_2}|$. The parameter $d_{\max} \in \mathbb{N}$ is the limit value of the difference for the edges to be in the path, i.e. $d \leq d_{\max}$.

2.2.1 Searching for paths algorithm

The searching for paths algorithm looks for an admissible path for each node of the graph G. An example algorithm is presented in Appendix A.4.

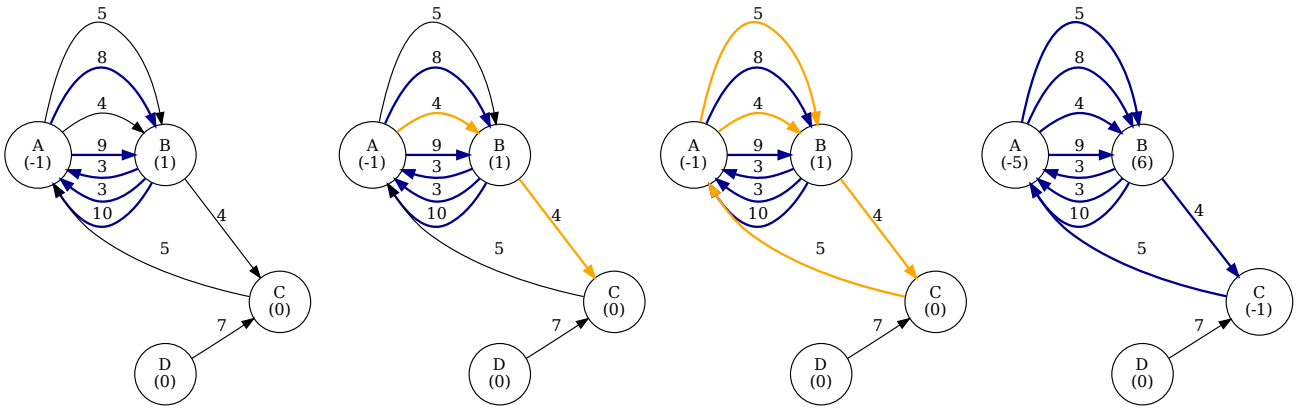
Let $r : E^2 \to]0, 1]$ be the ratio of the weights of the two edges given in an argument. Let $d : E^2 \to \mathbb{R}^+$ be the absolute difference of the weight of the two edges given in an argument.

We start from an initial node H. To choose the first two edges, the weight of each incoming edge is compared to that of each outgoing edge of H. Let $O'(H) \subset O(H)$ (respectively $I'(H) \subset I(H)$) be the set where the edges of $O(H)$ (resp. $I(H)$) in a path or in the configuration are not in $O'(H)$ (resp. $I'(H)$). The first two edges $u \in I'(H)$ and $v \in O'(H)$ check: if the ratio is the criterion, $\forall p \in I'(H) \forall q \in O'(H),\ r(u, v) \geq r(p, q)$; if the absolute difference is the criterion $\forall p \in I'(H) \forall q \in O'(H),\ d(u, v) \leq d(p, q)$. We choose an incoming and an outgoing edge of the initial node instead of only one edge because two edges are already a path. All that's left to do is to extend it so that this path is admissible if $s_{\min} > 2$.

When we move forward, we give J the last node and u the last edge chosen where J is its destination. Let $O'(J) \subset O(J)$ be the set where the edges of $O(J)$ in a path or in the configuration are not in $O'(J)$. The next edge $v \in O'(J)$ to choose for the path checks: if the ratio is the criterion, $\forall t \in O'(J),\ r(u, v) \geq r(u, t)$; if the absolute difference is the criterion $\forall t \in O'(J),\ d(u, v) \leq d(u, t)$.

When we move backward, we give u the last edge chosen where K is its source node i.e. $u \in O(K)$. Let $I'(K) \subset I(K)$ be the set where the edges of $I(K)$ in a founded path or in the configuration are not in $I'(K)$. The next edge $v \in I'(K)$ to choose for the path checks: if the ratio is the criterion $\forall t \in I'(K),\ r(v, u) \geq r(t, u)$; if the absolute difference is the criterion $\forall t \in I'(K),\ d(v, u) \leq d(t, u)$.

For example, we search for a path to complete the configuration obtained previously (cf. Figure 6a) where $s_{\min} = 3$. We start from the initial node B and we add an incoming and an outgoing edge of B (see Figure 6b). Then outgoing edges are added to the path i.e. we move forward from the initial node: first an edge from C to A, and then an edge from A to B (cf. Figure 6c). We are now on node B, but there are no edges available, so the path is complete. In the case where there is any other outgoing edge of the current node, we then look for incoming edges by going backwards from the initial node.



