NAP SuSe 2023:
Virtual Channel Optimization in Payment Channel Networks

Yannik Kopyciok, kopyciok@campus.tu-berlin.de

supervisor: Iosif Salem, iosif.salem@inet.tu-berlin.de

May 2023 – July 2023

1 Abstract

Payment channel networks (PCNs) emerged as a viable solution for several limitations that appeared while the demand for cryptocurrencies increased. PCNs offer fast, scalable, and secure ways of transferring funds by enabling off-chain transactions but they still show some drawbacks in terms of cost and security. A transaction that has to pass multiple intermediaries might suffer from high forwarding fees and intermediaries have the potential to be vicious. To tackle those downsides, the possibility of creating virtual channels (VCs) is currently researched. VCs show great advantages but the question arises when and where it is beneficial to open one, and when not. To answer this question an integer linear program (ILP) has been designed but as the underlying question is NP-hard, the ILP itself shows exponential growth.

2 Introduction

Payment Channel Networks are one of the most promising solutions for increasing blockchain throughput. For example, in the Bitcoin Lightning Network (LN) a node can open a PC with another node with one on-chain (on Bitcoin) transaction, do an arbitrary number of instant (off-chain, not on Bitcoin) payments between each other, and finally close the channel with the new balance with another on-chain transaction. LN is the network formed by all such channels. Two nodes in LN can safely make transactions even if they are not directly connected. For example if Alice and Bob, as well as Bob and Carol share a channel then Alice can pay Carol by paying a fee to Bob (given that Alice and Bob have sufficient funds and they conform to the payment's hash time-lock contract).

VCs can potentially reduce fees to intermediaries. In the example above, Alice and Carol can pay a fixed amount to Bob and build a direct Alice-Carol virtual channel above the Alice-Bob-Carol payment channels. Then, Alice can build a (recursive) VC with David using the Alice-Carol VC and the Carol-David PC. Given the cost of opening a virtual channel, two users (e.g. Alice and Carol or David) must decide whether it is preferable to pay the transaction routing fees or to open a VC. This question was addressed in [3]. The authors proposed an integer linear program for computing the optimal virtual channel establishment given a set of transactions, as well as a greedy and fast heuristic for solving the same problem more efficiently. Their work also proposes how to use VCs to mitigate known LN attacks.

This project aims to implement and test the ILP presented in [3]. The following report is structured as follows: In section 3 we provide some necessary theoretical background. We continue in section 4 with an overview of the whole project, stating its goals, and presenting the results of each work package. Section 5 describes the architecture and the implementation of the ILP, followed by section 6 which includes a lessons learned part and opens ideas for potential future work. We conclude in section 7 with a short summary and final notes.

3 Background

A **Sparse matrix** is a data structure that takes advantage of the appearance of many zeros or neglectable entries in a matrix by storing only the non-zero values or the entries that are needed for the computation including their position in the matrix. Thereby, less memory space needs to be allocated and other mathematical operations can leverage the data structure to perform efficient computations.
An **Integer Linear Program** refers to a mathematical optimization problem that only consists of linear constraints and integer variables. Similar to a common optimization problem it finds the best solution to either maximize or minimize an objective function.
The **matrix form** is a way of modelling the ILP so that we have a variable vector x with a corresponding coefficient vector c both of dimension (n x 1), a matrix A of dimension (m x n) representing one constraint in each row where the column position refers to the respective variable of the objective function, and a rhs- or boundary-vector of (m x 1) so that we get following model:

$$\begin{aligned} &\min c^T \cdot x \\ &\text{such that} \\ A \cdot x &\geq b \\ x &\geq 0 \end{aligned}$$

Gurobi is a company offering a state of the art linear optimization solver with many language integrations. The project will use their linear programming solver to solve a minimization problem. The solver brings the above together and leverages the structure of a sparse matrix to perform an efficient optimization.

4 Project Overview

The first work package (WP) includes the implementation of the ILP in Python, using the linear programming solver Gurobi (GurobiPy [4]). The second work package includes testing and evaluating the ILP with feasible instances, using both small graphs (random or custom-made) and subgraphs of LN snapshots. The third work package (its size depends on the time left in the project) includes approximations of the ILP to increase running time by sacrifizing accuracy. We will try two approaches: (i) reducing the universe of possible payment paths for each input transaction to a polynomial size and (ii) using a greedy LP relaxation and rounding rules. The results of WP3 three will be then compared to those from WP2. The deliverables of the project include (i) a github repository including the implementation, data, and the code for experimentation, (ii) a short report (current document) describing the project, the results, and a discussion of the project's findings, and (iii) a short presentation of your work in the end of the project.

