Comparative analysis of two network anonymization settings using Integer Linear Programming

Mike J.J.S Erkemeij[1]

Supervisor: Anna L.D. Latour[1]

[1]EEMCS, Delft University of Technology, The Netherlands

June 22, 2025

Abstract

In network science, user privacy is a major concern when handling data such as social networks. These often contain sensitive data on *e.g.*, individuals, companies, and governments. Therefore, it is essential to adequately anonymize this data before sharing or publishing to protect privacy and encourage open science. In this thesis we address this problem by studying k-anonymity in social networks. Specifically, we focus on (n, m)-k-anonymity measure. Building on an existing Integer Linear Program (ILP) encoding that achieves anonymity through edge deletion. We modify this encoding to add edges as well. Additionally, we propose a heuristic to improve the running time and memory usage of the modified ILP. We compare the anonymization settings, adding and deleting edges, on synthetic and real social network datasets, evaluating their running time, memory usage, and solution quality. This analysis provides insights into the trade-off between the two settings.

1 Introduction

In today's digital world, with widespread Internet access, network anonymization has become increasingly important when preserving privacy in social networks. Network data often contains sensitive information on *e.g.*, individuals, companies, and governments, and is frequently leaked, exchanged, or even sold, raising serious concerns about user privacy. A concern already raised by Samarati and Sweeney [Samarati and Sweeney, 1998]. Anonymizing such data enables safe sharing and encourages goals such as open science, as researchers are often hesitant to share data due to privacy risks.

Simply removing unique identifiers, *i.e.*, pseudonymization, from these networks to mitigate privacy risks has been shown to be ineffective against structural identification [Backstrom *et al.*, 2007]. As a consequence, individuals can be identified. Three proposed solutions to this problem are clustering-based methods [Thompson and Yao, 2009], differential-privacy-based methods [Li *et al.*, 2014], and k-anonymity methods [de Jong *et al.*, 2024]. Clustering methods group data together based on their structural similarity, often destroying network properties. Alternatively, in differential-privacy, users can query datasets with added noised, typically the noise increases with each query. Among these three approaches, k-anonymity is the most commonly used for network anonymization [Xie, 2023]. This method perturbs the original complete network dataset and generates a modified anonymized version in return.

The k-anonymity approach partitions the network into different equivalence classes. These partitions contain nodes with equivalent structural positions based on a signature chosen beforehand. A signature represents the attacker's ability to uniquely identify nodes in a network. However, for a network to be k-anonymous, the partitions need to have a size of k or greater. Following this, a node is k-anonymous if there exists at least $k-1$ other equivalent nodes in the network [de Jong *et al.*, 2023]. Using k-anonymity, along with different signatures for node equivalence, different levels of attacker knowledge can be modeled.

Various signatures, also known as measures, correspond to different levels of attacker knowledge. A commonly used measure is based on the degree of a node [Liu and Terzi, 2008]. Two other measurements, used by Xie [Xie, 2023], are d-k-anonymity and (n, m)-k-anonymity. For d-k-anonymity, the attacker's knowledge is tuned using parameter d, representing perfect knowledge of a node up to, a given distance d [de Jong *et al.*, 2023]. In (n, m)-k-anonymity, the equivalence class of a node is labeled through its degree, n and the degree distribution in its ego-network, m.

An exact method of (n, m)-k-anonymity is approached as an Integer Linear Programming (ILP) encoding, by Latour [Latour, 2024]. ILP is a mathematical optimization technique that is used to find the optimal solutions to a linear objective function. The solution is constrained by linear equalities and inequalities with all variables restricted to integers. Latour takes advantage of this technique by modeling the (n, m)-k-anonymity problem into an ILP encoding. To achieve anonymity, the author uses edge deletion, with ILP minimizing the number of edges deleted.

While Xie, focused on edge deletion due to its computational efficiency, Casas-Roma argues that "Edge add is the best method to keep graph's properties when perturbing scale-free networks" [Casas-Roma, 2015]. Scale-free networks represent graphs with a power-law degree distributions, a distribution commonly observed in social networks. Additionally, Casas-Roma mentioned that adding 'noise' edges is regarded by some authors as a better privacy-preserving technique. This is because it preserves the original data by obscuring it rather than removing it.