(a) Graph with a configuration obtained with the 'bilateral settlement' heuristic

(b) Beginning of a path at B

(c) Continuation of the path

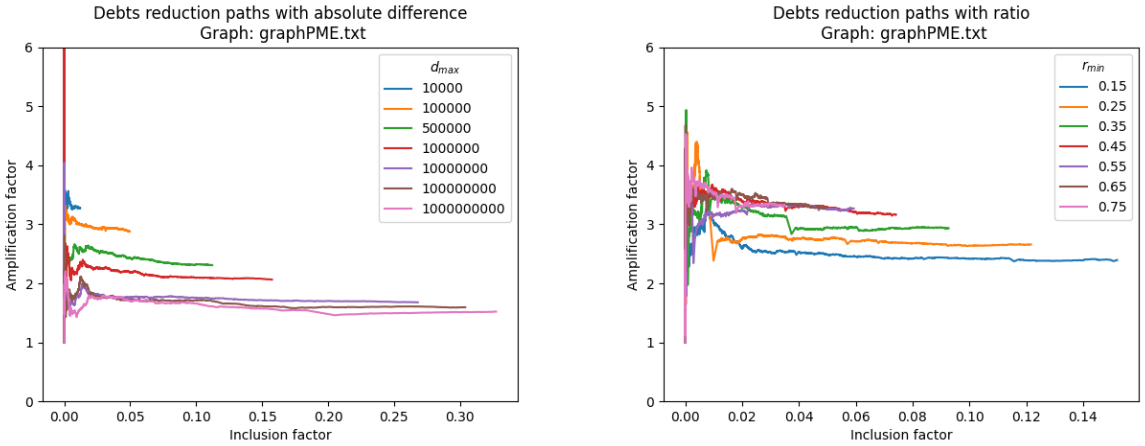
(d) Adding path to the configuration

Figure 6: Example of a search for a path: edges in black are not in the configuration, edges in blue are in the configurations, edges in orange are the path but not yet in the configuration, net position of the nodes in parentheses

2.2.2 Invoice settlement

We can compare the effect of the parameter that concerns the weight of two consecutive edges. We start from an empty configuration.

The amplification factor as a function of the inclusion factor during an invoice settlement on paths found in `graphPME` is represented (cf. Figures 7a and 7b). First, we take a look at the amplification factor. The final amplification factor reached with the ratio criteria is greater than 2 for all amplification factors tested. However, that's not the case for the criteria of absolute difference. Then we compare the amplification factor given an inclusion factor of 0.15: it is higher for paths found with ratio criteria than difference criteria.



(a) Absolute difference of weights

(b) Ratio of weights

Figure 7: Amplification factor as a function of the inclusion factor during an invoice settlement on paths of minimal length $s_{\min} = 3$

After that, we made several tests to determine which parameters to take. For the graph `graphPME`, the value of the minimal length of the path is $s_{\min} = 3$. And the heuristic to choose an edge given the weight of the previous one is the one that computes the ratio of the weights. Indeed, for a given final inclusion factor, the amplification factor is higher with this heuristic.

During the elaboration of this heuristic, various questions were raised. An hypothesis was that bilateral settlements could break some paths. However it is not the case, and searching for bilateral settlements before paths makes higher final inclusion and amplification factors after these two steps. So it is better to apply the 'bilateral settlement' heuristic before the one with paths. Another question was: what if we apply the 'paths' heuristic on a given configuration twice in a row. To answer this question, we have applied the 'paths' heuristic twice in a row on a configuration with bilateral settlements. The second time, the inclusion factor raised only of 0.01 and the amplification factor decreased. The gain is too poor to do this.

The 'paths' heuristic is not enough to have a configuration with an inclusion factor over 0.25. Plus, due to the time complexity of the algorithm, the processing time to find an acceptable path for each node can be very long (200 seconds for the `Gen_40000` graph). However, the gain on the inclusion factor is not negligible. So, this heuristic can be used after the 'bilateral settlement' heuristic.

Open question: In our case, to form a path, the next edge is chosen by taking into account the previous edge weight. An improvement of the heuristic could be the computation of the hypothetical net positions of the nodes between the edges that are in the path and take into account this net position. It could improve the final amplification factor by choosing edges that are more suitable or by stopping a path earlier. For example, the last edge chosen from A to B in the previous example (cf. Figures 6b and 6c) would not be chosen. However, this improvement might find configurations which have a worst inclusion factor than the current version. What would be the inclusion factor and the amplification factor if hypothetical net positions of nodes were calculated when an edge is added to a path ?

This heuristic is still not enough to find a configuration $\mathcal{C}$ that attains the maximal inclusion factor $\iota(\mathcal{C}) = 0.5$. So, another heuristic that can attain every inclusion factor wanted was developed.

2.3 Best edge

The *best edge heuristic* is a heuristic that searches for the best edge available where the best edge is the edge which has the better effect on the amplification factor of the configuration.

This heuristic has two algorithms: 'best edge adding' which adds edges to the configuration and 'best edge removing' which removes edges from the configuration. Previous heuristics search for edges to add to a configuration. So these heuristics create a configuration ex nihilo. However, another way is to start from a configuration $\mathcal{C} = E$, and to remove edges from this configuration.

When we modify a configuration $\mathcal{C}$, by definition $\alpha(\mathcal{C})$ changes. So before doing this modification, we compute the hypothetical amplification factor for each edge available. Where this availability changes according to adding or removing edges from the configuration.

For graphs with a very important number of edges (over 50 000 edges) that are given to this algorithm, the hypothetical amplification factor of all remaining edges cannot be computed each time an edge is added to or removed from the configuration. Indeed, it would take to much time to compute all hypothetical amplification factors and to sort the list of edges. To fix it, a parameter `stepEdge` is introduced. This is the number of best edges taken in the sorted list, before this list is updated with the new hypothetical amplification factors.

This heuristic allows the configuration to reach any inclusion factor wanted.

2.3.1 Best edge adding

The 'best edge adding' heuristic is a heuristic that searches for the best edge available to add to the configuration. So for all edges $u \in E \setminus \mathcal{C}$, the hypothetical amplification factor $\alpha(\mathcal{C} \cup \{u\})$ is computed, and then the best edge u is chosen such as $\alpha(\mathcal{C} \cup \{u\}) = \max\{x \in \mathbb{R}^+ | x = \alpha(\mathcal{C} \cup \{v\}), v \in E \setminus \mathcal{C}\}$.