4.1 WP1: Implementation

4.1.1 Theory & Intuition

As stated in the introduction, the project is based on the paper [3] and further improvements done to the presented model. The paper presents an Integer Linear Program in theory that solves the problem of when and where to create VCs in PCNs. The problem has the objective of minimizing the routing fees in the network while complying to several constraints. Furthermore, in future work including this project other benefits of VC establishment are outlined including security and privacy concerns in the network. Figure 1 gives you a glimpse of what the ILP produces. The input is an undirected graph with 13 nodes and 26 PCs and a set of five unique transactions which are repeated once resulting in ten transactions overall. The ILP optimizes VC establishment and due to the tractable nature gives us all VCs that have been created and all paths that the transactions took. Details in the Appendix Figure 7.

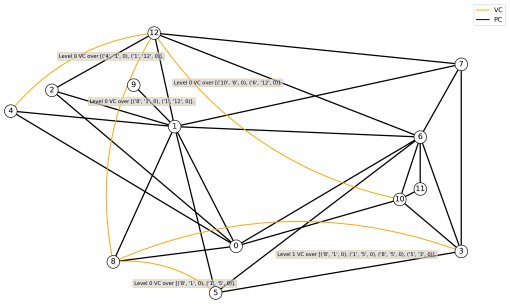


Figure 1: Sample network result

The objective of the ILP is to minimize the total cost which consists of the accumulated routing fees of the respected transactions and further the creation costs of all VCs. The creation cost furthermore is calculated by the proportional fee relative to the capacity of the VC and a base fee applicable to each VC that is created. The objective function has in total $|\mathbb{P}| + 2|E_{VC}|$ variables referring to each possible path of each transaction and two times the number of possible VCs. The set of path variables as well as the first set of VC variables are binary and refer to the chosen paths and the VC existence respectfully. The second set of VC variables are integer variables and state the capacity with which the VCs are opened. This makes the decision of the ILP computational tractable. Is a path variable or an existence variable set to 1, this path is taken and that VC is created with its corresponding capacity.

The constraints can be structured into four sets of constraints. It is important to mention that similar to the shape of the objective function, the number of constraints is fixed and can be derived from the input which we will follow on later.

- **C1:** The first set of constraints consists of $\mathcal{T}$ individual constraints and relates to the transactions whose routing will be optimized. This set makes sure that only one path is chosen for each transaction.
- **C2:** Due to the nature of a minimization problem, the ILP could simply decide to not route any transaction and therefore bring the routing fees and creation costs to zero. To avoid this obvious and non relevant solution, this constraint ensures a specified percentage of successful transactions.
- **C3:** Constraint three secures the validity of the optimized routing in terms of respective channel capacities. On the one hand, the constraint makes sure to keep the amount of routed transactions below the PC capacity and on the other hand, increases the VC to the necessary capacity. Therefore, this constraint has in total $|E_{PC}| + |E_{VC}|$ constraints.

- **C4:** The fourth and last set relates to the existence of a VC and has $|E_{VC}|$ many constraints. If a path is chosen this set ensures that the necessary VCs are created and further if a VC of a higher level is created, the underlying VCs exist as well.

The first set of constraints is a great example for the benefit of using a sparse matrix. If we take the first constraint from the first set we only look at the first transaction, meaning all path variables referring to the first transaction. All other path variables as well as the VC existence and capacity variables are negligible and therefore the first row of the matrix only has $|\mathbb{P}_t|$ instead of $|\mathbb{P}| + 2|E_{VC}|$ entries.

4.1.2 (Iosif's) ILP in $Ax \geq b$ form

Let $\mathbb{P} = \cup_{t=1}^{T} \mathcal{P}(s_t, d_t)$ be the set of all paths and $|\mathbb{P}|$ its cardinality. Let x be the variable vector. The first $|\mathbb{P}|$ positions of x, i.e., from 0 to $|\mathbb{P}| - 1$, refer to the variables $path_P(trans_t)$. The next $|E_{VC}|$ positions of x, i.e., positions $|\mathbb{P}|$ to $|\mathbb{P}| + |E_{VC}| - 1$, refer to the $exists\_vc_{ij}^k$ variables. The final $|E_{VC}|$ positions of x, i.e., positions $|\mathbb{P}| + |E_{VC}|$ to $|\mathbb{P}| + 2|E_{VC}| - 1$, refer to the variables $vc_{ij}^k.capacity$. Thus the size of x is $|x| = |\mathbb{P}| + 2|E_{VC}|$.