In this paper, we compare two different settings, adding edges and deleting edges for anonymization in networks based on the (n, m)-k-anonymity measure. In our approach, we use an existing ILP program to delete edges and modify it to also be able to add edges. We then solve the two ILP encodings on different network topologies, comparing the running time, memory usage and the solution quality of each solution. We expand on the quality of a solution in Section 2.4

This leads us to the main research question we aim to answer:

How does an ILP implementation of the setting in which we add edges compare to the one in which we delete edges, in terms of solving time, memory consumption and quality of the solution?

The remainder of this paper is organized as follows. In Section 2, we describe the common notations and utility metrics used during this paper. Section 3, presents additional background and related research. Furthermore, we describe the different implemented ILP encodings in Section 4. In Section 5, we discuss the experimental setup and address the sub-question. Finally, in Section 6 we summarize the paper and outline directions for future work.

2 Preliminaries

In this section, we expand on the notation and definitions used during the paper. This introduces commonly used graph terminology, defines anonymity, and presents utility metrics to evaluate the quality of a solution.

2.1 Graph

We start by modeling the network as an undirected, simple, self-loop-free graph $G = (V, E)$. Here, V denotes the set of nodes and E denotes the set of edges in a network. We denote $v \in V$ as a node and $\{u, v\} \in E$ as an undirected edge that represents a connection between two nodes, $u, v \in V$. The number of connections a node has is defined as $deg(v) = |\{u \in V : \{u, v\} \in E\}|$, degree. We denote Tri(v) as the number of incident triangles on node v. And we use $\Theta := \{(u, v, w) \mid (u, v), (v, w), (u, w) \in E \; u \neq v \neq w\}$ to denote the set of triangles in the network.

Let $dist(u, v)$ be the distance between two nodes, $u, v \in V$. This distance corresponds to the shortest path between node v and w. Since graph G is undirected, the distance between two nodes is symmetric, so $dist(u, v) = dist(v, u)$. Also, $dist(v, v) = 0$ and when no path exists between two nodes $dist(u, v) = \infty$. Furthermore, the set of direct neighbors of node $v \in V$ is indicated as $N_1(v)$, where $|N_1(v)| = deg(v)$ and $dist(u, v) = 1$, also known as their ego-network [de Jong *et al.*, 2025].

2.2 Anonymity measure

When achieving k-anonymity, we aim to partition the network into equivalence classes of size k or larger. Equivalence classes are groups of nodes that share the same signature, making each node within this class equivalent. We use signature (n, m), n for the degree and m for the incident triangle's of a node.

Definition 2.1 (k-Anonimity). *A node is k-anonymous if there exist at least $k - 1$ other nodes with the same signature.*

Definition 2.2 ((n, m)-Anonimity). *An undirected, simple, self-loop-free input network $G := (V, E)$ on nodes V and edges E, is called anonymous w.r.t. (n, m) (with k a positive and n and m nonnegative integers) if the following condition holds: for each possible combination of n neighbors and m incident triangles there are either zero nodes with that (n, m) combination, or at least k [Latour, 2024].*

2.3 Toy Example

In this section, we present a formal introduction to the problem using a toy example for illustration.

Given the input graph G shown in Figure 1 on the left, we can uniquely identify nodes c and d using (n, m)-k-anonymity. Their equivalence classes differ from the other nodes, as their signatures are $(3, 1)$ and $(1, 0)$, respectively. To address this problem, we anonymize graph G by either adding or deleting edges. In the middle graph, we achieve anonymization by deleting the edge $\{b, c\}$, while in the right graph we add the edge $\{b, d\}$. Both of these approaches achieve (n, m)-2-anonymity and represent optimal solutions for their setting, as they minimize the number of edge modifications. Looking closer at our example, we observe that multiple optimal solutions may exist, as deleting edge $\{a, c\}$ also achieves (n, m)-k-anonymity.