For example on the graph `graphPME`, the amplification factor as a function of the inclusion factor was computed for several values of `stepEdge` (see Figure 8). Depending on which inclusion factor is wanted, `stepEdge` can be chosen quite high. Let $\mathcal{C}$ be the final configuration with the inclusion factor requested. If $\iota(\mathcal{C}) \in [0.25, 0.35]$, `stepEdge` needs to be small. However, due to the number of edges needed to reach that inclusion factor, the processing time increases with the decrease of `stepEdge`. If $\iota(\mathcal{C}) \in [0.35, 0.5]$, a higher `stepEdge` can be chosen to gain time without decreasing the amplification factor.

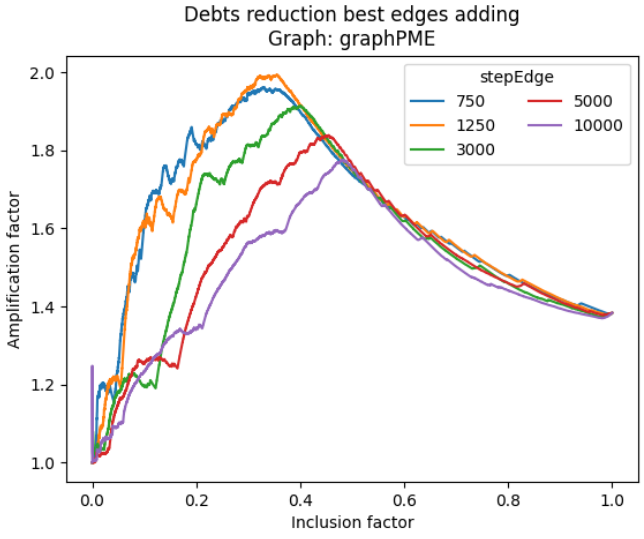


Figure 8: Amplification factor as a function of the inclusion factor during an invoice settlement with the best edges to add to the configuration

This heuristic is used when 'bilateral settlement' and 'paths' heuristics were not enough to reach the inclusion factor wanted. It allows the amplification factor of the final configuration to be higher. Indeed, the 'best edge adding' heuristic is applied on a graph with a non empty configuration with an acceptable amplification factor.

2.3.2 Best edge removing

The 'best edge removing' heuristic is a heuristic that searches for the best edge available to remove from the configuration. So for all edges $u \in \mathcal{C}$, the hypothetical amplification factor $\alpha(\mathcal{C} \setminus \{u\})$ is computed, and then the best edge u is chosen such as $\alpha(\mathcal{C} \setminus \{u\}) = \max\{x \in \mathbb{R}^+ | x = \alpha(\mathcal{C} \setminus \{v\}), v \in \mathcal{C}\}$.

As well as the 'best edge adding' heuristic, the amplification factor as a function of the inclusion factor was computed for various `stepEdge` (cf. Figure 9). Let $\mathcal{C}$ be the final configuration with the inclusion factor requested. If $\iota(\mathcal{C}) \in [0.3, 0.5]$, `stepEdge` needs to be small. However, due to the number of edges needed to reach this inclusion factor, the processing time increases with the decrease of `stepEdge`. Contrary to the 'best edge adding' heuristic, if $\iota(\mathcal{C}) \in [0.25, 0.3]$, a higher `stepEdge` can be chosen to gain time without losing in an amplification factor.

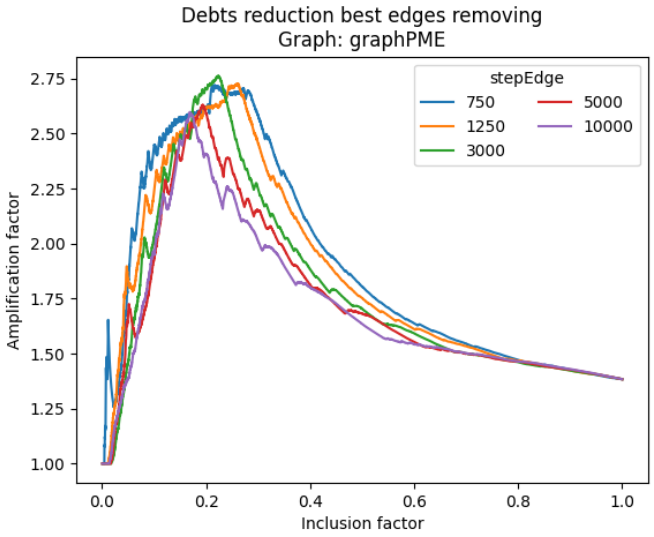


Figure 9: Amplification factor as a function of the inclusion factor during an invoice settlement with the best edges to remove from the configuration

This heuristic reaches a local optimum configuration where the inclusion factor is approximately 0.25 for both adding and removing edges. This is a surprising fact. Indeed, this optimum configuration has approximately a similar inclusion factor for both adding and removing edges. And we cannot explain this.

To find a configuration that reaches a desired inclusion factor there are several possibilities. On one hand, by starting from an empty configuration, we can combine 'bilateral settlement', 'paths' and 'best edge adding' heuristics to find a good configuration. On the other hand, by starting from a configuration $\mathcal{C} = E$, we can use the 'best edge removing' heuristic to find an acceptable configuration. Consequently, it is better to find a configuration by removing edges from a configuration $\mathcal{C} = E$, than adding edges to an empty configuration.