variable	**positions in x**	**type**
$path_P(trans_t)$	0 to $\|\mathbb{P}\| - 1$	binary
$exists\_vc_{ij}^k$	$\|\mathbb{P}\|$ to $\|\mathbb{P}\| + \|E_{VC}\| - 1$	binary
$vc_{ij}^k.capacity$	$\|\mathbb{P}\| + \|E_{VC}\|$ to $\|\mathbb{P}\| + 2\|E_{VC}\| - 1$	integer

The objective function is the following:

$$\min \Bigg[\sum_{pc_{ij} \in E_{PC}} \sum_{t=1}^{T} \sum_{P \in \mathcal{P}(s_t, d_t)} routing\_fee(t, P, pc_{ij}) \cdot path_P(trans_t) + \sum_{vc_{ij}^k \in E_{VC}} \left(\sum_{t=1}^{T} \sum_{P \in \mathcal{P}(s_t, d_t)} routing\_fee(t, P, vc_{ij}^k) \cdot path_P(trans_t) \right) + exists\_vc_{ij}^k \cdot vc_{ij}^k.base\_fee + vc_{ij}^k prop\_fee \cdot vc_{ij}^k.capacity \Bigg]$$

The constraints are in total $n_{constr} = T + |E_{PC}| + 5|E_{VC}|$. Thus the constraints matrix A has size $n_{constr} \times |x|$ and the bound vector b has size n_{constr}. Recall that we aim at defining the constraints in matrix form: $Ax \geq b$. Therefore, in the following we write each constraint inequality in the form:

$$A(i, 0)x[0] + \ldots + A(i, |x| - 1)x[|x| - 1] \geq b[i]$$

(C1) $\forall t \in \{1, \ldots, T\}$ (T constraints, from 0 to $T - 1$):

$$\sum_{P \in \mathcal{P}(s_t, d_t)} -path_P(trans_t) \geq -1$$

(C2) This is constraint T:

$$\sum_{t=1}^{T} \sum_{P \in \mathcal{P}(s_t, d_t)} trans_t \cdot path_P(trans_t) \geq c_{tr} T$$

(C3) We have one set of constraints for PCs and one for VCs.
$\forall pc_{ij} \in E_{PC}$ we have constraints $T+1$ to $T+|E_{PC}|$:

$$\sum_{vc^j_{ix} \in E_{VC} : vc^j_{ix} = (ch_{ij}, \bullet)} (-vc^j_{ix}.base\_fee \cdot exists\_vc^j_{ix} - vc^j_{ix}.prop\_fee \cdot vc^j_{ix}.capacity - vc^j_{ix}.capacity) +$$

$$\sum_{t=1}^{T} \sum_{P \in \mathcal{P}(s_t, d_t)} -routing\_fee(t, P, pc_{ij}) \cdot path_P(trans_t) +$$

$$\sum_{t=1}^{T} \sum_{P \in \mathcal{P}(s_t, d_t)} -In(pc_{ij}, P) \cdot trans_t \cdot path_P(trans_t) \geq -pc_{ij}.capacity$$

and $\forall vc^k_{ij} \in E_{VC}$ we have constraints $T+|E_{PC}|+1$ to $T+|E_{PC}|+|E_{VC}|$:

$$\sum_{t=1}^{T} \sum_{P \in \mathcal{P}(s_t, d_t)} -routing\_fee(t, P, vc^k_{ij}) \cdot path_P(trans_t) + \sum_{t=1}^{T} \sum_{P \in \mathcal{P}(s_t, d_t)} (-In(vc^k_{ij}, P) \cdot trans_t \cdot path_P(trans_t))$$

$$+vc^k_{ij}.capacity \geq 0$$

(C4) The last $|E_{VC}|$ constraints are from $T+|E_{PC}|+4|E_{VC}|+1$ to $T+|E_{PC}|+5|E_{VC}|$. The $|E_{VC}|$ constraints are $\forall vc^k_{ij} \in E_{VC}$:

$$(T+1)exists\_vc^k_{ij} + \sum_{t=1}^{T} \sum_{P \in \mathcal{P}(s_t, d_t)} -In(vc^k_{ij}, P) \cdot path_P(trans_t) \geq In(vc^k_{ij}, \mathcal{C})$$

4.1.3 Implementation structure

The implementation can be separated into three different tasks. First, we have to build the actual input networks with which we want to test the ILP on. Our network topology will be described in more detail in section 4.2 but as it can be individually adjusted for different tests and it is not part of the main project we will not discuss the graph generation further here. The second task is to build the graph specific inputs in terms of finding all VCs for the respective level of recursion that we allow and finding all paths for each transaction. Finally, we have to construct the vectors x, c, and b and the sparse matrix A which will be handed over to the Gurobi solver. The challenge in building the Gurobi input is to create the sparse matrix and corresponding vectors correctly. As shown in the Gurobi matrix example, we need to declare the non-zero entries with:

```
A = sp.csr_matrix((val, (row, col)), shape=(m, n))
```

