Analysis of payment graphs between companies with a goal to reduce mutual debts

Yosyp Mykhailiv
Dissertation 2021
Master of Computer Science - MFLS

Faculty of Science and Technology,
University of Lorraine,
Nancy, France.

A dissertation submitted in partial fulfilment
of the requirements for the
Erasmus Mundus MSc in Advanced Systems Dependability

Team: Mocqua
Supervisor : Dr Nazim Fatès
Programme supervisor: : Dr Didier Galmiche
Date: 29 June, 2021

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study as part of Erasmus Mundus Joint MSc in Advanced Systems Dependability qualification, is entirely my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: Yosyp Mykhailiv Date: 29 June, 2021

Acknowledgements

I want to thank my supervisor, Dr Nazim Fatès, for developing my curiosity in the topic, giving me directions, criticising when needed and giving valuable comments on my work.

I would also like to acknowledge EACEA (Education, Audiovisual and Culture Executive Agency) for giving me an opportunity to study at one of the best European universities and covering all my expenses.

I wish to show my gratitude to my family, friends and DEPEND students who were supporting me when I was challenged by different issues while working on the project.

Abstract

The solvency of agents in a financial network is a crucial factor for having a stable economy and lowering the risks of a financial crisis. Since mutual debts have a negative effect on solvency, in this work, we try to develop an algorithm that would minimize the amount of mutual debt in a network of unpaid invoices. At first, we review and analyse the existing works on the given topic. Then, we define the problem we are trying to solve using mathematical notations. We focus on integral debt cancellation and show a few approaches that we tested while looking for a sufficient algorithm. We present algorithms that are based on brute-force settlement as well as a more sophisticated method that takes into account different properties of a directed multigraph. Each algorithm is presented with its evaluation for different data sets. We mention the effect of some parameters on the execution results and express our thoughts on how to achieve better results. Finally, we talk about the influence of the graph structure on debt settlement and outline the directions for future work.

Contents

List of Figures

Chapter 1: Introduction

This chapter gives an overview of the research topic. It explains why the problem is important. We also give a brief description of the methods used in the research and the metrics used for evaluating the results.

1.1 Topic addressed in the research project

Recently, there has been a lot of research in modelling financial networks. This was mainly motivated by financial crises and a lack of solutions for modelling and measuring financial relations in big networks. Gazda *et al.* mentioned that "The research results demonstrate a significant impact of the financial network structure on the financial risks given by the potential extent of the insolvency contagion spreading through the net" [1]. Additionally, the authors mention several research works that have quite controversial results in terms of how exactly a financial network is influenced by its structure. As a result, they say "we base our approach on the intuition that the increasing number of debt relations has a positive impact on the danger of financial insolvency." [1]. Overall, we share this idea and will work on deriving practical methods and algorithms that would reduce mutual debts.

1.2 Motivation

We believe the importance of this research lies in the opportunity to improve the financial situation of companies that often face liquidity problems and cannot pay off their liabilities due to delayed incoming payments. Solving such a problem on a more global level could potentially stabilize bigger financial networks and minimise the risks of getting into a new financial crisis. This work could also help to highlight some correlations between the structure of financial networks and their properties that would be useful for further research.

1.3 Problem overview

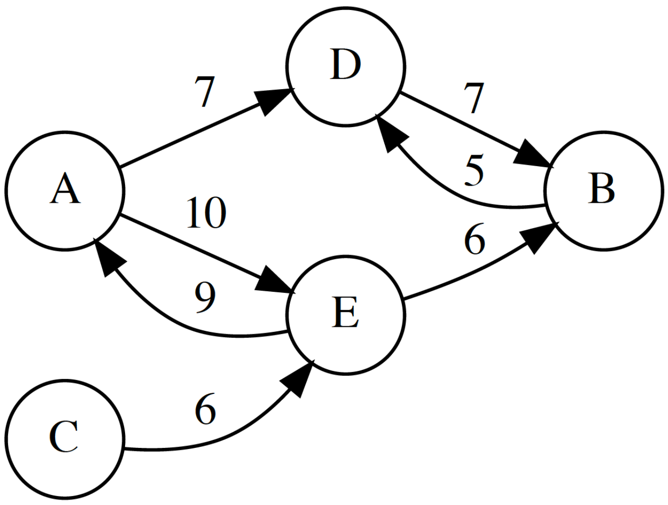


Figure 1.1: Example of a financial-network graph

In Figure 1.1 we can see an example of a financial network that is represented as a graph (here we show a directed graph for simplicity of the example but in reality, we will work with directed multigraphs). In this graph nodes represent companies and edges represent unpaid invoices. One of the main goals of our research is to find an approach that would help us identify the invoices that need to be financed first and as a result, it would give us the best ratio between the total amount of settled debt and financing used for settling the debt.

As per the given example, we can observe that the best ratio between removed debt and financed amount can be achieved by removing the cycle AE (edge AE with capacity 10 and edge EA with capacity 9). In this case, the settled debt is equal to 19(10+9) while the financed amount is equal to 1. The financed amount is the total amount of money the companies still need to pay after some invoices were removed. As in our example, company A still needs to pay 1 euro to company B to compensate the difference between the amounts of the two invoices. If we do not do this then, company E and A would lose and gain 1 euro respectively which results in changing the balances of the companies. That is something that is prohibited by the definition of the mutual debt reduction problem. In reality, we are not interested in finding the highest possible ratio of settled debt to the financed amount, but we want to find the minimum amount of financing needed for settling at least a given percentage of the total debt.

The main difficulty in getting satisfactory results is the size of the network. For example, one of the data sets, that we possess access to, consists of 38,334 companies and 145,540 invoices. Clearly, the bigger the graph is, the more time it will take to process it. This basically means that the algorithm for solving the given problem should be of polynomial time complexity to get the output within a reasonable amount of time.

1.4 Approach

First of all, we have to mention that we will focus on *integral debt cancellation* which means that we will not be changing the invoice amount but either remove it from the graph or not. This leads us to the point when in order to achieve a good result we will need to apply external financing. This is due to the fact that we rarely will encounter two invoices with the same amount issued in opposite directions between two companies.

The first and the most naive approach that we will test is brute force exploration of a graph (the algorithm is called "BruteForce"). This algorithm will check every possible combination of edges to see what settlement options are optimal. Another similar algorithm that we will use is called "IterativeBruteForce". It will find the best possible ratio between the settled debt and financed amount factor and step by step will be decreasing it by adding more edges to the initial combination in such a way that every next combination will be optimal in relation to the previous one. These two approaches clearly are going to work only for small graphs due to the exponential complexity of checking all the possible combinations.

Another approach, which can be used for bigger graphs, is picking combinations randomly while doing this for a limited number of iterations. It is obvious that this approach is unlikely to be always giving us the best possible result, but it is a good way to explore bigger graphs.

And finally we will develop a more advanced algorithm that will take into account different properties of directed multigraphs and financial networks for achieving good result(often not the best) within reasonable execution time.

1.5 Metrics

The first property that will be used for estimating the results is the *amplification factor*. That is the already mentioned ratio between the settled debt and the financed amount. For example, if the amount of the settled debt is equal to 100,000 euros and the financed amount is 20,000 euros then, the amplification factor is 100,000/20,000 which equals to 5.

Sometimes the amplification factor can be really good, maybe even too good. This usually happens when a small amount of debt is settled with very little financ-

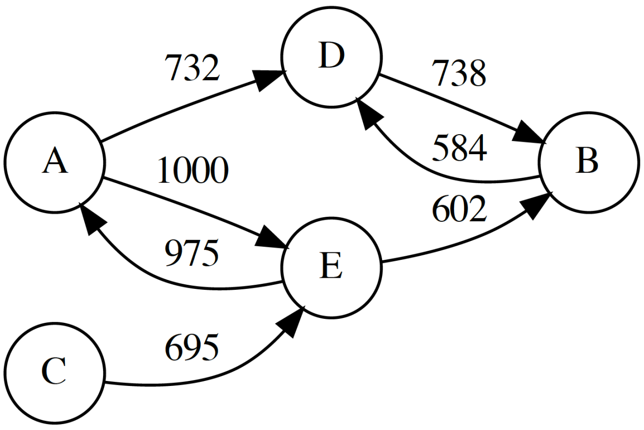


Figure 1.2: Too high amplification factor

ing(possibly none). For example, in Figure 1.2 we can see two invoices between A and E that are issued in opposite directions. The AE one for the amount of 1,000 euros and the EA for the amount of 975 euros. Those two invoices create a cycle elimination of which costs only 25 euros while the settled debt is equal to 1,975 euros. As a result, the amplification factor is equal to 79. In reality, we are not interested in getting the highest amplification factor for a given graph, but we also want to make sure that a certain percent of the total debt is settled. This way we will be looking for a high amplification factor under the condition that at least a given percent of the total debt is settled.

Of course, the execution time of the algorithm is important and it will also be taken into account when estimating our results. Another aspect that we are interested in is the influence of the shape of the graph on the ability to reduce mutual debts. So we will study the correlations between mutual debt reducibility and the graph structure.

1.6 Dissertation outline

The report consists of five chapters, each of which is briefly described below:

- The first chapter gives an overview of the research topic, emphasises the importance of solving the given problem and also presents the approaches that will be tested while trying to find a solution.
- The second chapter dives into the work that has been already done in the field. It mainly focuses on existing work for mutual debt cancellation as well as similar problems in the context of graph theory.

- The third chapter precisely defines the problem. In this chapter, we will take an advantage of using mathematical notations to minimise ambiguity.
- The fourth chapter presents all the attempts to find a good solution, the ones that were successful as well as those that did not bring significant results. Evaluation of results for each approach is also given in this chapter.
- The fifth section, which is the last one, concludes the work that is done. It emphasises the main results and also suggests directions for further research.

Chapter 2: Related Work

This chapter focuses on the work that has already been done in the field. At first, we will analyse existing publications on topics close to ours. Then we will discuss different problems and solutions in graph theory that could be useful for our research.

2.1 Mutual debt cancellation problem

Due to multiple financial crises, there has been a surge in the amount of research in the field of financial networks. Researchers try to explain the influence of the topology of a financial network on its stability and solvency of its members. One of such research works has been done by Iosifidis et al. [2]. The work sheds some light on the Sardex network which, as the authors claim, is a "real alternative economy" [2]. Sardex is a real financial network, the only difference is that its members do not use any fiat currency within the network. They use an electronic-only complementary currency that cannot be exchanged for fiat money which makes it a mutual credit system. The authors focus on cyclic motifs of the network and present a set of metrics for analysing the presence of cycles. They discuss the possible roles of cycles in such a network but they do not make any conclusion about the influence of the presence of cycle.

Besides the integral debt cancellation, that we focus on, some researchers have explored the problem of *partial debt cancellation*. The idea of partial debt cancellation is based on the ability to change the amount of any invoice without changing the net position of a source or destination company. We already mentioned one of the works that explores the problem of partial debt cancellation, that is the one by Gazda et al. [1]. The authors base their approach on the identification and elimination of cycles in a given graph. They present the problem from a different angle by saying that the maximum debt reduction can be achieved by finding the maximum circulation. The authors suggest that the maximum circulation problem can be tackled by Klein's algorithm which was originally used for finding minimum cost circulation [3]. The work also presents scenarios where the organizer of the debt reduction process receives some profit or when the process is handled by a non-profitable organization, like the Ministry of Finance.