Once a configuration is found with the inclusion factor wanted, heuristics that improve the amplification factor can be applied.

3 Heuristics to improve the amplification factor

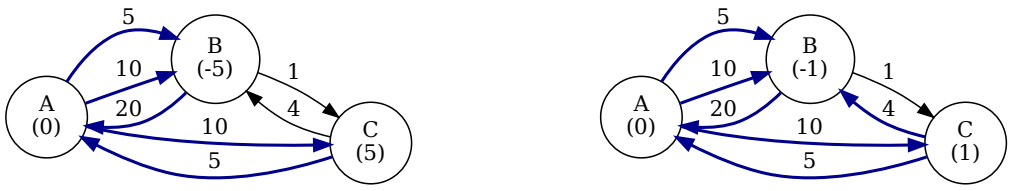
Previous heuristics were increasing the inclusion factor to reach a wanted one. This section presents heuristics which improve the amplification factor with very little impact on the inclusion factor. Note that these heuristics cannot work on an empty configuration. It means that they can only be used after the previous heuristics have been applied.

3.1 Spreading

Let $\mathcal{C}$ be a non-empty configuration. We define the spreading as an heuristic that given a node A where $\pi_{\mathcal{C}}(A) \neq 0$, the net position available of A can be used to add edges to the configuration. If $\pi_{\mathcal{C}}(A) < 0$, an incoming edge of A can be settled if its weight is less than the absolute value of $\pi_{\mathcal{C}}(A)$. If $\pi_{\mathcal{C}}(A) > 0$, it is the same procedure on an outgoing edge of A.

This heuristic is used between two heuristics which increase the inclusion factor. It allows the amplification factor to increase. Indeed, this procedure requires little or no financing to add edges to the configuration. Plus, the value of the beginning amplification factor has an effect on the final amplification factor after a configuration is found with a heuristic which increase the inclusion factor.

For example, given three companies A, B, C and a configuration where bilateral settlements were added (see Figure 10a). The net position of C is positive and greater than the weight of an outgoing edge from C to B. So, with the spreading, that edge is added to the configuration (see Figure 10b).



(a) Graph and a configuration before spreading

(b) Configuration after spreading

Figure 10: Example of the application of a spreading on a configuration

3.1.1 Spreading max

Let $B \in V$ be a node where $\pi_{\mathcal{C}}(B) \neq 0$. Let us assume that $\pi_{\mathcal{C}}(B) < 0$, $I(B) \cap F \neq \emptyset$, where $F = \{u \in I(B) | w_u \leq |\pi_{\mathcal{C}}(B)|\}$. The spreading max will choose the edge $u \in F$ where $w_u = \max(w_v, v \in F)$, to add to the configuration.

This heuristic is used between 'bilateral settlement' and 'paths' heuristics. As the configuration is small, the spreading is just used to increase the amplification factor. Indeed, the inclusion factor increases by only $2 \cdot 10^{-3}$ and the bilateral settlement heuristic is only used to start a good quality configuration.

3.1.2 Spreading financing

The 'spreading financing' method authorizes a financing. The parameter that represents the amount authorized is $\varphi \in [0; 1]$. If $\varphi = 0$, the 'spreading max' method is called. This parameter is the portion of the net position of the nodes $A \in V_{\mathcal{C}}$ that can be added to the net position where $V_{\mathcal{C}} = \{A \in V | (O(A) \cup I(A)) \cap \mathcal{C} \neq \emptyset\}$.

Let $D \in V$ a node where $\pi_{\mathcal{C}}(D) \neq 0$. Let us assume that $\pi_{\mathcal{C}}(D) > 0$, $O(D) \cap P \neq \emptyset$, where $P = \{u \in O(D) | w_u \leq \pi_{\mathcal{C}}(D)(1 + \varphi)\}$. The 'spreading financing' will choose the edge $u \in P$ where $w_u = \min(|w_v - \pi_{\mathcal{C}}(D)|, v \in P)$.

As the financing is authorized, it can be used to improve the amplification factor and increase the inclusion factor more than 'spreading max'. It can be used between 'paths' and 'best edge adding' heuristics.

The spreading is used between heuristics that increase the inclusion factor. However, the spreading increases the inclusion factor, even if there are only a few. So, we searched for a method to improve the amplification factor for the final step when the inclusion factor is reached. This method is simulated annealing.

3.2 Simulated annealing

The previous heuristics are specialized to our current problem. To increase the amplification factor for a given inclusion factor, a more general approach was studied: simulated annealing. This meta-heuristic approximates a

global optimization in a large search space.

In our case, we desire a global solution for a fixed inclusion factor. This global solution is the one that maximizes the amplification factor with a given inclusion factor.

Simulated annealing is inspired by annealing in metallurgy [8]. This is a heat treatment. The material is heated beforehand to give it a high energy. Then it is slowly cooled, with a slow temperature decrease. This treatment gives an equilibrium state to the material, with a global minimum of energy.

Two results of statistical physics are presented to better understand the algorithm:

- Given a system S at a temperature T, the probability p_j of S being in state j with energy ε_j is proportional to the Boltzmann factor [9]
$$p_j \propto \exp\left(-\frac{\varepsilon_j}{k_B T}\right)$$
where $k_B \simeq 1.380649 \times 10^{-23}$ is the Boltzmann constant [10].