The vectors val, row, col have the same length: the number of non-zero elements of A. The way they are parsed is: A[row[0], col[0]] = val[0], A[row[1], col[1]] = val[1], etc. The objective vectors c and x have the length of the highest entry of the col array and the bounding vector b, also referred to as right hand side vector or rhs vector, is the size of the highest entry of the row array which are the dimensions of the matrix A. As an example, take a first constraint, `x + 2 y + 3 z <= 4` we would implement:

val.add(1), row.add(0), col.add(0)
val.add(2), row.add(0), col.add(1)

val.add(3), row.add(0), col.add(2)
rhs[0] = 4

and for a second constraint `-x -y <= -1` we would implement:

val.add(-1), row.add(1), col.add(0)
val.add(-1), row.add(1), col.add(1)
rhs[1] = -1

This would gradually produce the lines:

```
val = np.array([1.0, 2.0, 3.0, -1.0, -1.0])
      row = np.array([0, 0, 0, 1, 1])
    col = np.array([0, 1, 2, 0, 1]) and
        rhs = np.array([4.0, -1.0])
```

4.2 WP2: Experimental evaluation

4.2.1 Objective:

Our objective in this project is to test the limits of the ILP with increasing graph sizes. We want to see until which size the ILP is still manageable and computes an optimal solution and when the execution time exceeds a reasonable timeframe or is not able to produce a solution at all.

4.2.2 Design:

Our evaluation process started with creating graphs that actually give us interesting results which in our case means are as close as possible to the actual appearance of the LN. The attached repository features a graph generator that builds random graphs that can be described as a miniature versions of the LN. The code is mainly written by Iosif Salem with slight adjustments to the number of PCs. Generally, the graphs are generated in a way that they consist of few nodes with high and several nodes with a low degree centrality. This represents the hubs and individuals in the LN. On the 29.08.2023 the LN featured a node to edge ratio of around 1:4.2 [2]. Our implementation uses miniature versions with a node to edge ratio of 1:2 which is based on two reasons. First, the LN itself consists of PCs which are directed whereas the graph that we construct and is handed to the program is of undirected nature. Second, smaller graphs tend to get dense more early and a dense graph is not interesting when we want to investigate the creation of VCs. If every transaction path is just sender to receiver, the creation of a VC is irrelevant.

4.2.3 Setup:

We constructed 25 sample graphs for every graph size to avoid outliers and to be able to draw a significant average. We started with a small graph size of 5 nodes and increased the nodes one by one. Along with the graphs we built a set of five unique transactions for each graph. The ILP runs for every graph and for every level of recursion successively, saving several parameters like execution time, number of potential paths and VCs, or graph diameter to a text-file which later is the data for creating an average over the samples.

4.2.4 Challenges:

One of the main challenges has been in finding a fitting graph structure. We started with fully randomized graphs and a node to edge ratio of 1:1. We continued with fully randomized graphs but a closer ratio to the LN ratio being 1:4. And finally reached a presentable graph structure that we described above. As mentioned, the main reasons for choosing this and not another version or ratio is the similarity to the LN. There might still be room for improvements in terms of separating the hubs differently or choosing a more diverse set of transactions to actually examine the behavior of VC creation. But the graphs chosen are close to the LN and work well for our goal of finding a limit in feasibility of the ILP.

4.2.5 Results:

For our final set of graphs we reached the limit of the ILP after 14 nodes. We were able to run most of the graphs for 15 nodes but the execution of two graphs did not finish and therefore determined the limit. In Figure 2 we can see the exponential growth in execution time and in the Appendix, Figure 4 shows the exponential growth for both, the construction of the ILP prerequisites and for the execution part of the Gurobi solver. The sharp increase can further be connected to the size of the ILP in terms of dimensions of the matrix and the vectors. For a graph size of 13, the exponential growth starts and we can immediately see that also the gaps in the size of the ILP sharply increases what directly affects the running time. Visualized in the Appendix in Figure 6. Besides the limit of the ILP we measured several behaviors and outcomes of the VC creation which are presented in the Appendix. One main insight can be seen in Figure 5 (a) which shows the objective ratio when allowing only level 0 VCs. We can see that even with using unique transactions, the cost can be decreased by around 30%. In Figure 5 we can additionally see that there has been almost no further decrease in the cost when allowing VCs of level 1. This is actually reasonable due to the fact that the graph is quite small and transactions mostly have at most a path length of two and for a path of length of two, one VC is enough to span from sender to receiver making level 1 VCs irrelevant.