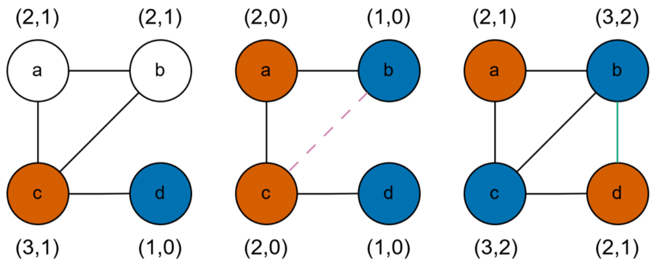


Figure 1: A toy example showing graph anonymization employing (n, m)-k-anonymity. The input graph (left) is modified by deleting an edge (middle) or adding a fake edge (right). Colors represent equivalent nodes within each graph.

2.4 Utility metrics

The quality of a solution is important for analyzing anonymization measures. Ideally, we want to preserve the network topology as much as possible, otherwise trivial solutions (*e.g.*, deleting all edges) become viable, undermining the purpose of anonymizing networks. For this, we have adopted a few utility measures used in [Zhang *et al.*, 2021] and [Casas-Roma, 2015] to evaluate the quality of the solutions.

Global transitivity (GT). The global transitivity (global cluster coefficient) is the ratio in a graph between the number of connected triplets and triangles in a graph. This represents a graph's tendency to cluster together. Here, a triplet denotes three vertices that are connected by two or three edges.

$$GT = \frac{3N_\Delta}{N_3} \quad (1)$$

Shortest path (SP). The shortest path between two nodes $u, v \in V$ is the minimum number of edges that must be traversed to travel from node u to node v. Since we assume an undirected graph, this value is equivalent to $dist(u, v)$.

Closeness centrality (CC). The closeness centrality of node, $u \in V$ is calculated by taking the reciprocal of the sum of all the shortest path lengths $dist(u, v)$, with $v \in V$ and $u \neq v$. This reveals the speed at which information travels in the network.

3 Related Work

Several methods have been developed to solve the k-anonymity problems. Most of them are either heuristic or exact approaches. Latour approaches this problem using an exact method by encoding it as an ILP. In this encoding, the author minimizes the amount of edges deleted from graph G. This is implemented in Python using Gurobi, an off-the-shelf optimization solver. However, this encoding tends to blows up as the size of the network grows.

In addition to network anonymization, researches have also explored de-anonymization techniques. For example, Zhang et al. [Zhang *et al.*, 2019] have investigated different graph anonymization measures and exploited their weaknesses. Their analysis revealed that most of these measures do not take into account the important structural characteristics of a social graph. This resulted in them successfully recovering most of the original graph.

To address these issues, we propose a heuristic in Section 4.3, which aims to improve both running time and memory usage. Additionally, the heuristic is intended to help prevent the identification of fake edges.

4 Approach

In this section, we discuss our approach for anonymizing graphs, using the (n, m)-k-anonymity measure. We encode our ILP problems in Python using Gurobi and Networkx to process the networks. We first present the existing ILP model for deleting edges. Then we introduce our modifications for adding edges and lastly we introduce a heuristic to improve the running time.

4.1 Edge deletion

In this section, we give an overview of the existing edge deletion ILP encoding. Given that the approach has not yet been disclosed, specific implementation details are omitted. However, this should give a general understanding of the approach. For the full encoding, we refer the reader to [Latour, 2024].

We start by modeling our variables. To indicate which edges are deleted, we use binary variables for E. This tells us whether or not an edge is included in the solution. We also model Θ as binary indicator variables. These help us to formulate constraints so that the solution adheres to the (n, m)-k-anonymity measure. We use L and T to represent the sets of edge variables and triangle variables, respectively.

$$L := \{\ell_{(u,v)} \mid (u, v) \in E\} \tag{2}$$

$$T := \{t_{(u,v,w)} \mid (u, v, w) \in \Theta\} \tag{3}$$

In order for the solver to compute (n, m)-k-anonymity, the variables must be constrained. Firstly, constraints are created indicating the degree and incident triangles that each node has. These are linked to each other, forming the (n, m) signature. We also define the relationship between L and T, such that these are updated accordingly. Furthermore, constraints are added to ensure that the size of an equivalence class is either 0 or at least k. Below is an example illustrating how such a constraint is formulated. These are the constraints to indicate the degree and incident triangles that each node has.