The approach of doing integral debt cancellation is not absolutely new and has also been explored. For example, M.M.Güntzer et al. in their work related to interbank payment clearing demonstrate algorithms that use integral debt cancellation [4]. They do the clearing process by running a bilateral integral debt cancellation algorithm in the morning and a multilateral algorithm in the afternoon.

One of the most important questions in studying the mutual debt cancellation problem is its complexity. It has been already shown by Güntzer et al. [4] and Pătcaş [5] that the problem is NP-complete.

2.2 Relevant problems and solutions in graph theory

There are multiple solutions in graph theory that could be potentially re-used when trying to solve the mentioned problem. The most obvious one is the detection of all the cycles in a graph as a cycle is a structure that allows obtaining a very high amplification factor. A simple depth-first search can be applied when looking for cycles, but there are more sophisticated solutions such as Johnson's algorithm [6]. However, usage of cycle-detection algorithms is quite limited in this situation due to their high time complexity.

Another solution relates to the max-flow minimum-cut theorem. Using an algorithm that finds max flow for a subset of edges (e.g. cycle, path, etc.) could be applied in the context of integral debt cancellation. It could be used to estimate how beneficial it is to finance a certain set of edges in a graph based on the max flow and total debt of the set. Ford–Fulkerson [7] method is a good example of the algorithm that solves the maximum flow issue.

One more interesting idea is to analyse strongly connected components of a graph first rather than analysing the graph as a whole. They could be identified by application of Tarjan's algorithm [8]. This simply could improve performance. However, it is very unlikely that in a real network we would get many of them. Probably for the majority of real financial networks, a graph itself would be a strongly connected component, but this is something that could be studied. Besides using strongly connected components, we could split a graph in pieces by dividing it into clusters of nodes that have many edges between each other.

2.3 Software and libraries

First of all, we have to mention that we use Python for implementing all the algorithms mentioned in this report. Taking this into account, it is important to highlight a Python library called NetworkX [9] that provides a lot of ready-to-use graph algorithms such as detecting cycles, finding maximum flow(minimum cut theorem), etc. The library supports non-directed and directed graphs as well as directed multi-graphs. At the beginning of our work, we relied on this solution since it helps to quickly start testing different ideas. However, later we implemented our own set of classes (Edge, Node, etc.) to have more control over how any operation (e.g. adding, removal of an edge) is processed. Having control over every operation is crucial in the context of performance, especially when there is a need to tune it up.

Another piece of software that we use is Graphviz [10]. The package allows visualising graphs of different complexity(including directed multi-graphs). It consists of a set of tools that can read and generate graphs described using DOT language. The library supports different rendering engines and multiple output formats. The only drawback of this package is that it cannot be used for rendering huge graphs as it takes relatively a lot of time to produce a PDF with 1000 edges or so. Another handy library that we used is the widely known Matplotlib [11]. We used it for visualizing the shape of the datasets and visual estimation of the results.

Chapter 3: The Problem

In this chapter we look more deeply into what we call a financial network, define some notions that we will use and clearly describe the problem with mathematical notations.

3.1 Financial network, graph, net positions, etc.

As mentioned already a financial network is a directed multigraph. In such a graph, nodes and edges represent companies and invoice respectively (edge weight indicates invoice amount). While working with such a data structure we will often refer to its different parts and some concepts. Let's clearly define what is what using notations.

Let $G = (N, E)$ be a directed multigraph and we consider this to be a fixed parameter. $N = \{n_1, ..., n_i\}$ is a set of nodes and $E = \{e_1, ..., e_j\}$ is a set of edges where $i, j \in N$. Each node $n \in \{A, \ldots, Z\}^*$ is a word of letters from A to Z. Each edge e is presented as a triple of form (s, d, a) where s is a *source* node, d is a *destination* node and a is the edge *weight*. We denote the functions of edge source, destination and weight as follows $\sigma(e) = s$, $\delta(e) = d$, $\omega(e) = a$. $out(n) = \{e \in E : \sigma(e) = n\}$ denotes a set of *outbound* edges and $in(n) = \{e \in E : \delta(e) = n\}$ denotes a set of *inbound* edges.

Similarly we define the set of outbound and inbound edges in a subset $S \subset E$ as follows: $out'(n, S) = \{e \in S : \sigma(e) = n\}$ and $in'(n, S) = \{e \in S : \delta(e) = n\}$. A set of nodes that are part of an edge subset are defined as $nodes(S) = \{n : n = \sigma(e) \text{ or } n = \delta(e), e \in S\}$. Also, let $\Sigma(S)$ be the sum of weights of edges in S calculated as follows: $\Sigma(S) = \Sigma_{e \in S}\omega(e)$.

Let's also introduce the idea of *settlement* formally. Our goal is to find a *configuration*, the subset of edges that will be removed from the graph with financing applied where needed. We will denote a configuration as $C \subset E$. We want to emphasise that a configuration is always a subset of edges but, not every subset of edges is a configuration, e.g. a few edges that can be potentially added to a configuration is a subset but not a configuration.

We will be often referring to absolute and relative net positions that are defined as follows:

- **Absolute net position** - the difference between the amounts of all inbound and outbound edges, calculated as follows: $anp(n) = \Sigma(in(n)) - \Sigma(out(n)) = \Sigma_{e \in in(n)} \omega(e) - \Sigma_{e \in out(n)} \omega(e)$. For example, in Figure 3.1, $anp(D) = 578$;
- **Relative net position** - the difference between the amounts of inbound and outbound edges in a subset, calculated as follows: $rnp(n, S) = \Sigma(in'(n, S)) - \Sigma(out'(n, S)) = \Sigma_{e \in in'(n,S)} \omega(e) - \Sigma_{e \in out'(n,S)} \omega(e)$. For example, in Figure 3.1, $rnp(A, S) = -25$ for $S = \{(A, E, 1000), (E, A, 975), (E, B, 602)\}$;

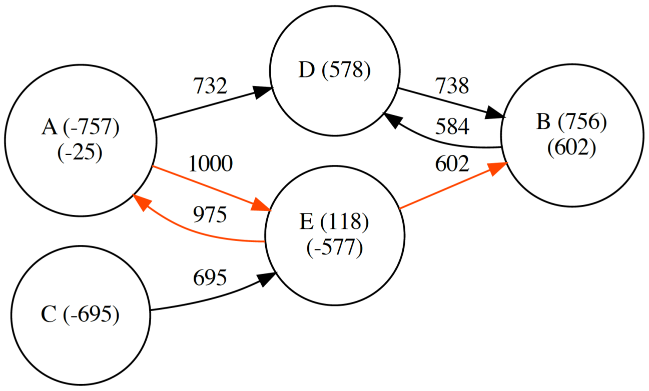


Figure 3.1: Net positions (absolute net position - in parentheses near a node name; relative net position - in parentheses below node name); Red colour represents the current configuration

When all the edges in a graph are settled ($C = E$) then $\forall n \in nodes(C)$: $rnp(n, C) = anp(n)$.

3.2 Mathematical definition of the problem

As we stated before our goal is to find a configuration to remove from the graph. Before defining the criteria for a configuration, let's define a few notions and notations that we will use to describe the goal precisely:

- **Settled debt** - the sum of weights of edges in a subset, calculated as follows: $sd(\mathrm{S}) = \Sigma(S) = \Sigma_{e \in S} \omega(e)$;

- **Financing** - the sum of absolute values of negative relative net positions, calculated as follows: $fin(S) = \Sigma_{n \in nodes(S)} |rnp(n, S)|$ if $rnp(n, S) < 0$;
- **Amplification factor** - the ratio between the settled debt and financing, calculated as follows: $\alpha(S) = sd(S)/fin(S)$;

Now, we say that we look for C that produces the highest $\alpha(C)$ when $fin(C)$ is fixed.

Chapter 4: Solutions and Evaluations

This chapter shows the range of approaches and solutions that were explored and implemented. Here we also present results that we obtained for each implementation and compare different executions. Before we start introducing different algorithms it's worth mentioning that we do not only aim to find one configuration, but to explore how the amplification factor evolves while we add more edges. Of course, at each step we will be trying to add as many edges as possible with minimum amount of financing willing to reach the best outcome.

4.1 BruteForce algorithm

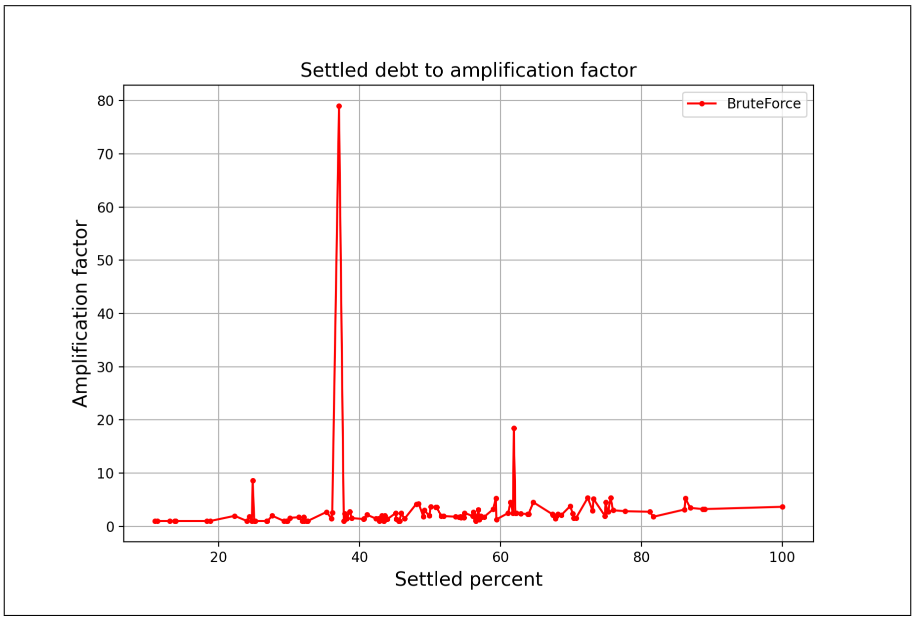


Figure 4.1: The best ratios between settled debt and amplification factor for $G5$-7

The first algorithm that we will use is just a simple brute force exploration of all the possible combinations of edges in a graph for calculating the amplification factor for each combination. By knowing the settled amount for each combination and the resulting amplification factor we will find what is the best possible amplification factor for any possible amount of settlement for a given graph. In Figure 4.1, we can see the curve that is a result of the execution of this algorithm. It shows what is the highest amplification factor for any possible amount of settlement for the graph we saw in the introduction. From now on, we will denote this graph G5-7 where $i = 5$ and $j = 7$ which is per our definition the number of nodes and edges respectively. What we also can observe in Figure 4.1, the highest amplification factor for the given graph overall, as stated before, is 79.

This algorithm shows us what the best possible result is. However, its usage is restricted to small graphs. For the mentioned graph, the algorithm produces results instantly. However, it is not the case for larger graphs. Since the algorithm checks all of the combinations of edges, the more edges we have in a graph the more time it takes to process it. In the left part of Figure 4.2, we can clearly see this dependency.