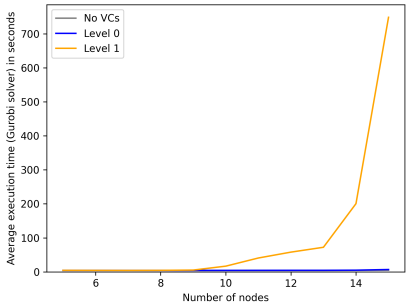


Figure 2: Nodes to Execution Time

4.3 WP3: Extensions: approximations & alternative adversaries/attacks

WP 3 has been partly integrated in WP2. Due to early observations that including the whole solution space with no restrictions is not feasible, we already made the restrictions of including only paths of maximum the diameter of the underlying graph. Nonetheless, this should not be an approximation as a path length longer than another should not be cheaper as base and relative fees may vary but not as great as justifying longer path length than from one end of the network to the other end. Furthermore, the possibility of including adversaries is implemented in the code but no further experiments have been made on potential attacks.

5 Architecture and Implementation

As described in section 4.1.3, the implementation of the main program can be split into two parts. The processing of the underlying input graph and second the creation of the input vectors and the sparse matrix. Visualized in 3, the architecture on a holistic scale consists of three different loops, automating the handling of our sample graphs as we want to experiment with different graph sizes, a number of graphs to draw an average, and different levels of recursion for each graph. Within the loops, for each graph we build the model, set the objective, compute the constraints, and finally hand it over to the Gurobi solver to calculate the optimal result. Building the prerequisites is the most work but follow all the same principle which will now be described in detail.

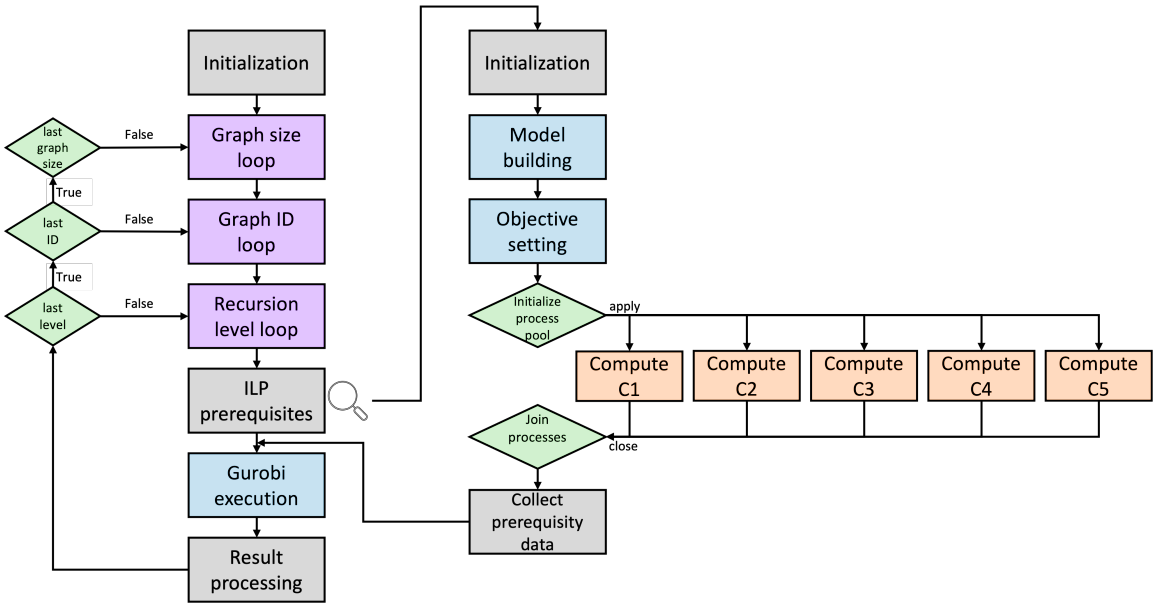


Figure 3: Program architecture

5.1 Code structure

The **Initialization** starts with the collection of the graph specific parameters, finding all VCs, and finding all paths. Whereas the paths can be found with the help of a function provided by networkx, the VC function had to be implemented. the recursive function keeps track of the different levels and for each level, the function loops over every node in the graph and finds all edges and edges connected to the edge. Basically, all paths of length two outgoing from the initial node. If the start and destination of this path does not already have a connecting edge, a new edge will be added as one and represents a VC. Although we do not want to have the same edge added multiple times, on higher levels two nodes can have different VCs as the underlying VCs can be different. This is achieved by recording all intermediaries and thereby building a unique ID for each graph.
The **model building** continues with creating the model and the x vector. For the model, we do not need any information but for building the x-vector we need to know what our objective function will look like in terms of size and types of variables. As previously described, the x-vector has the length