$$C_n := \left\{ n_v = \sum_{u \in N_1(v)} \ell_{(u,v)} \mid v \in V \right\} \tag{4}$$

$$C_m := \left\{ m_v = \sum_{(u,v,w) \in Tri(v)} t(u, v, w) \mid v \in V \right\} \tag{5}$$

The goal is to minimize the amount of edges deleted from graph G. This is achieved by setting the object function to maximize the number of edge considered in the solution.

$$\max \sum_{\ell \in L} \ell \tag{6}$$

4.2 Edge addition

We now present our modification on the existing ILP encoding. We start by redefining L and T, for the setting in which we add edges. In this setting the existing edges are fixed and are therefor not modeled. The only variables the model considers are the set of edges $(u, v) \notin E$, and set of triangles $(u, v, w) \notin \Theta$.

$$L := \left\{\ell_{(u,v)} \;\middle|\; \{u, v\} \notin E, u \neq v\right\} \tag{7}$$

$$T := \{t_{(u,v,w)} \mid \{u, v, w\} \notin \Theta, u \neq v \neq w\} \tag{8}$$

Since we are adding edges in this approach, we change the objective function to minimize the amount of edges used in the solution. As a result, the model adds the minimum number of edges required to anonymize the graph G.

$$\min \sum_{\ell \in L} \ell \tag{9}$$

The constraints remain unchanged, as we still employ (n, m)-k-anonymity.

4.3 2-neighborhood heuristic

We introduce the 2-neighborhood heuristic to improve the running time and memory usage. This approach is motivated by the skewed degree distribution commonly observed by social networks. These networks often follow a power law distribution, meaning most nodes have low degrees while a small number of nodes have high degrees [de Jong *et al.*, 2025].

Our modified approach considers all possible edges when adding new ones. However, this tends to grow exponentially as encoding triangles is very costly. ILP solvers like Gurobi, depend on branch-and-bound and cutting planes. When the number of variables and constrains explode, the branch-and-bound tree grows to large to search efficiently. This may even prevent the solver from reaching feasible solutions within acceptable time limits.

The heuristic reduces the search space by only considering the edges $(u, v) \notin E$, such that $dist(u, v) = 2$. Note that $dist(u, v) = 1$ implies that $(u, v) \in E$. This excludes connections between vertices that are far apart, focusing only on those at distance two. This greatly reduces the number of potential edges. As a result, the branch-and-bound tree is smaller, which improves computational efficiency.

Taking the 2-neighborhood as a heuristic serves to imitate the natural way people form connections in social networks. We assume that individuals are more likely to connect with others close to their existing connections, such as friends who introduce their friends. Following this reasoning, the heuristic may reduce the likelihood of fake-edge detection, a vulnerability highlighted in [Zhang *et al.*, 2019].

Additionally, this heuristic should help preserve the utility measures SP and CC. Since we only consider adding edges within a node's 2-neighborhood, the impact on the shortest path distances is limited. Each edge addition shortens the distance by at most one.

5 Experiments

In this section, we first present the sub-questions and our expectations. Then we describe the experiments for evaluating the methods in 4. Lastly, we show the results using the utility measures from 2.4.

5.1 Experimental Setup

Solving methods. We compare the different anonymity approaches discussed in Section 4. In this section, we refer to edge addition and edge deletion as opt_add and opt_del, respectively. These return optimal solutions. We refer to the 2-neighborhood heuristic approach as heur_add.

Software. All programming was done using Python version 3.11.13. These includes scripts for processing the results, creating graphs and the ILP implementations. The ILP encoding was modeled using Gurobi version 12.0.0 and to initialize the input networks, we used Networkx 3.5. Lastly, we processed the results using Igraph version 0.11.9 for gathering the utility metrics described in Section 2.4.