- The Metropolis–Hastings algorithm with the Boltzmann factor as the probability distribution allows to describe the evolution of a thermodynamic system [8]. Due to the adding or the removing of an edge, there is a variation $\Delta\alpha$ of the amplification factor of the configuration. If $\Delta\alpha >= 0$, the modification is accepted, or otherwise it can be accepted with a probability $\exp(\Delta\alpha/T)$[11]. Accepting a "bad" configuration then allows us to explore a larger part of the set of configurations and tends to avoid locking ourselves too quickly into a local optimum.

As the inclusion factor is fixed, edges are removed or added to the configuration according to the current value of the inclusion factor: if the ongoing inclusion factor is greater than the one fixed, an edge is removed from the configuration, and conversely in the opposite case. So, during simulated annealing, the inclusion factor varies around the set value. To choose an edge to add to the configuration, it can be done by two different ways: a random choice or a choice in the set of frontier edges. We define the set of frontier edges $\mathcal{F} \subset E$ as edges that are not in the configuration $\mathcal{C}$ but share at least one node with an edge in the configuration: $\mathcal{F} = \{u \in E | u = (A, B, w_u), (O(A) \cup I(A) \cup O(B) \cup I(B)) \cap \mathcal{C} \neq \emptyset \text{ and } u \notin \mathcal{C}\}$. When we start from a configuration obtained with the 'best edge removing' heuristic, a better amplification factor is reached with the random choice in simulated annealing.

The variation of $\Delta\alpha$ is in the order of 10^{-3}, 10^{-4}. So the initial temperature needs to be close to this variation. The higher the temperature, the higher the probability to accept modifications to the configuration which are decreasing the amplification factor. The evolution of the temperature T follows a law of geometric decay with a coefficient of 0.999. This coefficient was chosen empirically and the results were satisfying. As the algorithm needs to stop, we have fixed a limit of iterations. For example, this limit is 100 000 for the graph `graphPME`.

With these parameters, we have made some tests. In Figure 11, the amplification factor as a function of the inclusion factor with the 'best edge removing' heuristic is represented in blue. After this invoice settlement, we applied simulated annealing on configurations $\mathcal{C}$ with an inclusion factor $\iota(\mathcal{C}) \in \{0.25, 0.3, 0.35, 0.4, 0.45, 0.5\}$ to see the improvement of the amplification factor when simulated annealing is applied as a final step on different configurations obtained previously. The improvement of the amplification factor is approximately between 0.1 and 0.06 for this case (cf. Figure 11). Moreover, the improvement is greater for smaller inclusion factors.

Open problem: If this internship were longer, we could study the different laws of evolution of the temperature and their effects on the results. We could also put a limit on the temperature with or instead of the limit of iterations. Furthermore, we could better optimize the parameters of the simulated annealing as explained in Appendix A.5.

The simulated annealing is important as a final step. Indeed, the amplification factor is greater. However, we find just a better configuration given an initial configuration as it is an NP-complete problem.

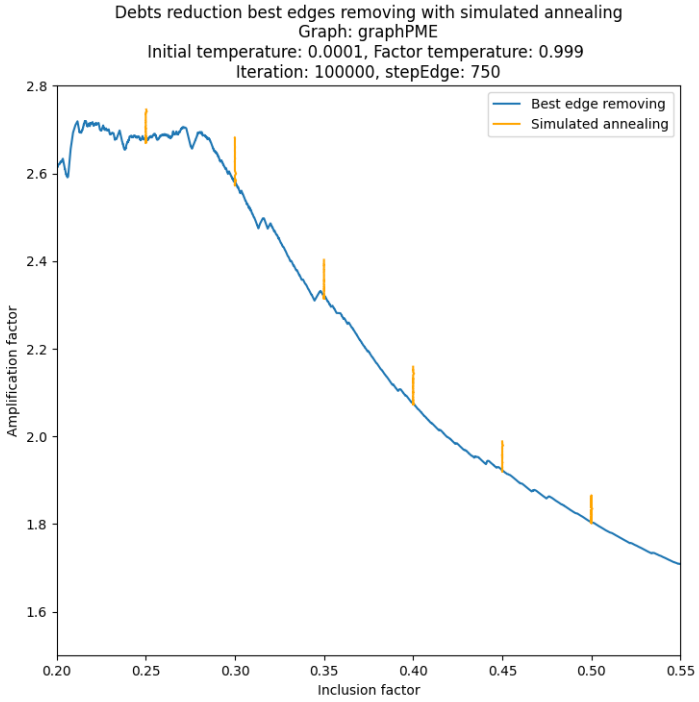


Figure 11: Amplification factor as a function of the inclusion factor during an invoice settlement with the 'best edge removing' heuristic and simulated annealing applied on some configuration with fixed inclusion factors

Conclusion

Given an inclusion factor in the interval $[0.25, 0.5]$, we tried to find an optimal configuration by developing heuristics. Compared to the results obtained previously [2] (cf. Figure 2), given an inclusion factor, some configurations found have a greater amplification factor. It means that our results are encouraging.

Firstly, heuristics that increase the inclusion factor were developed and two ways to reach this inclusion factor were studied. We discovered that removing edges from a configuration which contains all the edges of the graph is better than adding edges to an empty configuration. In this section, there was a lot of thinking to find heuristics which work. Moreover, a long process of reasoning and modifications were necessary to optimize these heuristics. Indeed, heuristics were applied on large graphs and their execution time needed to be low (less than some minutes for bigger graphs).