$|\mathbb{P}| + 2|E_{VC}|$ whereby the variables $|\mathbb{P}| + |E_{VC}|$ are of binary nature and the last $|E_{VC}|$ variables are integer.
The next stop is **setting the objective** and **computing all constraints**. The objective vector as well as all rows of the A matrix (each row representing one constraint) have to have the corresponding entries at the correct position. Referring to the x-vector, each column in the matrix and each entry in the objective vector corresponds to one path or one specific VC referring to a specific index in the x-vector. Therefore, we simply loop over the paths and VCs while continuously increasing our indices. The objective vector is filled completely but for each row in the sparse matrix, we only save the relevant values. As visualized in the graphic, the constraints are computed in parallel using multi-processing. The computations are not very complex but number of computations needed grows exponentially following the NP-hardness of the underlying problem and seen in section 4.2.
After all constraints have been computed, the resulting arrays are combined and handed over to the **Gurobi solver** which optimizes the ILP. Finally, the results are saved for later analysis and the next level or graph is initialized until the desired set of graphs is fully solved.

5.2 Environment

The implementation is fully written in Python, making use of several libraries with the most important ones being networkx [1] for most graph handling and the gurobipy [4] library for the actual model building and optimization. A full list of all external libraries including the version will be provided in the Appendix. The implementation consists of three files. `fullILP.py` is the main file executing the main function and accessing utility functions written in `fullILPUtils.py`. The third file `constants.py` contains the implementation-specific parameters which can be adjusted for the desired graphs, levels, and transactions that we want the code to run for so we do not need to make adjustments in the main code.
The execution has been done on personal hardware, to be precise a `MacBookPro12,1` with a `Dual-Core Intel Core i5`, `2,7 GHz`, `1` processor, `2` cores and `8 GB` of memory.
The program requires two input files. One `.csv` file containing the graph data in the format `cid,satoshis,nodeA,nodeB,base_fee,relative_fee`. The other input file is of type `.txt` and contains the transaction details where the first and second column defines the start and destination repectively, and the third column states the amount of the transaction.

6 Learnings and Outlook

In the following section I present some learnings that I made while acquiring the needed theoretical background as well as during the actual implementation and evaluation. Furthermore, I am going to present some potential upgrades that could be done and give an outlook on the potential.

6.1 Lessons learned

Do not underestimate exponential growth. Playing around with small graphs in the beginning showed me already that the ILP will take its time but it was still at a level of just a few seconds which were actually just based on the general overhead of spawning the different processes. The result that the ILP seems infeasible for graphs bigger than just shortly over the graph size of ten surprised in the beginning but are reasonable when looking at the underlying parameters as described in the evaluation section. So, NP-hardness is not for no reason called hard and one should take it seriously.
Many a mickle makes a muckle. or as we say in German: *Kleinvieh macht auch Mist.* Besides the

fact that we run into time constraints when executing the ILP, the ILP became very memory heavy, completely overflowing the memory, making the execution time even more slow as the computer tries to compress the data and decompress it again. But the program actually does not move big junks of data back and forth. Actually, until the Gurobi solver starts, the program only appends single entries to an array. It comes along with the exponential growth, the sheer number of those small data types gets so big that at some point they do start to matter.