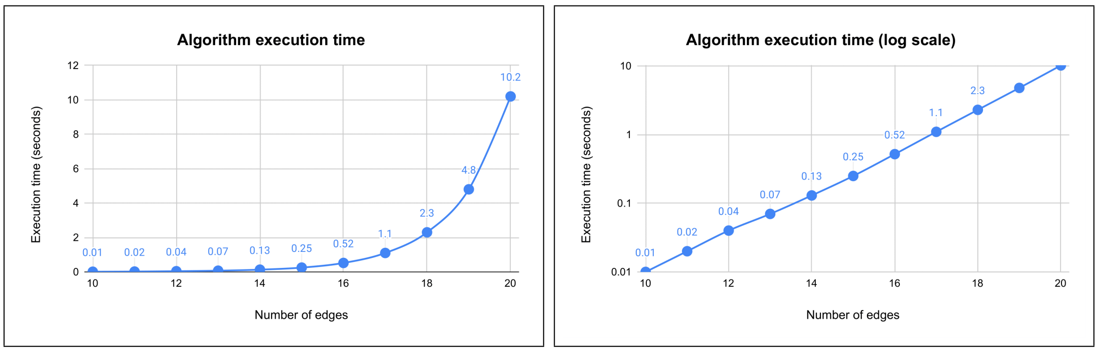


Figure 4.2: BruteForce execution time

In this example, the number of nodes was fixed to 8 while the number of edges was changing from 10 to 20 to demonstrate the rise in the execution time. Also, to prove that the algorithm has an exponential-time complexity we draw a line as a function of the execution time in logarithmic scale that is shown in the right part of Figure 4.2. We can observe that the spect of the function is linear which indicates that the execution time is exponential.

4.2 IterativeBruteForce algorithm

The second algorithm that we implemented is IterativeBruteForce. It considers all the possible combinations of not settled edges in the given graph and finds which combination produces the highest amplification factor in a combination with already settled edges. Below we can see a more precise description of the algorithm:

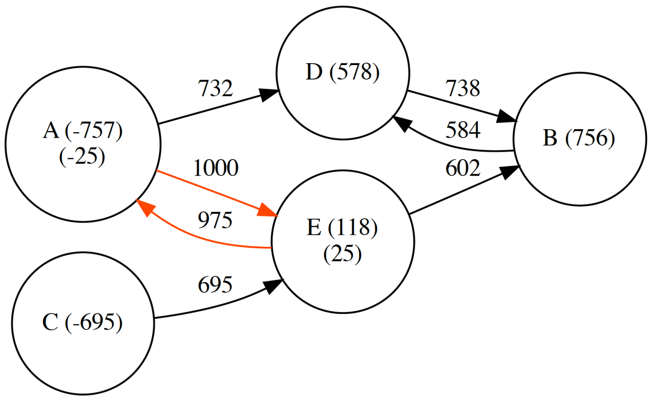


Figure 4.3: Mutual debt reduction for G5-7 using IterativeBruteForce algorithm

- Until the desired amount of debt is settled, repeat the following steps:
  - Find a combination of edges(at least 1 edge) that together with already settled debt and injected financing produces the best amplification factor;
  - Based on the found combination, increase the amount of settled debt and injected financing;

This algorithm is also simple and useful only for small graphs because of the already mentioned inefficiency of analysing all of the combinations of edges in a graph. We want to emphasise the fact that only the first combination produced by the algorithm produces the highest amplification factor while the following ones are the best only in relation to already settled debt. We will see later that those non-primary combinations may not be the optimal choices for their specific amounts. In Figure 4.3 we can observe the output this algorithm produces for the graph G5-7. As expected, the picked configuration is $C = \{(A, E, 1000), (E, A, 975)\}$ with $sd(C) = 1975$, $fin(C) = 25$ and $\alpha(C) = 79$. In this situation, the amount of total debt is 5326 euros while the percentage of the settled debt is about 37. As we mentioned before, we are interested in settling at minimum a specified per cent of the total debt. Let's say we want to settle at least 60 per cent and see what would be the highest amplification factor and amount of financing needed. The implemented algorithm supports this scenario.

In Figure 4.4 we can see the result of running the algorithm with the requirement to settle at least 60 per cent of the total debt. As a result, $C = \{(A, E, 1000),$ $(E, A, 975), (D, B, 738), (B, D, 584)\}$ with $sd(C) = 3297(\sim 62\%)$, $fin(C) = 179$ and $\alpha(C) = 18.42$.

From the previous two examples, we can observe a sharp drop in the resulting amplification factor. By seeing this, we could probably say that the higher the amount of settled debt, the lower is the amplification factor. But is it always the case that a

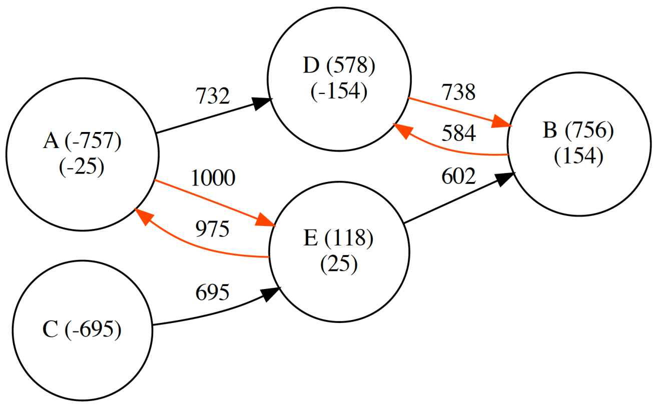


Figure 4.4: Reduction of at least 60 per cent of total debt for $G5$-7 using IterativeBruteForce algorithm

higher percentage of the total debt requires more financing? Going further, we will try to learn more about possible scenarios.

The algorithm is designed in a way that whenever we want to settle a fraction of the total debt, it brute forces not settled edges of the graph the number of times it needs to reach the goal. It allows us to see the intermediary results at each iteration. Using this approach, we are saying we want to settle 100 per cent of the total debt and output a settled per cent and the corresponding amplification factor at each step. The chart that represents the output of this execution is presented in Figure 4.5. The first two points correspond to the two settlement cases we mentioned earlier. From the chart, we can observe that the more debt we settled the lower the amplification factor is, which leads to a higher amount of financing.

4.3 TinyCycles algorithm

As could be noticed, the algorithms presented before are quite simple, even trivial and do not give us the results we want. The main reason they are presented is to show the bound of the problem and get some intuition about further directions that can be developed. Here we present an algorithm that is more sophisticated, produces relatively good results and can be used on large datasets. One of the key points of the algorithm is using ”tiny cycles” (cycles of length two). Such cycles are quite common in real financial networks so it makes sense to utilise them. Another part of the algorithm relies on taking augmenting paths (e.g. traversing a graph in directions that match certain conditions). Before giving a more precise description of the algorithm we will define a few notions:

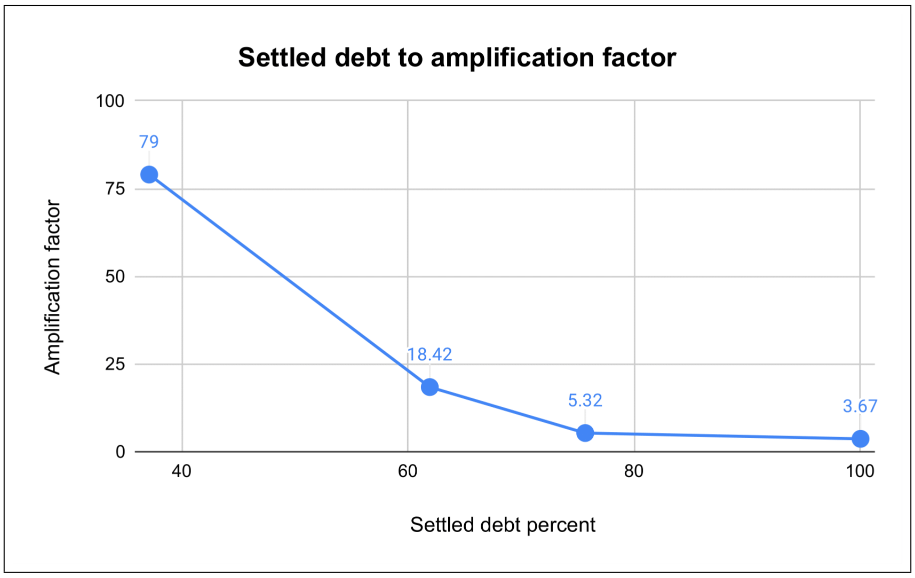


Figure 4.5: Amplification factor trend for $G5$-7

- **Abstract cycle** - a pair of nodes that have at least two edges in opposite directions (notation: (s, d) for $s, d \in N$);
- **Concrete cycle** - two edges in opposite directions between 2 nodes (notation: $\{(s, d, w_1), (d, s, w_2)\}$ for $(s, d, w_1), (d, s, w_2) \in E$);
- **Best concrete cycle** (defined for an abstract cycle) - a concrete cycle with minimum difference between amounts of its two edges;
- **Elementary bilateral settlement** - addition of the edges of a concrete cycle to a configuration;
- **Augmenting paths** - a set of consequent edges that can be reached from a node that is already a part of a configuration (notation: $ap(n) = \{(n, d_1, w_1), (d_1, d_2, w_2), (s_3, d_2, w_3), (s_4, d_2, w_4), ...\}$ for $n \in N, \forall e \in ap(n) : e \in E$). From the mathematical definition we can see that a path can be a combination of forward and backwards edges and it can also split into multiple paths;

Another concept that is important for better understanding of the algorithm is *minimum amplification factor* (notation: *mamp*). That is the parameter (a lower bound) to which the amplification factor of a configuration can be decreased at each iteration of the algorithm.

Now when we are aware of the introduced terms, let's see a brief description of the algorithm:

1. Pre-process the graph:
   (a) Find all the abstract cycles;
   (b) For each abstract cycle, pick the best concrete cycle. The algorithm checks all the combinations of forward and backwards edges to find the 2 edges in opposite directions with minimum weight difference;
   (c) Perform elementary bilateral settlement if an amplification factor of a concrete cycle is not lower than **CAT**[1];
   (d) Based on the settled amount and financing injected so far, define the initial minimum amplification factor according to the formula used for calculation of an amplification factor;
2. Until the desired amount of debt is settled repeat the following steps:
   (a) Find the best concrete cycles for already known abstract cycles and perform elementary bilateral settlement if an amplification factor of a concrete cycle is not lower than the minimum amplification factor;
   (b) From nodes in the configuration, sorted descendingly by relative net position, take forward or backwards augmenting paths if the relative net position is positive or negative respectively. Try the opposite direction for a path if no edges are available or if the relative net position changes its sign after some edges were settled. The depth of a path is limited by **APMD**[2]. Add an edge to the configuration if the resulting amplification factor is not lower than the minimum amplification factor;
   (c) If the minimum number of edges, which is 1 by default, is not settled after one iteration, decrease the minimum amplification factor.

In the very last step, we say that we decrease the minimum amplification factor if the desired number of edges was not settled. It is important to know how we decrease it. It is based on a few parameters:

1. **Initial decline percent** (notation: *idp*) - the percent by which the minimum amplification factor will be decreased for the very first time;
2. **Decline percent coefficient** (notation: *dpc*) - the coefficient by which the decline percent is multiplied if the desired number of edges was not settled during the very last iteration;

[1]Cycle amplification threshold, TC_CYCLE_AMPLIFICATION_THRESHOLD in the configuration file.