Then, we developed heuristics that improve the amplification factor. The improvement is significant. This part allowed me to discover the meta-heuristic of simulated annealing. The major difficulty in this section was to find good parameters for the simulated annealing. Indeed, the bibliography gave us some clues for the choice of those parameters. However, tests were needed for finer adjustment.

Several tracks exist to enhance the results obtained during this internship:

- by improving the current heuristics:
  - the 'bilateral settlement' heuristic: by applying an order on finding and financing bilateral settlements,
  - the 'paths' heuristic: by updating net positions of nodes in the path when paths are searched,
  - simulated annealing: by studying other laws of evolution of the temperature and their effect;
- by searching for new ideas:
  - genetic algorithms could be used after several configurations are found,
  - instead of a static system, a dynamical system could be used. More information would be in nodes, and this information would change when an edge is added or removed from the current configuration. They could guide the search for an optimal configuration.

This internship was my first research experience, and it was an amazing experience. Indeed, I was able to put into practice my theoretical knowledge acquired during my university studies, and to learn a new way of working by creating my own tools to reach my goal. Moreover, I was able to observe and understand how a research process works. I am glad to have been welcomed by the staff and thank them again for their warm welcome.

References

[1] Banque de France, *Impact économique des défaillances d'entreprise*, Bulletin de la banque de France, issue 147, 2005, pages 59-74

[2] AMATO,M., FATÈS,N., GOBBI,L., *The economics and algorithmics of an integral settlement procedure on B2B networks*, 2021, https://dx.doi.org/10.2139/ssrn.3915380

[3] IOSIFIDIS,G., CHARETTE,Y., AIROLDI,E.M. et al., *Cyclic motifs in the Sardex monetary network*, Nature Human Behaviour, Volume 2, Issue 11, 2018, pages 822–829, https://doi.org/10.1038/s41562-018-0450-0

[4] GÜNTZER,M.M., JUNGNICKEL,D., LECLERC,M., *Efficient algorithms for the clearing of interbank payments*, European Journal of Operational Research, Volume 106, Issue 1, 1998, pages 212–219

[5] ROUSSEAU,A., *Génération de graphes pour la compensation de dettes mutuelles entre entreprises*, 2021, https://members.loria.fr/nazim.fates/doc/lecture/rapportfinal-ArthurRousseau.pdf

[6] Git repository, https://gitlab.inria.fr/fates/kerdos-stage/-/tree/master/stageMarieVelaMena/PythonCodesIMDCP

[7] KUMAR,S., *python-dotenv 0.19.2*, Pypi, updated 11/11/21 [12/02/21], https://pypi.org/project/python-dotenv/

[8] SIARRY,P., *Métaheuristiques*, 2014, Eyrolles, pages 21-42

[9] SCHEAR,D.B., *The generalized Boltzmann distribution*, Journal of Theoretical Biology, Volume 39, Issue 1, 1973, pages 165-169, https://doi.org/10.1016/0022-5193(73)90211-7

[10] Bureau International des Poids et Mesures, *The International System of Units - 9^{th} edition*, 2019, page 18, www.bipm.org/documents/20126/41483022/SI-Brochure-9.pdf

[11] CHIB,B., GREENBERG,E., *Understanding the Metropolis-Hastings Algorithm*, The American Statistician, Volume 49, Issue 4, 1995, pages 327–335, https://doi.org/10.2307/2684568

A Appendix

A.1 Combinations finding

Before trying to find all combinations, the question was how many combinations there are when the number of edges between two nodes is known.

Let n be the number of edges between two nodes A and B. Let m be the number of incoming edges in A coming from B, and m' the number of outgoing edges from A going to B. We have $n = m + m'$. The number of combinations C_m of m edges without repetition is

$$C_m = \sum_{k=1}^{m} \binom{m}{k} = 2^m - 1.$$

Consequently, there is C_n combinations of bilateral settlement between A and B, where

$$C_n = C_m \cdot C_{m'} = (2^m - 1)(2^{m'} - 1) < 2^n.$$

For example, $2^{20} = 1,048,576$. As the time taken by the algorithm to research all combinations needs to be low, the number of combinations is limited.

During this internship, two ways to limit the number of combinations were developed. Let $n_{\max}$ be the maximal number of edges per combination. The number of edges $n = m + m'$ is greater than $n_{\max}$.

First, we introduce $n_{\min}$ the minimal number of edges per combination. Instead of finding all combinations, the number of edges in the computed combinations is limited by $n_{\min}$. So combinations contain less than or equal to $n_{\min}$ edges. In that case, the number of combinations $C_{n_{\min}}$ is

$$C_{n_{\min}} = \sum_{k=1}^{n_{\min}} \binom{n}{k}.s$$

After that, another way is to choose $n_{\max}$ edges in the set of available edges. Those edges can be chosen randomly, but in our case the criterion is their weight. Indeed, the goal is to reach an inclusion factor quite high. So more the settlement is high, faster the inclusion factor wanted is reached. The number of combinations in that case is $C_{n_{\max}} = 2^{n_{\max}} - 1$.

As we are more informed about the number of combinations, bilateral settlements can be searched.

A.2 Search bilateral settlement algorithm

The algorithm (cf. Algorithm 1) searches the best bilateral settlement for each pair of nodes which are strongly connected.