6.2 Potential

Regarding the memory, the way the sparse matrix is build could completely remediate the heaviness of building the prerequisites. The code is currently structured in multi processing the different sets of constraints. This could actually be changed one step further and multi processing each single line of the sparse matrix / each single constraint instead of the set. Furthermore, each line is completely independent from the other line and although it is crucial that the columns are in the correct order the rows of the matrix only need to align with the rhs/boundary-vector which can be build with the computation of the single constraint. Therefore, it is possible to store the rows in a database and only in the end reading again from it. This could even be done by different worker nodes when thinking about distributing the workload to a cluster.
Nonetheless, although those adjustments to the code might enable the creation of bigger vectors and a larger sparse matrix, we should not forget our lessons learned and have to consider the Gurobi solver as well. While the time of the solver seems still feasible for the overall limit of the written code, we can actually see an exponential growth for the Gurobi part as well which will finally determine a limit of the ILP. Still, it would be an interesting step to also see the limit of the solver itself and although the total number of nodes and edges in the LN seem overwhelming, metrics of the LN like the diameter which is just twelve on the 28.08.2023 [2], could actually be solvable for the ILP with further smart restrictions to the solution space.
Reducing the solution space offers another big potential to the feasibility of the ILP in bigger networks. The strength in the ILP lies in the fact that it is exact and it considers every possible solutions. In comparison to an empiricial solution, the ILP has a global view. And although this global view should not be neglected, some solutions could actually be cut out. One possible way in minimizing the ILP can be to analyze the potential paths that are handed over to the ILP. We already made a restriction that we only consider paths of the length of the diameter which excludes paths that first go to the whole network before finishing at the destination. A further adjustment could be to restrict the path length to the length of the shortest possible path in the case of no VCs. When a transaction needs four hops in the network to reach a destination, paths that are of length twelve should not be cheaper. How this restriction acts with the appearance of adversary nodes and graphs where the capacity is actually limited, need to be investigated further but I see an interesting approach in the idea.
Furthermore, with bigger graphs the number of VCs started to influence the size of the ILP more and more. But similar to the paths is the global view, the ILP considers every possible VC in the network. An approach of limiting the number of possible VCs could be closely attached to the possible path. With the current implementation the VC variables are derived from the graph itself but it could actually be drawn from the paths that are computed. If a VC is not used by a single path it will never be used by definition of the VC creation. Limiting the VCs only for the part of the graph that is important for us could reduce the solution space with no drawback in the correctness of the optimization.
The strength of the ILP is definitely the global view but that does not necessarily mean the complete underlying graph. Downsizing the graph to the specific part where the transactions happen could

greatly influence the size of the ILP. The downsized graph could still be the complete graph if the senders and receivers happen to be evenly distributed at the border of the graph but also this can be considered in further implementations.

7 Conclusion

Testing the limit of a NP-hard problem is hard. But seeing the exponential growth in the different aspects of execution time and problem complexity underlines the expected NP-hardness and offers great findings not just in terms of the actual objective. Memory management plays a crucial role even when working with seemingly small data and finally heavily affects performance when reaching a limit. Nonetheless, the program shows promising results and the code still offers enhancements that could benefit the performance and possibly enabling the execution of the ILP for bigger networks. Otherwise, fundamental constraints in the final execution by the Gurobi solver should not be underestimated. The idea of restricting the solution space but still keeping the global view of the ILP is a path that can yield further improvements to the scalability of the ILP. A balance between computational feasibility and still keeping the exactness of an ILP is a tricky affair but the project offers space for further exploration helping us to investigate the behavior of PCNs and VCs.

8 Links and Resources

The initial paper that the project is based on is [3] but a follow-up paper from the same authors is currently in progress with slight adjustment that have been made in accordance to this project. Once the paper is published, the git repository will be updated with a link to the paper. The git repository including a `README.md` can be found **here**. Further resources are mentioned in the References section.

References

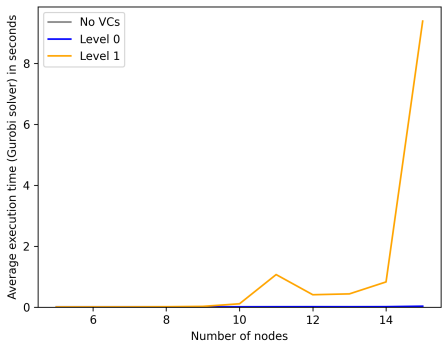
[1] Networkx documentation. `https://networkx.org/documentation/stable/`. Accessed: [May - September].

[2] 1ML. 1ml - lightning network search and analysis engine, 2023.

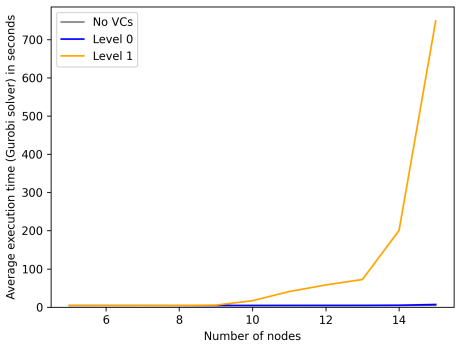
[3] Lukas Aumayr, Esra Ceylan, Matteo Maffei, Pedro Moreno-Sanchez, Iosif Salem, and Stefan Schmid. Demand-aware payment channel networks. *arXiv preprint arXiv:2011.14341*, 2020.

[4] Gurobi Optimization, LLC. Gurobi optimizer reference manual. `https://www.gurobi.com/documentation/9.5/refman/index.html`. Accessed: [May - September].