Hardware. All of the experiments are performed on the DelftBlue supercomputer clusters [Delft High Performance Computing Centre (DHPC), 2024]. Each experiment utilize an Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz. The clusters use a Red Hat Enterprise Linux 8.10 operation system. We allocated each to each experiment one CPU core and 4GB of ram. With the exception of two social networks described in Section 5.2, as they require more memory. These are given 32GB of ram.

5.2 Datasets

This section briefly explains the datasets used during the experiments. This includes synthetic networks and existing social networks. For generating graphs, we use software igraph. The real data sets consist of five undirected graphs obtained from Network Repository [Rossi and Ahmed, 2015].

Synthetic Networks. We have generated networks using the Erdös-Rényi and Barabási-Albert model. Barabási-Albert [Barabási and Albert, 1999] is a model, which simulates scale-free networks. Upon generation each added node is connected to m different nodes. We set this value to 3. Using this model, we generated 10 networks ranging from node size 5 to 15. The Erdös-Rényi model [Erdös and Rényi, 2006], is a random graph model, where each edge has an equal probability of appearing in the network. Using this model we generated graphs with 12 nodes and number of edge counts ranging from 15 to 65. This covers most of the edge density. All networks are generated using a fixed random seed of 1 to ensure reproducibility.

Social Networks. The selected networks in this study are derived from animal social networks. These typically represent interactions between animals within a certrain species. Our datasets were gathered through trophallaxis (the exchange of food or fluids), grooming (the act of cleaning and maintaining the another body), and spatial proximity (physical closeness between animals). These social networks mirror patterns of human interaction, providing insights into bigger and more complex networks.

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert\Theta\rvert$	GT
insecta-ant-trophallaxis-colony1-day1	30	37	12	0.14
insecta-ant-trophallaxis-colony1-day8	37	68	39	0.17
mammalia-baboon-grooming-group07	16	41	51	0.27
mammalia-raccoon-proximity-24	14	41	162	0.66
mammalia-raccoon-proximity-3	23	50	96	0.48

Table 1: Network topologies

5.3 Research Questions

In this section we present the sub-questions we intend to answer with the experiments. With each setting, we refer to adding edges and deleting edges.

Q1 How does opt_add and opt_del scale in solving time and memory usage as networks become larger?

Q2 How does opt_add and opt_del scale in solving time and memory usage with different network topologies

Q3 How does opt_add and opt_del perform in terms of solution quality, based on the utility measures in section 3?

Q4 How does heur_add compare to opt_add in terms of solving time, memory usage and solution quality?

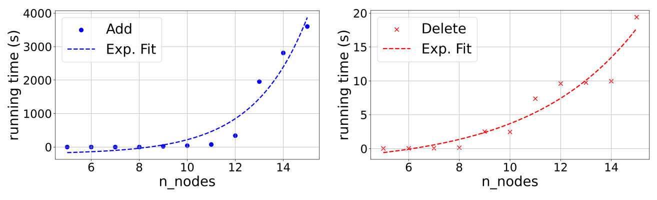
As already mentioned, we expect adding edges to be computationally more expensive than removing edges. Consequently, memory consumption is likely to be worse for adding edges. However, we anticipate adding edges to introduce less perturbations in the utility metrics.

5.4 Experimental Results

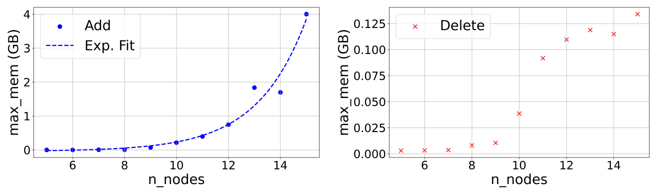
In this section, we report our experimental results and address our research questions. We use the methods explained in Section 4 and the datasets described in 5.2. The running times and memory usage are averaged over ten runs to ensure authenticity. Additionally, we use a fixed random seed of one, for reproducibility.