The different heuristics to choose the edges when the number of combinations is high were tested. Processing times were measured to compare the computation speed of search bilateral settlement algorithms on several graphs tests. In this test, $n_{\max} = 20$, $\alpha_{\min} = 5$.

In the algorithm `Combi_min`, if there are too many combinations, it searches all the combinations with a size less than or equal to $n_{\min} = 5$. In the algorithm `Combi_choice`, it searches all combinations in a defined set of edges with a size $n_{\max}$.

The algorithm `Combi_min` is slower than the other for `graphPME`. It is due to some bilateral settlements that have over 50 edges and the number of combinations that is greater than the two others algorithms. Indeed, the number of combinations can be greater than $2^{n_{\max}}$.

The bilateral settlements are added to the configuration. The results with bilateral settlements obtained with algorithms `Combi_min` and `Combi_choice` are quite similar. However, the processing time to search all bilateral settlements in algorithm `Combi_min` is greater than in algorithm `Combi_choice`. Therefore, algorithm `Combi_choice` is preferred to algorithm `Combi_min`.

A.3 Generator function

To search all combinations of a bilateral settlement between A and B two nodes, we first compute all combinations of incoming edges in A coming from B, and of outgoing edges from A going to B. So we have two generator iterators as output. Then with loops, we make combinations between those two generator iterators.

Algorithm 1 Search bilateral settlement

Input:
graph $G = (V, E)$
maximal number of edges per combination $n_{\max}$
Output: bests bilateral settlements for each pair of nodes that are strongly connected `listBestsBilSet`

```
for node in V that are not marked do
    node is marked
    destinationNodes ← set of nodes that have an incoming edge from node
    for nodeDest in destinationNodes do
        if nodeDest is not marked then
            n ← number of edges between node and nodeDest
            if n > n_max then
                Computing combinations with the way chosen
            else
                Computing all combinations
            end if
            Search for the best bilateral settlement bestBilSet among all combinations
            Adding bestBilSet to listBestsBilSet
        end if
    end for
end for
```

Combinations have different lengths. The generator function find all combinations without repeat value for each length of possible combinations (cf. Algorithm 2). It deals with the position of the edges in the input list. To avoid repeating a position, `listIndices` is always ordered before generating the edges combination.

A.4 Search path algorithm

The algorithm that search paths takes nodes randomly and then find a path by calling the algorithm with the wanted heuristic, for example the one with ratio (cf. Algorithm 3).

A.5 Statistics of simulated annealing

The parameters chosen to have the results show in Figure 11 are not optimized for all the configurations. The amplification factor as a function of the iterations was represented for some configurations with different inclusion factors (cf. Figures 12 and 13).

The function represented in Figure 12 does not seem to have a horizontal asymptote, contrary to the one represented in Figure 13. So, the number of iterations is not optimized for the configuration with an inclusion factor of 0.3. The variation of the temperature could not be optimal either.

Furthermore, in Figure 12 the decrease of the function in the beginning of simulated annealing have a variation of 3029 iterations in abscissa and approximately 0.013 in ordinate. To compare, in Figure 13 the variation is 831 iterations in abscissa and approximately 0.002 in ordinate. So, the variation of the amplification factor in Figure 12 of the configuration between the initial configuration and the configuration with the minimal amplification factor is greater than the one of Figure 13. Thus, a larger part of the space of possible configurations with an inclusion factor of 0.3 was explored. The initial configuration, the initial temperature and the variation of the temperature are the elements which influence this decrease.

Algorithm 2 Generator function

Input:
list of edges `listEdges`
length of combination `combiLength`
Output: a generator of tuples which have the weight of all edges in the combination and the combination `(edgesCap, combination)`

```
nbrEdges ← the number of edges in listEdges
existCombi ← True
if combiLength > nbrEdges then
    return
end if
for i = 0 to combiLength - 1 do                          ▷ Initialization of the list of indices
    listIndices append i
    combination append listEdges[i]
    edgesCap ← edgesCap + weight of listEdges[i]
end for
yield edgesCap, combination
while existCombi do
    for k ← combiLength − 1 to 0 do
        if listIndices[k] < k + nbrEdges − combiLength then
                                                  ▷ Boundary to avoid index out of the listEdges range
            listIndices[k] ← listIndices[k] + 1
            for j ← k + 1 to nbrEdges do
                                ▷ Reset next indices to their smallest possible value to not miss any combination
                listIndices[j] ← listIndices[j − 1] + 1
            end for
            combination ← empty list
            edgesCap ← 0
            for i = 0 to combiLength - 1 do
                            ▷ Generating the edges combination corresponding to the current indices combination
                combination append listEdges[i]
                edgesCap ← edgesCap + weight of listEdges[i]
            end for
            yield edgesCap, combination
            break                                  ▷ Restart the loop to find a new combination
        end if
    end for
    else                          ▷ No more combinations to find, all iterations through the for loop are done
        existCombi ← False
end while
```

Algorithm 3 Search path from an initial node with ratio

Input:
graph G,
initial node `initNode`,
minimal value for the ratio `limitRatio` $\in]0, 1]$,
minimal size of the path `sizeMin`