A Additional Graphs

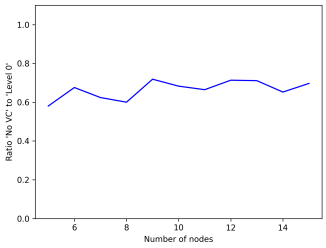


(a) Execution Time Gurobi Solver

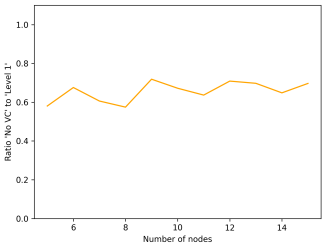


(b) Execution Time Prerequisites

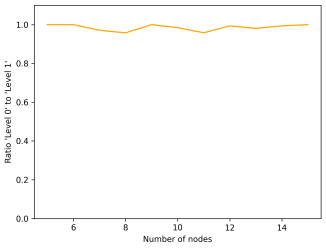
Figure 4: Average Execution Time To Number of Nodes (5 transactions)



(a) Ratio 'No VC' to 'Level 0'

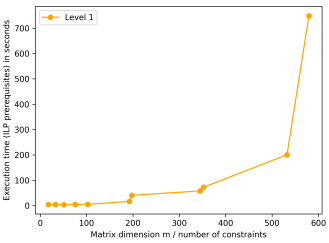


(b) Ratio 'No VC' to 'Level 1'

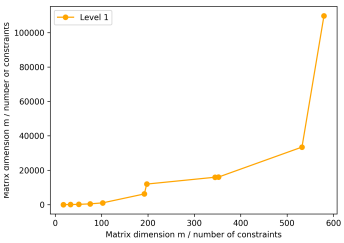


(c) Ratio 'Level 0' to 'Level 1'

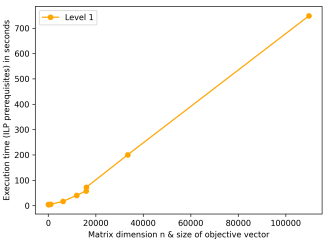
Figure 5: Average Objective Ratio (5 transactions)



(a) Execution Time ILP Prerequisites to Matrix Dimension m



(b) Matrix Dimension m to n for Different Graph Sizes



(c) Execution Time ILP Prerequisites to Matrix Dimension n

Figure 6: Matrix dimensions

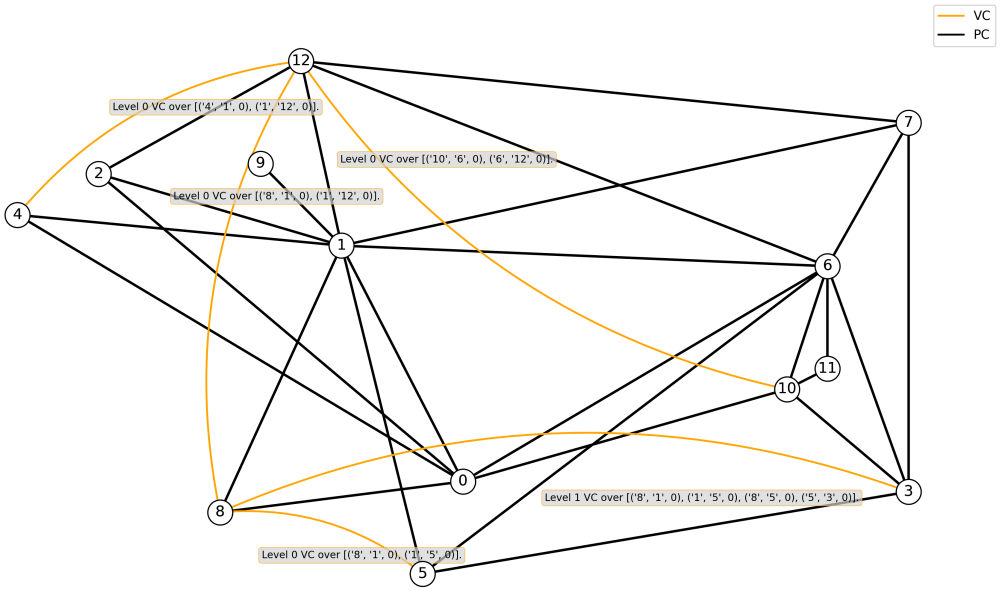


Figure 7: Network with 13 nodes and 26 PCs

5 VCs are created while allowing level 1 VCs and for 10 transactions. The level 0 VC from 8 to 5 is not used for a transaction but to create the level 1 VC from 8 to 3. The 10 transactions consist of a set of 5 transactions that are repeated once: 8 - 12, 8 - 3, 0 - 2, 4 - 12, 10 - 12.