[2]Augmenting path maximum depth, TC_AUGMENTING_PATH_MAX_DEPTH in the configuration file.

3. **Maximum decline percent** (notation: *mdp*) - the maximum percent by which the minimum amplification factor can be decreased.

Based on the parameters above, the minimum amplification factor is calculated as follows:

$$dp = idp;$$

$$dp' = dp \cdot dpc^{iteration-1}$$

$$dp' = min(dp, mdp);$$

$$mamp' = mamp \cdot (1 - dp \cdot 0.01)$$

where *iteration* is the number of the current iteration during which the minimum amplification factor should be recalculated. Using this approach allows us to control how the minimum amplification factor is decreased in the beginning and after a few iterations. As result we can chose how greedy each iteration should be.

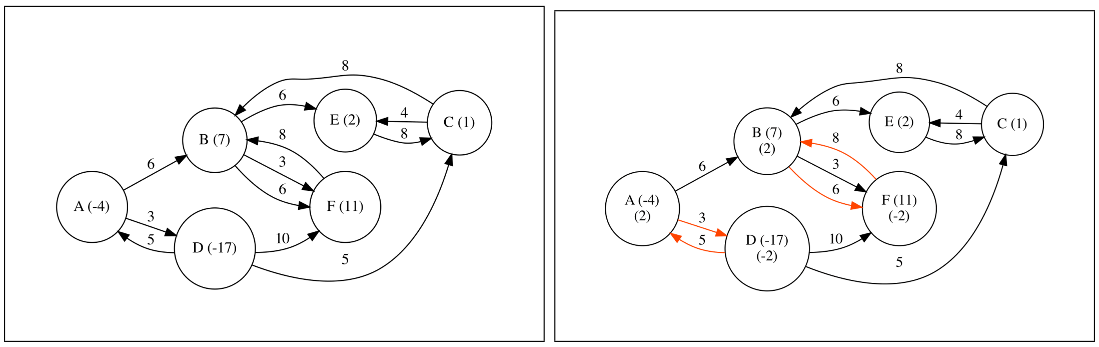


Figure 4.6: TinyCycles: $G5$-7 initial state and after pre-processing

Since we already know how the algorithm works, let's consider an example to see how a graph is processed by this algorithm. In the left part of Figure 4.6, we can observe the initial version of a graph that we will run the algorithm on. The algorithm has a few parameters that can be configured. However, for this execution we change only two parameters with the rest being set as default:

- **CAT** = 4 (the value that is used in step **1.c**);
- **APMD** = 2 (the value that is used in step **2.b**);

In the right part of Figure 4.6, we have the same graph, but already in the state that is a result of the pre-processing stage. To reach this state, the following steps were executed:

1.a : found abstract cycles (A, D), (B, F) and (E, C);

1.b : picked the best concrete cycles $\{(A, D, 3), (D, A, 5)\}$, $\{(B, F, 6), (F, B, 8)\}$, and $\{(E, C, 8), (C, E, 4)\}$;

1.c : Performed elementary bilateral settlement of concrete cycles with amplification factor not lower than 4 ({(A,D,3),(D,A,5)} and {(B,F,6),(F,B,8)} added to configuration);

1.d : Calculation of $mamp = 22/4 = 5.5$;

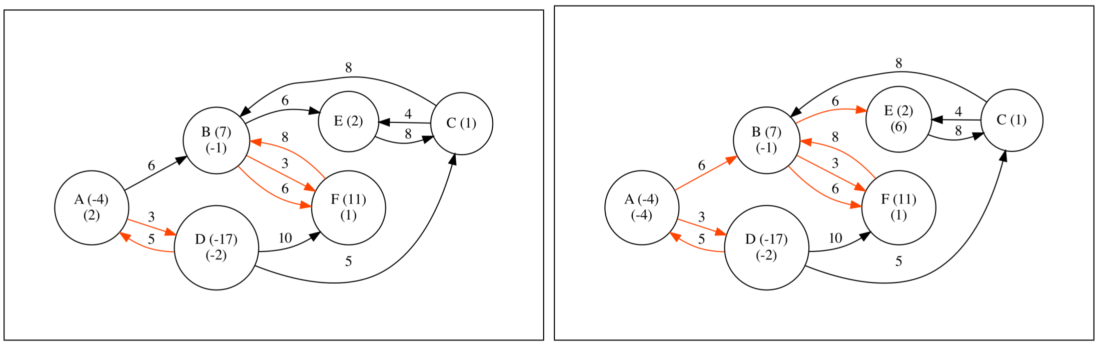


Figure 4.7: TinyCycles: $G5$-7 after 1st and 2nd iterations

In the left part of Figure 4.7, the graph is in the state after the first iteration of the main part of the algorithm. Below we can see how the state was reached:

2.a : picked the best concrete cycle $\{(E, C, 8), (C, E, 4)\}$ but not settled because $\alpha(\{(E, C, 8), (C, E, 4)\}) < mamp$;

2.b : took a forward augmenting path from node B by adding edge $(B, F, 3)$ to C;

2.c : $mamp$ is not decreased since one edge is settled;

During the next 3 iterations, no edge was added to the configuration due to a high minimum amplification factor. As a result, $mamp$ is decreased from 5.5 to 5.12. After the 5th iteration, two more edges are settled, as can be seen in the right part of Figure 4.7. Below we have the steps the algorithm went through to reach this state.

2.a : picked the best concrete cycle $\{(E, C, 8), (C, E, 4)\}$ but not settled because $\alpha(\{(E, C, 8), (C, E, 4)\}) < mamp$;

2.b : took a forward augmenting path from node A by adding edges $(A, B, 6)$ and $(B, E, 6)$ to C;

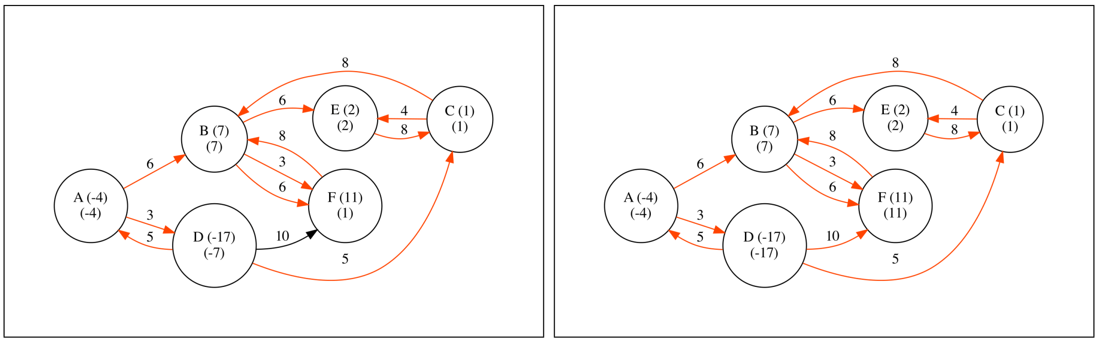


Figure 4.8: TinyCycles: G5-7 after 3rd and 4th iterations

2.c : *mamp* is not decreased since two edges are settled;

Iteration number 6 also only decreases *mamp* from 5.12 to 4.81 and does not add a single edge to the configuration. In the next iteration, 4 more edges are settled as depicted in the left part of Figure 4.8. Let's see the steps the algorithm went through:

2.a : picked the best concrete cycle $\{(E, C, 8), (C, E, 4)\}$ but not settled because $\alpha(\{(E, C, 8), (C, E, 4)\}) < mamp$;

2.b : took a forward augmenting path from node E by adding edge $(E, C, 8)$ and split into two paths from node C by adding edges $(C, B, 8)$ and $(C, E, 4)$ to C. Took a backwards augmenting path from node C by adding edge $(D, C, 5)$;

2.c : *mamp* is not decreased since two edges are settled;

Before the last edge is added to the configuration six more iterations are executed to decrease the minimum amplification factor from 4.81 to 3.32. After that, the algorithm performs the last iteration as follows:

2.a : no concrete cycle found;

2.b : took a backwards augmenting path from node F by adding edge $(D, F, 10)$;

2.c : *mamp* is not decreased since one edge is settled;

The right part of Figure 4.8 depicts the state of the graph after the last iteration (all the edges have been added to the configuration).

Now, we would like to compare how this algorithm performs in comparison with the IterativeBruteForce algorithm. For this purpose, we will run both of them to process the graph $G5$-7 and will draw curves that represent the correlation between the per cent of settled debt and the resulting amplification factor. Figure 4.9 shows 3 curves. The red one shows the correlation between settled debt and the amplification factor produced by the IterativeBruteForce algorithm. Visualization of each step of this execution is available in Appendix A. The blue and the green ones show the result for the TinyCycles algorithm for the cases when the **CAT** is set to 4 and 5 respectively and **APMD** is equal to 2. While the blue line is the result of the execution we just went through step by step, the visualization of the execution that produced the green line is available in Appendix B.

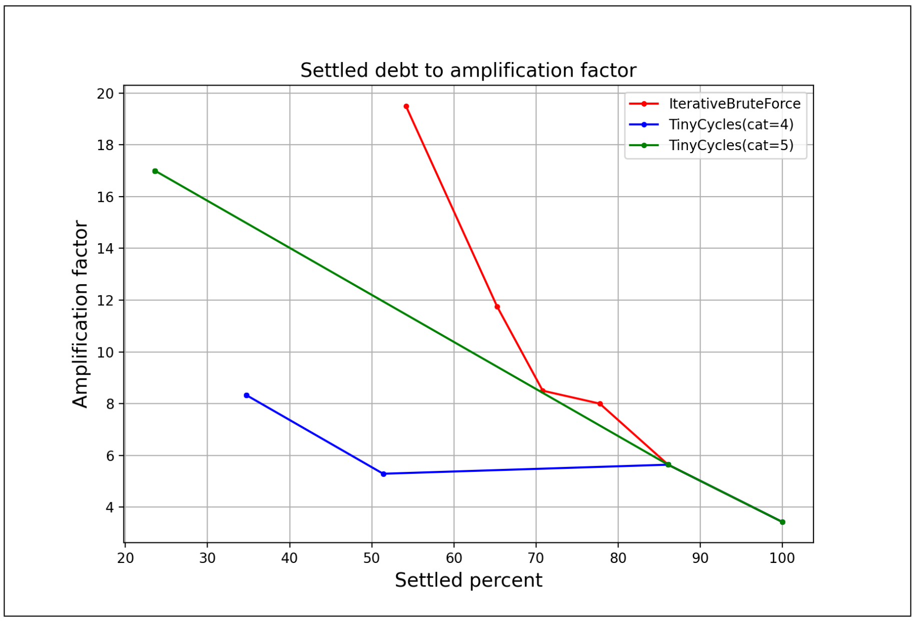


Figure 4.9: TinyCycles vs IterativeBruteForce on $G5$-7

Clearly, the IterativeBruteForce algorithm shows the best result out of those three but why the other two are so bad? Let's first compare the output of the TinyCycles algorithm for two different configurations. The blue line shows the execution we just went through step by step and it didn't seem to be so bad. The green line seems to show a better result. However, it has only three points, the first one of which gives quite a good amplification factor but with a relatively small amount of settled debt. The second point is too far from the first one, it gives a relatively small amplification factor with a big percentage of total debt settled. The key difference between the two executions of the TinyCycles algorithm is that the blue execution settled two cycles in the begging while the green only one. On the other hand, what is interesting is that the IterativeBruteForce algorithm settles a cycle of length three in the very beginning to achieve such a good ratio between the settled debt and the amplification factor. That is something that the TinyCycles algorithm clearly doesn't do.