Output: list of edges that makes one path `listEdgesChosen` or `None`

```
a tuple tup := (inEdge, outEdge) of non marked edges of initNode is chosen where the ratio between the 2 edges capacity is over limitRatio
if tup is not in ({(None, None), (None, outEdge), (inEdge, None)}) then
    inEdge and outEdge are marked
    nextN ← the destination of outEdge
    previous N ← the source of inEdge
    Append inEdge and outEdge in listEdgesChosen
    while nextN is not None do                          ▷ i.e. outEdge is found for nextN
        an edge Edge from nextN is chosen where the ratio between its capacity and the outEdge capacity is over limitRatio
        if Edge is not None then
            outEdge ← Edge
            nextN ← the destination of outEdge
            Append outEdge in listEdgesChosen
        else
            nextN ← None
        end if
    end while
    while inEdge is not None do                         ▷ i.e. inEdge is found for previousN
        an edge Edge from previousN is chosen where the ratio between its capacity and the inEdge capacity is over limitRatio
        inEdge ← Edge
        previousN ← the source of inEdge
        Append inEdge in listEdgesChosen
    end while
    if size of the path is less than sizeMin then
        return
    else
        return listEdgesChosen
    end if
end if
```

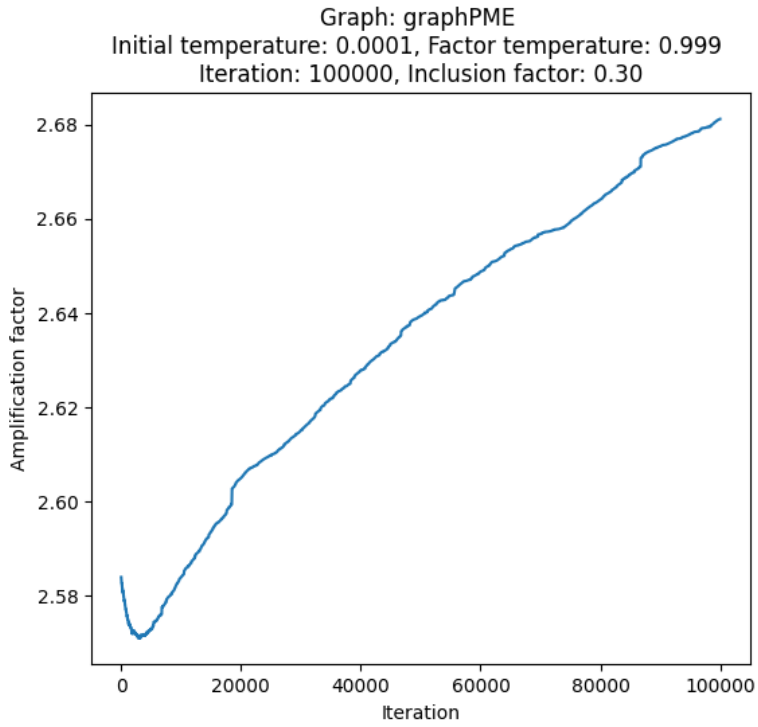


Figure 12: Amplification factor as a function of the iterations during simulated annealing. The configuration has an inclusion factor of 0.3 and was obtained with the 'best edge removing' heuristic with `stepEdge` = 750.

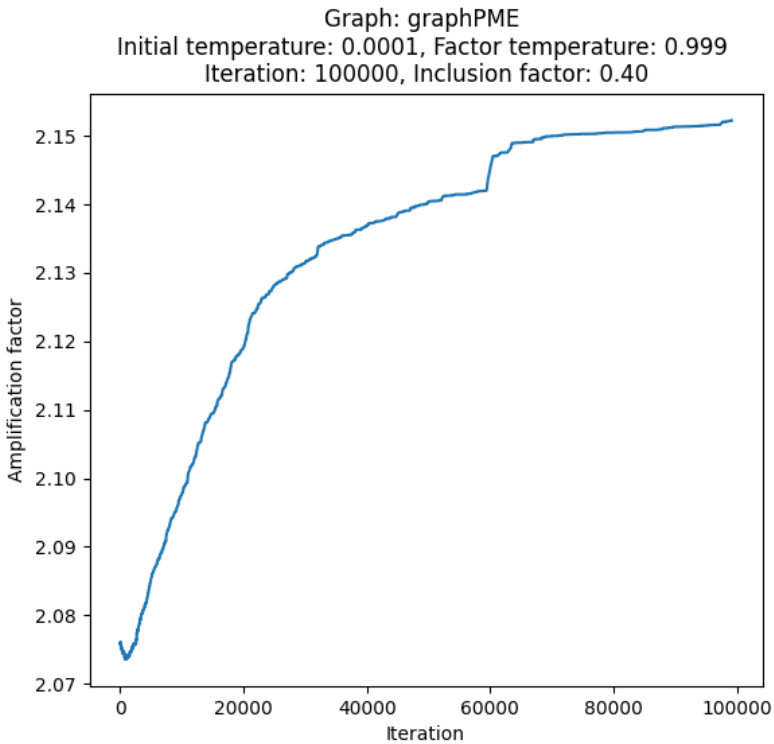


Figure 13: Amplification factor as a function of the iterations during simulated annealing. The configuration has an inclusion factor of 0.4 and was obtained with the 'best edge removing' heuristic with `stepEdge` = 750.

Abstract

Given an amount of debts to settle, the objective of this internship was to find a solution of invoices that minimize the supply of liquidity required to clear this amount. This is an NP-complete problem.

First, we present heuristics that find a good solution given an amount of debts to settle. Then, we present heuristics that minimize the supply of liquidity required given a solution.

Keywords: graphs, heuristics, integral mutual debts compensation, NP-complete problem