As we can observe on this small graph, the presented algorithm is quite far from the results produced by the IterativeBruteForce algorithm, let alone BruteForce. There are a lot of improvements that could be made, such as:

- analysing cycles of length three or other structures that would have a positive influence on the amplification factor;
- giving priority to some paths over the others (e.g. by analysing the benefits of settling a specific edge, path, direction, etc.);
- optimizing the way the minimum amplification factor is calculated (e.g. by by taking into account how many edges were added to a configuration during the previous iteration);

In the next section, we will present a variant of the algorithm to see what influence a single modification can have on the results we obtain.

4.4 TinyCyclesV2 algorithm

The modification that we introduce is to make the algorithm capable of taking into account the cycles of length three. It is a relatively small modification. We will just show the results and will not explain what exactly was changed in the algorithm because it is rather trivial. This version of the algorithm is called "TinyCyclesV2".

To have a good idea of what influence this modification has on the execution, let's compare it with the execution of the original algorithm and the IterativeBruteForce algorithm on the graph $G5$-7. In Figure 4.10 we can see the same two executions we saw before and the execution of the TinyCyclesV2 algorithm with **CAT** equal to 5 and **APMD** equal to 2 (the visualization of the execution can be found in Appendix C).

At first, might seem to be quite weird, but if to take a closer look we can observe a few things. First of all, the weirdness comes from the fact that the line starts somewhere in the middle (horizontally) of the canvas which is, in fact, close to where the IterativeBruteForce line starts. It means that after the pre-processing stage and the first iteration, a high per cent (around 66-67%) of the total debt was settled. Even though the amplification factor is lower than for the similar amount settled by the IterativeBruteForce algorithm, it is already better than the amplification factor produced by the TinyCycles algorithm during the two previous executions. The second thing is that the last two iterations are the same as for the IterativeBruteForce algorithm what also can be considered as a positive change. Moreover, for around 71% of the settled debt, we can clearly see that the TinyCycles2 algorithm produces a better amplification factor than the IterativeBruteForce one.

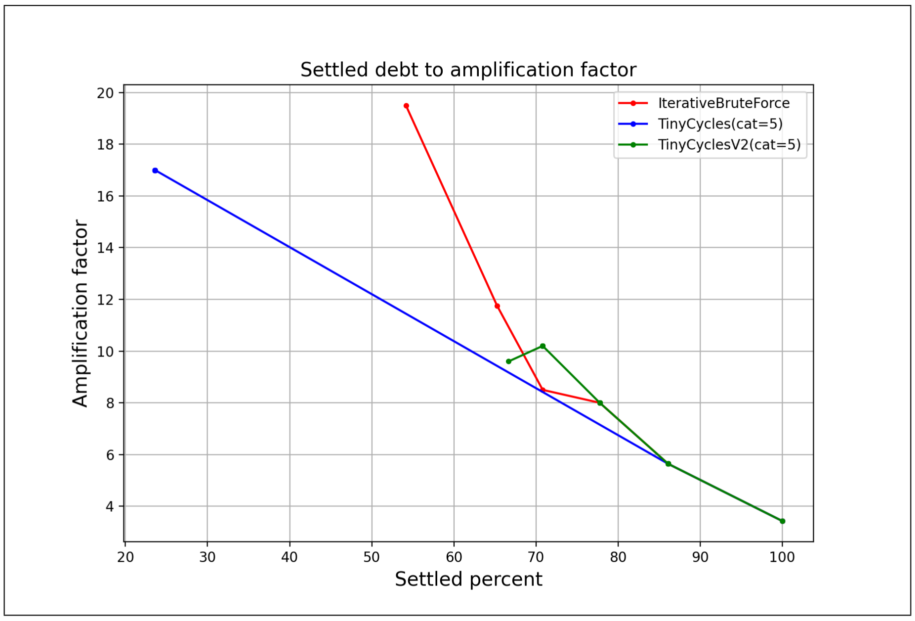


Figure 4.10: TinyCyclesV2 vs TinyCycles vs IterativeBruteForce on $G5\text{-}7$

This confirms our earlier claim stating that our IterativeBruteForce algorithm does not produce the highest amplification factor for every possible amount of settled debt, but only the first amplification factor is truly the highest and every next iteration is just an incrementation. From this, we could conclude that for an arbitrary graph, the best amplification factor for 30 per cent of the settled debt is unlikely to be based on the same set of edges that produces the highest amplification factor for 20 per cent of settled debt. This means if one is interested in finding the highest possible amplification factor for a certain amount of settled debt then it does make more sense to tackle the goal directly rather than following an iterative approach.

We claim that it is probably not optimal to follow an iterative process for getting the highest amplification factor for a certain amount of settled debt which actually correlates with the fact that there is no simple solution for an NP-complete problem. However, we will continue with this approach for the moment. We will try to explore the problem more deeply to see if this approach is viable at least for some specific scenarios and to improve the results we obtain.

Observing a positive change on such a small and limited dataset is not really enough to be sure that there is a positive change. As we mentioned before, we have in our possession a real dataset that reflects a topology of at least one real financial network. It consists of 38,334 companies(nodes) and 145,540 invoices(edges). We will call this dataset/graph G_{r1}. Let's see how both TinyCycles and TinyCyclesV2 algorithm perform on this graph. When running these two algorithms all the parameters have their default

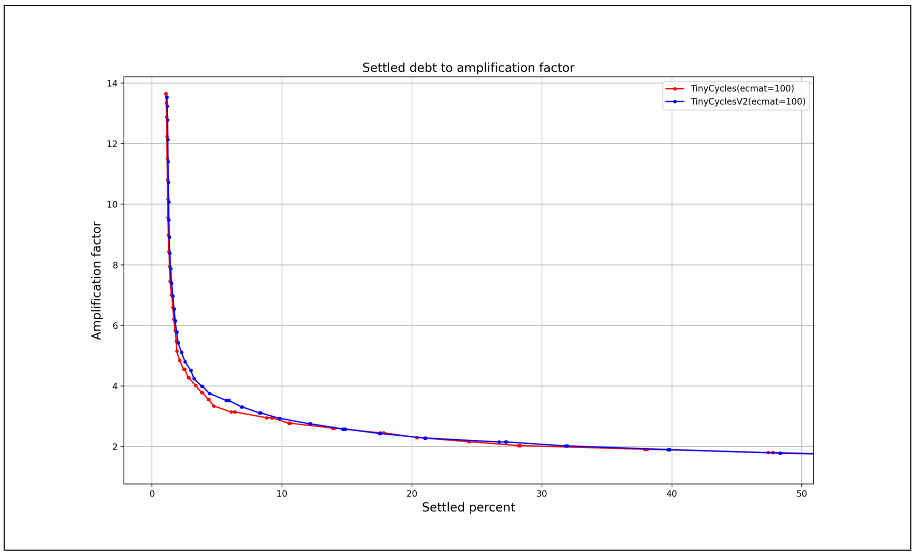


Figure 4.11: TinyCyclesV2 vs TinyCycles on G_{r1}

values only **ECMAT**[3] is equal to 100. In Figure 4.11, we can observe that the modified algorithm shows better results, especially in the range of settled debt between around 2 and 8 per cent. Around 6 per cent, the amplification factor differs by approximately 0.5 which is significant in the context of thousands of invoices. What is interesting is that the execution time for the TinyCycles algorithm on the given graph was 1 minute 19 seconds while for TinyCyclesV2 it was 1 minute 27 seconds. The difference can be noticed but it is neglectable for graphs of such scale.

As we already learned that the IterativeBruteForce algorithm does not provide the best amplification for each amount possible, let's compare the result of the TinyCyclesV2 algorithm also with the BruteForce. Such a comparison should show us how far is TinyCyclesV2 from the optimal results. In Figure 4.12 we can see the executions of IterativeBruteForce and TinyCyclesV2 algorithms as well as the curve produced by the BruteForce algorithm.

The red line shows what is the best amplification factor for any possible amount of settlement. From this visualization, we can see that the TinyCyclesV2 algorithm achieved the highest possible amplification factor at its second iteration but did not reach the point that is the result of the second iteration of the IterativeBruteForce algorithm. On the other hand, the IterativeBruteForce algorithm at first reaches the highest amplification factor at its second iteration and then plunges and does not capture the highest amplification factor at its second iteration. If we recall that both of these algo-

[3]The parameter defines the maximum number of settled edges during a single iteration causing the decrease of the minimum amplification factor. If more edges are settled, the minimum amplification factor is not decreased. It corresponds to TC_EDGE_COUNT_MIN_AMPLIFICATION_THRESHOLD in the configuration file.

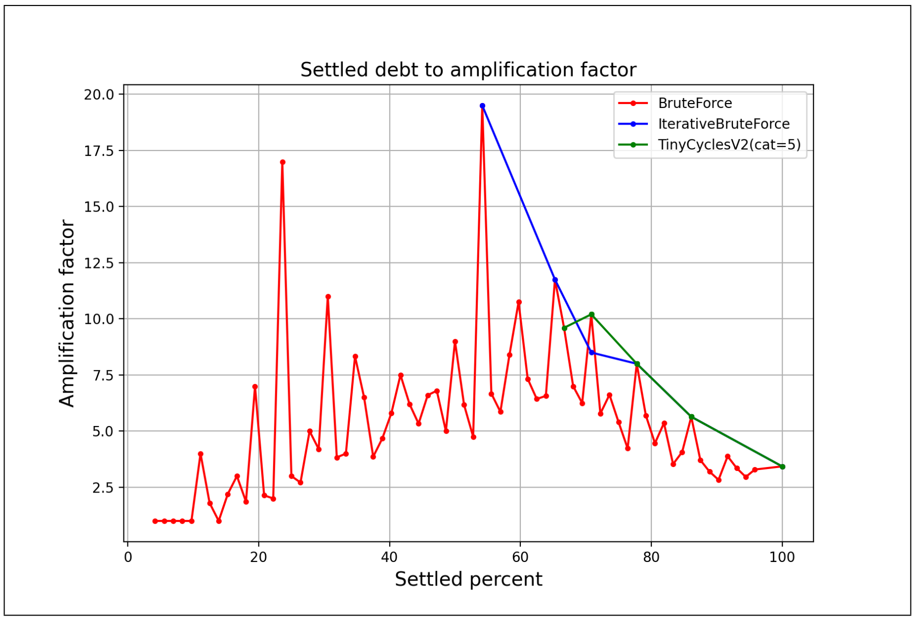


Figure 4.12: BruteForce vs IterativeBruteForce vs TinyCyclesV2 on G_{r1}

rithms are iterative (e.g. every next configuration is an extension of the previous one), we can guess that what prevents any of these algorithms from getting the highest amplification factor at each step is the presence of some edges in their previous configuration that worsen the result of the next iteration. Indeed, if we check the visualizations of these two executions (specifically their 2nd steps), we would see that the configuration of the IterativeBruteForce algorithm includes the edge $(D, A, 5)$ while the configuration of the TinyCyclesV2 algorithm does not include that edge. It tells us that to achieve the highest amplification factor for a certain amount of settled debt we should "skip" some edges on our way to it. The question here is how could we do it?

Since the TinyCycleV2 algorithm has a few parameters, let's try to change at least one to see what effect it would have on the results we get. The parameter that plays an important role in the algorithm is **CAT**. It is used in the pre-processing stage for getting the very first cycles to settle, which results in the initial minimum amplification factor. The parameter sets the lower bound for the amplification factor of a cycle to be settled. During the main iterations of the algorithm, the role of this lower bound plays the minimum amplification factor, which again initially comes from the pre-processing stage. So let's run the algorithm with **CAT** set to 3. The value of **ECMAT** equals 100 as before.

In Figure 4.13 we see two curves representing two executions of the TinyCyclesV2 algorithm with different values of cycle amplification threshold. The red line is the execution we saw before while comparing it with the result of the TinyCycles algorithm. The blue one is the new execution for which the **CAT** is equal to 3. We can

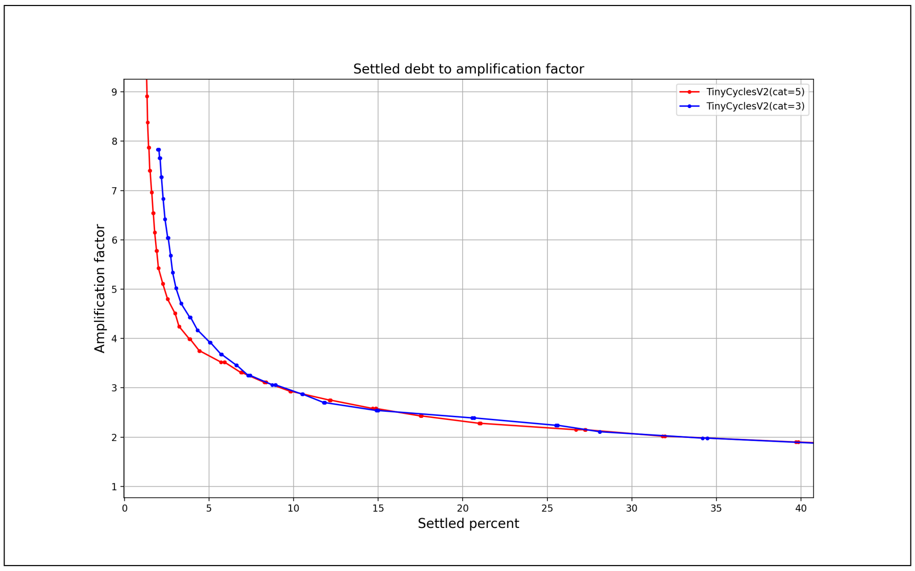


Figure 4.13: TinyCyclesV2 on G_{r1}: cat=5 vs cat=3

observe that since the initial value for the minimum amplification factor is lower (almost 8 in comparison with 14), the second execution shows much better results for settled per cent of between around 1.5 and 10. Then the blue line plunges a bit from 10 to 15 and again shows a better result from 15 to 28 per cent.

Besides the mentioned dataset G_{r1}, we have one more real dataset called G_{r2}, which has 41446 nodes and 163624 edges. The graph represents the same network, but just for another period of time. We will run the TinyCyclesV2 algorithm with the same configurations as above to see if the influence of the change in the configuration is still the same or at least similar. The chart representing the two executions is given in Appendix D. By seeing the second chart, we can say that the change of the configuration has the same influence, though the change in the result is not as dramatic as in the first case. Also, two more charts for the executions with the same configurations on randomly generated graphs are present in Appendix D.

Clearly, for achieving the goal of finding the best amplification factor, we need to better understand what edges to add to a configuration and which to skip. Potentially we could be adding edges and removing them later if we see that some have a negative influence on the resulting amplification factor. Overall, there is much to be explored and we leave this problem for further research.

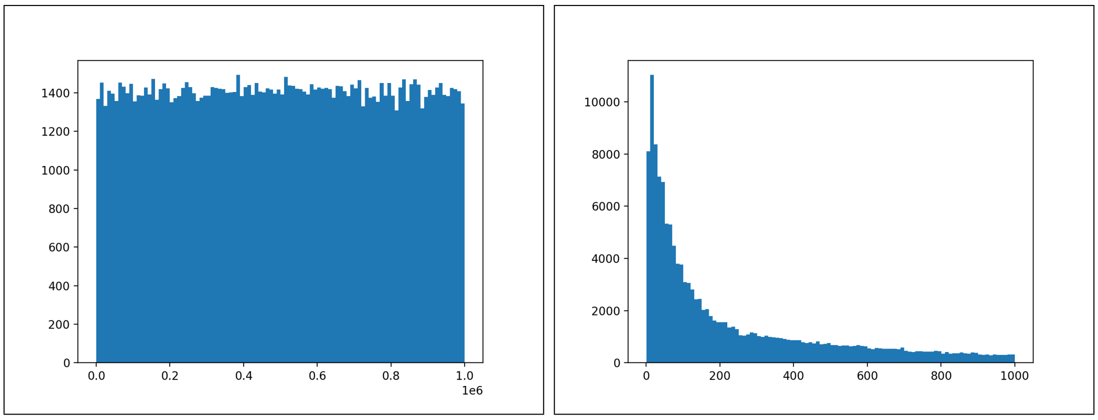


Figure 4.14: Distribution of edge weights for G_{uniform} and $G_{\text{ps-weibull}}$ graphs

4.5 Data shape and limits

What about the distribution of weights in a graph? Does it have any influence on the settlement process? As we had a topology of another real dataset, consisting of 26826 nodes and 140871 edges, we completed it with weights. Based on that topology, we created a dataset by adding uniformly distributed weights, which we call G_{uniform}. The second dataset got the pseudo-weibull distribution of edge weights, we will name it $G_{\text{ps-weibull}}$. The distribution of weights in the mentioned datasets are displayed in Figure 4.14.

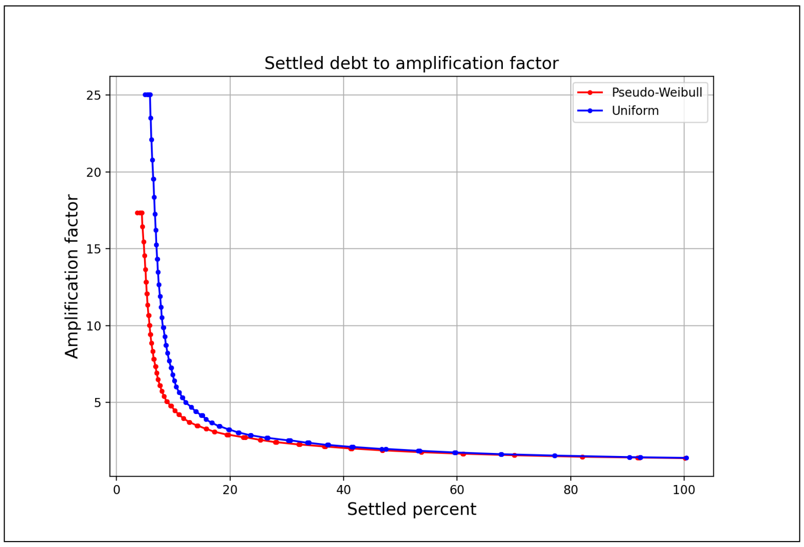


Figure 4.15: TinyCyclesV2: G_{uniform} vs $G_{\text{ps-weibull}}$

We will run the TinyCyclesV2 algorithm on these two datasets to compare the results. All the parameters have their default values only **ECMAT** is equal to 100. The chart representing the curves of the two executions is displayed in Figure 4.15. As we can see, the distribution of edges has an influence on how successful the settlement process is. For the graph with uniform distribution of edges the results produced by the TinyCyclesV2 algorithm are much better. We believe the reason for this is a higher probability of having edges with similar weights close to each other in G_{uniform} that results from the fact that all the weights are uniformly distributed. Taking this into account, we could probably benefit from splitting a graph into multiple graphs in such a way that each graph would have only edges with weights in a fixed range. This could possibly help with processing of big graphs.

Since we already mentioned big graphs, it would be interesting to see the limits of the algorithm we presented. Especially how much time we need to process a graph of a certain size. For this purpose, we will generate a few datasets for different numbers of nodes and edges and run the TinyCyclesV2 algorithm on them to measure the execution time. We are aware that randomly generated datasets might not be good enough to represent the time needed for processing real networks. However, they would give us an intuition of how much the execution time changes when the size of a graph rises.

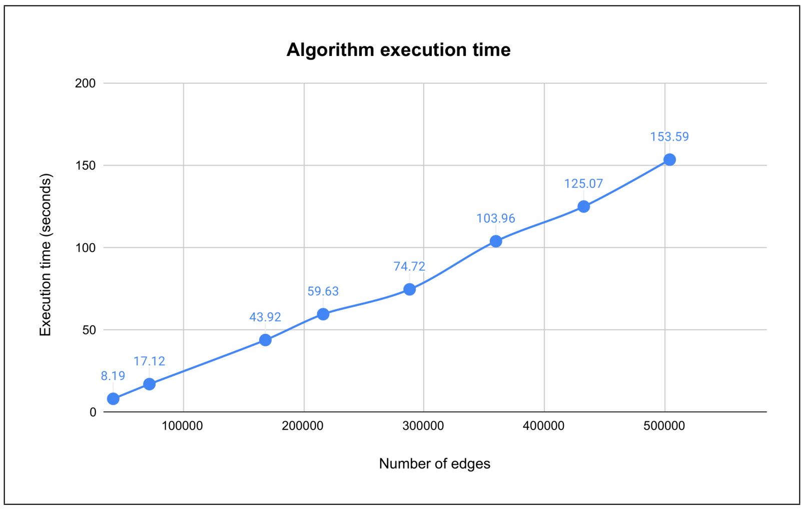


Figure 4.16: TinyCyclesV2 execution time

In Figure 4.16 we can see how the execution time of the TinyCyclesV2 algorithm changes when the number of edges in a graph changes from 10,000 to 500,000. In the executions that were used for the production of this chart, we settled the whole graph and the value of **ECMAT** parameter was set to 100. Of course, the execution time will vary depending on the configuration, but we can see that the execution time is not exponential. Since the processing of 500,000 edges takes only two and a half minutes, we could definitely process even much bigger graphs. When the execution time becomes an issue, we could try splitting a graph into multiple graphs by edge weight or by creating subgraphs based on already existing clusters that can be found using specific algorithms.

Summarising all of the above, we can say that the TinyCycles and TinyCyclesV2 algorithms produce interesting results and they are relatively quick to execute. However, these algorithms can be definitely optimized for getting better results and execution time. Moreover, there is a need for more research to see how far the results produced by these algorithms are from the optimal results.

Note: The access to the code of the project can be provided on request.

Chapter 5: Conclusion

In this work, we analysed the problem of integral mutual debt cancellation and outlined the main difficulties of finding a viable solution. As a result, we developed an algorithm for mutual debt reduction in financial networks and demonstrated how it works. The key idea behind our approach is taking into account different properties of directed multigraphs and financial networks. This way we base the algorithm on the method of settling cycles of lengths two and three as well as taking augmenting paths. Clearly, the algorithm does not produce optimal results, but it has a good execution time on graphs with hundreds of thousands of edges.

We analyzed how different parameters influence the settlement process and showed that with the same topology but different distribution of edge weights the results can be significantly different. Of course, there is a need for more research to learn how different topologies influence the settlement process. We showed that getting the highest amplification factor in an early stage is likely to keep us from getting a good amplification factor later due to the nature of the iterative algorithms. However, we think that there are a lot of improvements that can be made to achieve better results even with an iterative algorithm.

While working on the project we deepened our knowledge about financial networks and graph theory in particular. For future work, we plan to do more research on finding or approximating the optimal settlement curve for bigger graphs and try to improve our algorithm and develop new ones in order to get as close as possible to the optimal results. The next possible steps for developing new algorithms would be finding and analysing clusters of nodes with a relatively high number of edges, trying to remove edges from a configuration instead of only adding them, analysing local criteria when deciding to add an edge to a configuration such as an amplification factor of a single edge.

Bibliography

[1] V. Gazda, D. Horváth, M. Rešovsky, “An Application of Graph Theory in the Process of Mutual Debt Compensation,” *Acta Polytechnica Hungarica*, vol. 12, (3), 2015.

[2] G. Iosifidis et al, “Cyclic motifs in the Sardex monetary network,” *Nature Human Behaviour*, vol. 2, (11), pp. 822-829, 2018.

[3] M. Klein, “A Primal Method for Minimal Cost Flows with Applications to the Assignment and Transportation Problems,” *Management Science*, vol. 14, (3), pp. 205-220, 1967.

[4] M. M. Güntzer, D. Jungnickel and M. Leclerc, “Efficient algorithms for the clearing of interbank payments,” *European Journal of Operational Research*, vol. 106, (1), pp. 212-219, 1998.

[5] C. Pătcaş, “The debts’ clearing problem’s relation with complexity classes,” *Acta Mathematica Academiae Paedagogicae Nyíregyháziensis*, 28(2):217–226, 2012.

[6] D. B. Johnson, “Finding All the Elementary Circuits of a Directed Graph,” *SIAM Journal on Computing*, vol. 4, (1), pp. 77-84, 1975.

[7] L. R. Ford and D. R. Fulkerson, “Maximal Flow Through a Network,” *Canadian Journal of Mathematics*, vol. 8, pp. 399-404, 1956.

[8] R. Tarjan, “Depth-First Search and Linear Graph Algorithms,” *SIAM Journal on Computing,* vol. 1, (2), pp. 146-160, 1972.

[9] NetworkX — NetworkX documentation `https://networkx.org/` 2021.

[10] Graphviz - Graph Visualization Software `https://graphviz.org/` 2021.

[11] Matplotlib: Python plotting — Matplotlib 3.4.2 documentation `https://matplotlib.org/` 2021.

Appendices

Appendix A: IterativeBruteForce execution visualization

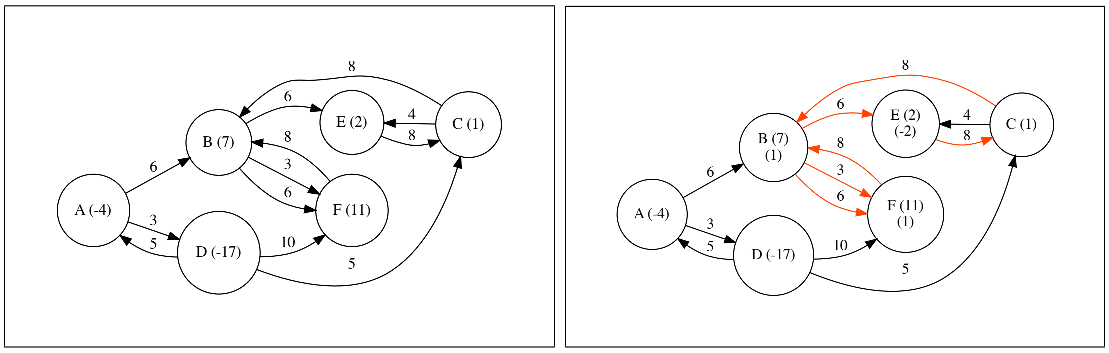


Figure A.1: IterativeBruteForce: initial graph $G5$-7 and its state after step 1

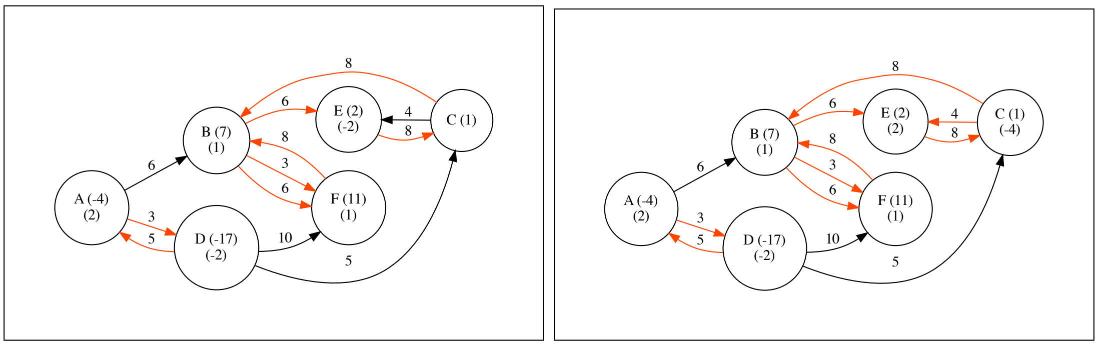


Figure A.2: IterativeBruteForce: graph $G5$-7 after steps 2 and 3

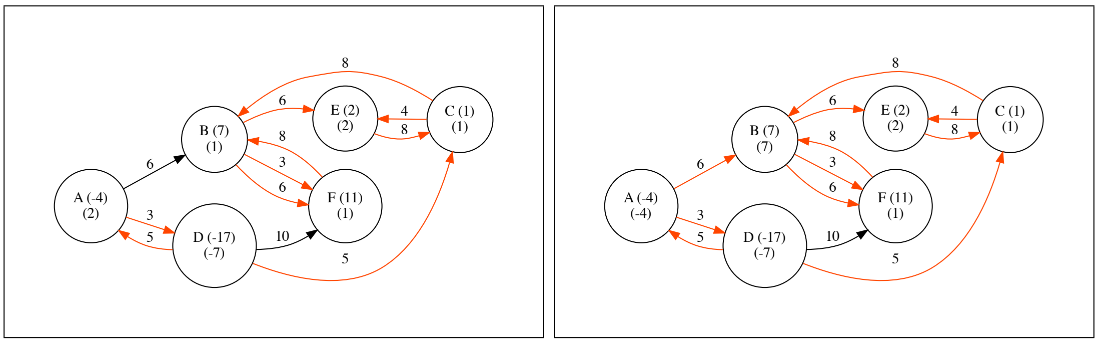


Figure A.3: IterativeBruteForce: graph $G5$-7 after steps 4 and 5

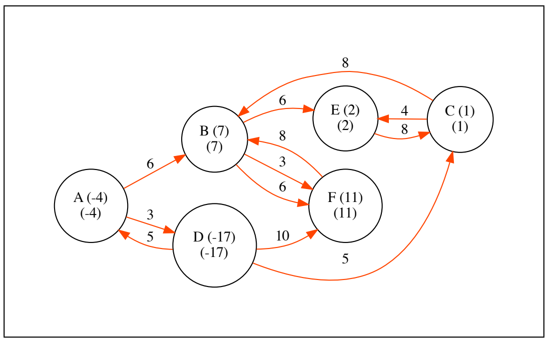
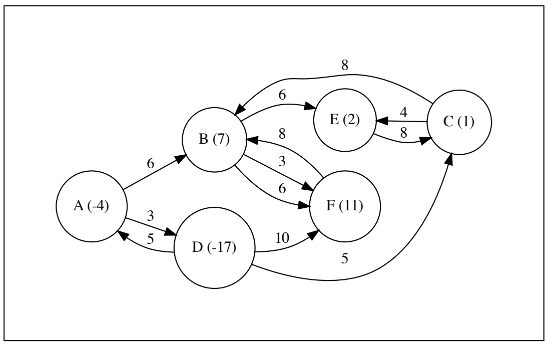


Figure A.4: IterativeBruteForce: graph $G5$-7 after step 6

Appendix B: TinyCycles execution visualization (CAT=5)



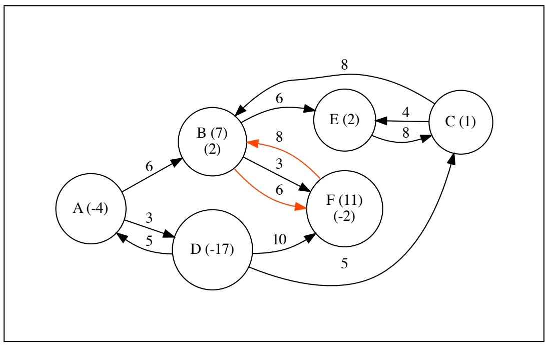
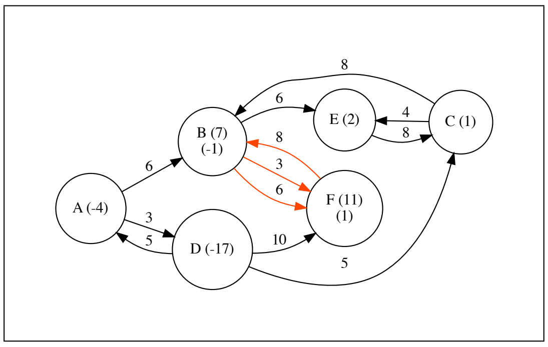


Figure B.1: TinyCycles(**CAT**=5): initial and pre-processed graph $G5$-7



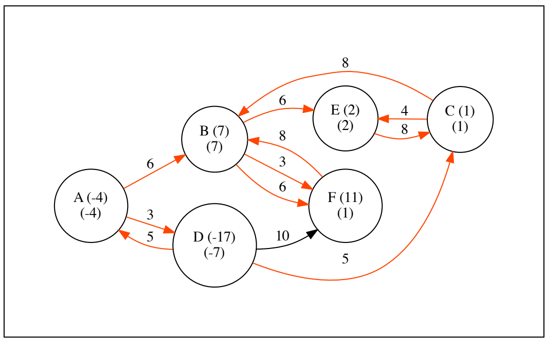


Figure B.2: TinyCycles(**CAT**=5): graph $G5$-7 after steps 1 and 2

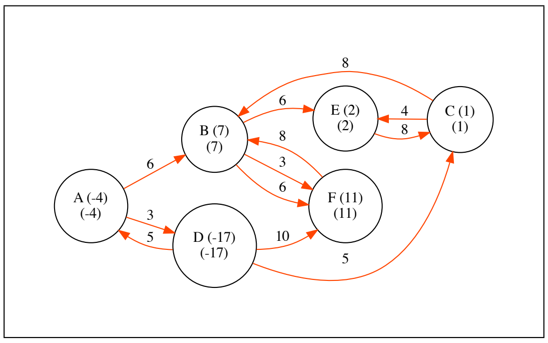


Figure B.3: TinyCycles(**CAT**=5): graph G5-7 after step 3

Appendix C: TinyCyclesV2 execution visualization (CAT=5)

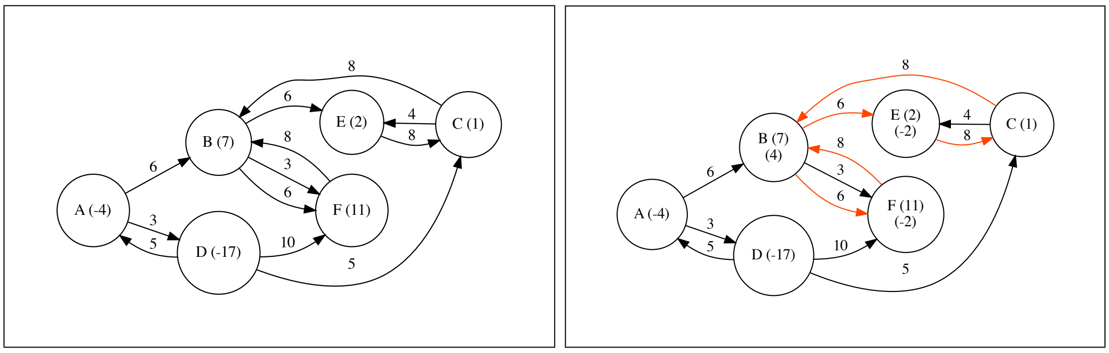


Figure C.1: TinyCyclesV2(**CAT**=5): initial and pre-processed graph $G5$-7

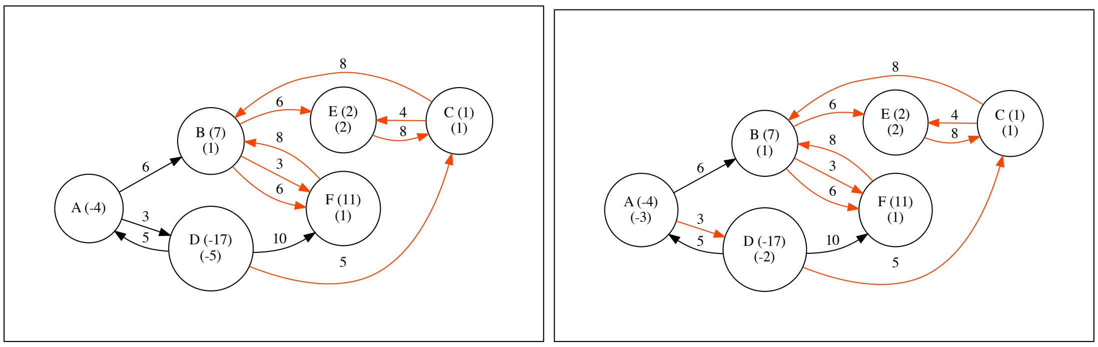


Figure C.2: TinyCyclesV2(**CAT**=5): graph $G5$-7 after steps 1 and 2

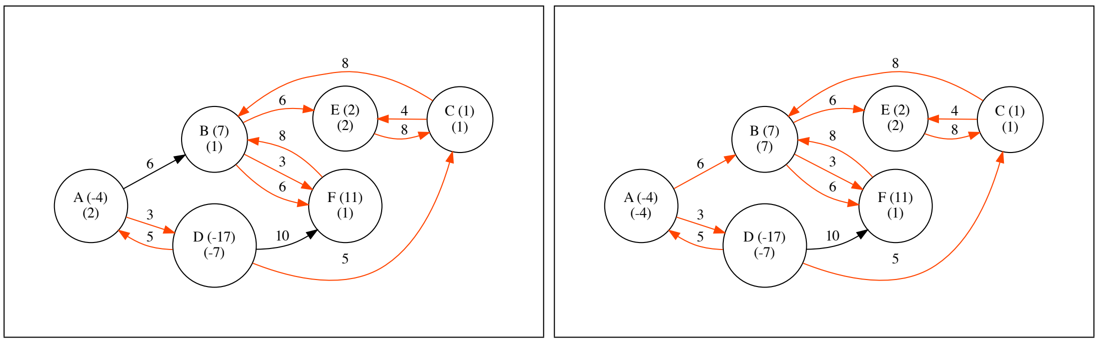


Figure C.3: TinyCyclesV2(**CAT**=5): graph G5-7 after steps 3 and 4

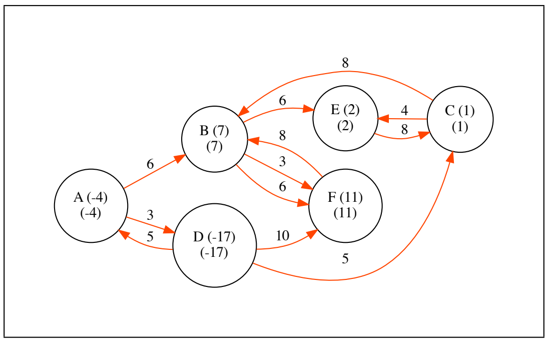


Figure C.4: TinyCyclesV2(**CAT**=5): graph G5-7 after step 5

Appendix D: TinyCyclesV2 results on other datasets

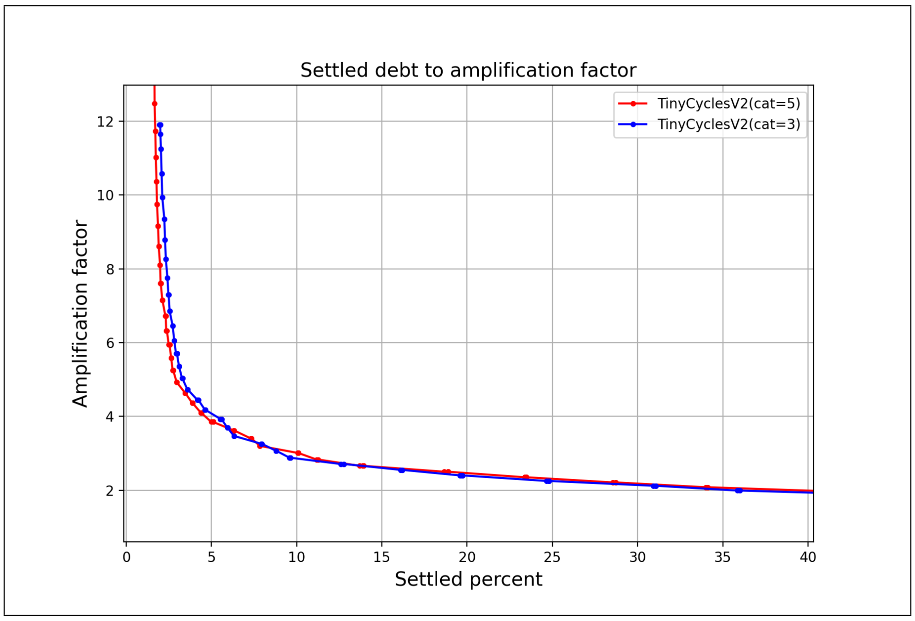


Figure D.1: Graph G_{r2}: TinyCyclesV2(**CAT**=5) vs TinyCyclesV2(**CAT**=3)

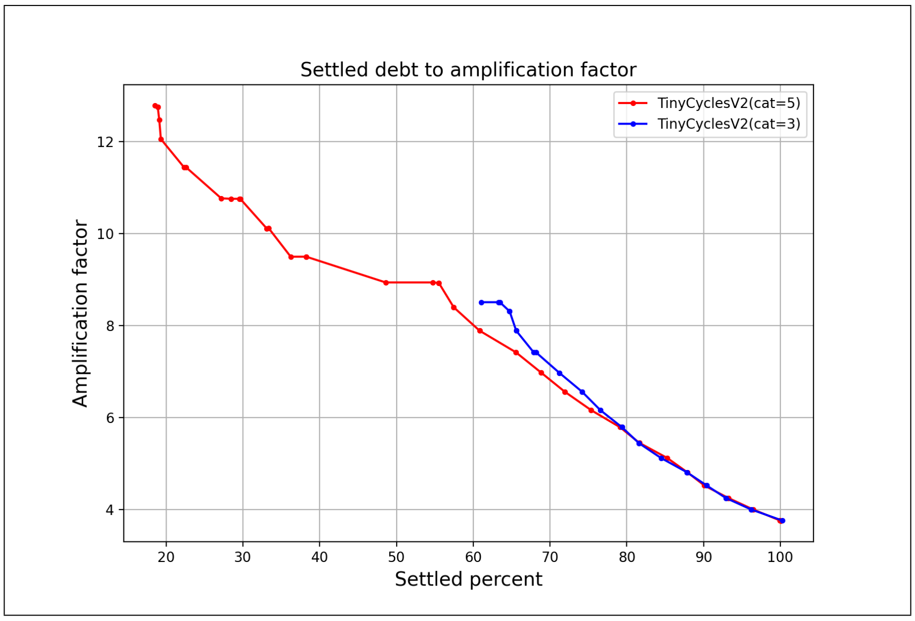


Figure D.2: TinyCyclesV2(**CAT**=5) vs TinyCyclesV2(**CAT**=3), **ECMAT**=25: randomly generated graph with 1000 nodes and 4123 edges

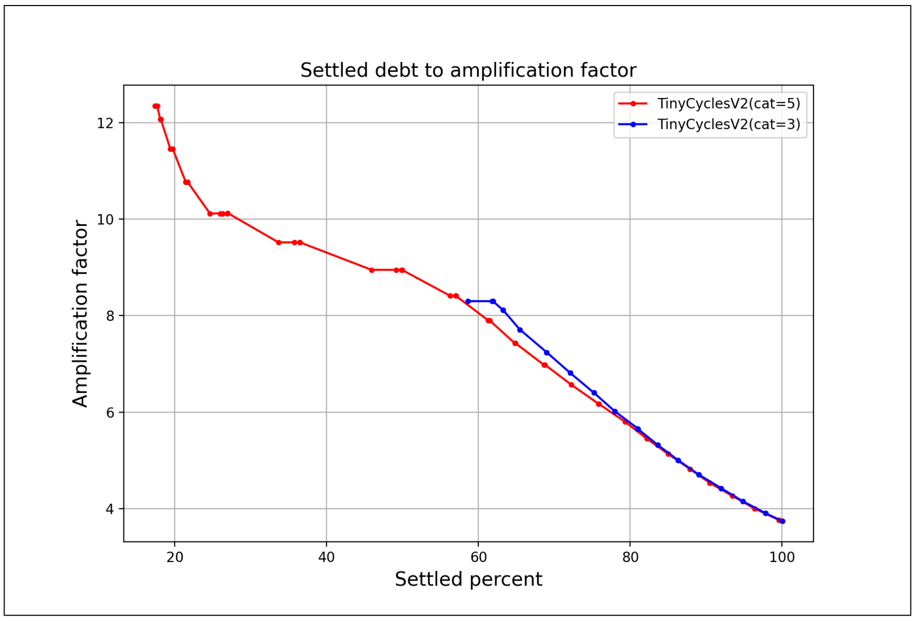


Figure D.3: TinyCyclesV2(**CAT**=5) vs TinyCyclesV2(**CAT**=3), **ECMAT**=100: randomly generated graph with 10000 nodes and 41958 edges