Q1: Network size. We compare opt_add and opt_del accros different networks of increasing size to assess how they scale in terms of running time and memory usage. These networks are generated using the Barabási-Albert model, mimicking the structure of scale-free networks. The results are shown in Figure 2.

From the results, we observe that opt_del compared to opt_add is significantly more efficient in both running time and memory usage. Moreover, Figures 2a, 2b, suggest that both approaches experience an exponential growth in running time, as network size increases. As for memory usage only opt_add follow an exponential growth, shown in Figure 2c. In Figure 2d, we can see the growth staggers after a size of 12. The overal decreaase in Computational efficiency is expected, since larger networks contain more variables and constraints, leading to a bigger and more complex ILP model to solve.



(a) opt_add running time. (b) opt_del running time.

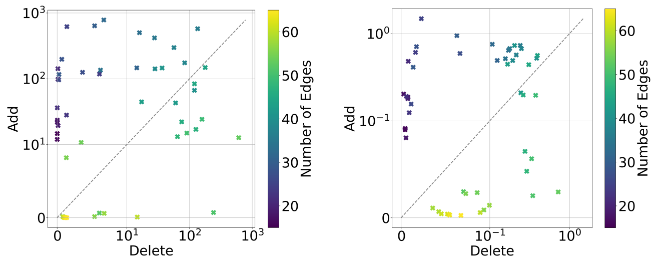


(c) Add max memory. (d) Delete max memory.

Figure 2: Running and memory usage over different network sizes.

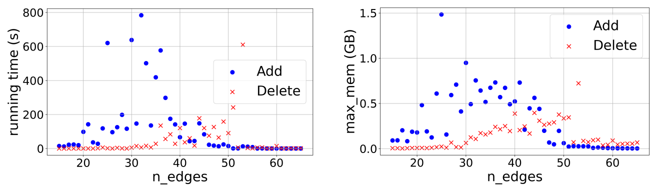
Q2: Network topology. To gain deeper insight into the impact of different network topologies we fix our network size. By keeping the size constant, we isolate the effect of the network topologies on running time and memory usage. We conduct this analysis for both opt_del and opt_add using the synthetic networks generated with the Erdös-Rényi model. Since this model is based on randomness, it helps us better understand the general behavior of the two approaches across varying topological structures.

By analyzing the results in Figure 3a and in Figure 3b, we identify a clear turning point where one approach becomes more computationally efficient than the other. Specifically, opt_del outperforms opt_add when the size of the edges are smaller. As the number of edges grows, this trend reverses, and opt_add tends to outperform opt_del. In the figures, the two methods are compared with respect to the number of edges in the network. This topology is taken as we generated the datasets on a range of edge counts. However, we observe equivalent results with other topologies, such as the number of triangles and GT. The results are provided in Appendix A.



(a) Running time (s). (b) Max memory (GB).

Figure 3: Performance comparison of edge addition and deletion operations across varying triangle densities. Color indicates the number of edges in each graph.



(a) Running time. (b) Max memory.

Figure 4: Comparison of running time and memory usage with varying edge counts.

Upon examing Figures 4a, 4b, both approaches initially decrease in performance. However beyond a certain point their running time and memory usage start to improve again. Consider opt_del for example. As the number of edges increases, the ILP model grows in size and complexity, since it must account for more possible deletions. At some point we get to the turning point and opt_del performs worse than opt_add. However, as the number of edges continues to increase, we notice a drop-off. At this point, performance starts to improve again, although it remains worse than opt_add. The reverse holds for opt_add.

Notably, neither setting is consistently dominant across the full range of edge counts. However, according to Table 2, which shows the results of the selected networks, opt_del is significantly more efficient than opt_add in both running time and memory usage. Given that social networks typically follow a power-law edge distribution, we suspect that these networks are more likeley to fall into the region where opt_del performs better.

Q3: Solution Quality. We compared the solution qualities returned by opt_add and opt_del on the five selected datasets. The quality of a solution is evaluated using the utility metrics described in Section 2.4

For SP, opt_del has the greatest impact among the approaches. Figure 5, illustrates a representative example how edge perturbations affect SP. Edge deletions, destroy paths and therefore increase the frequency of long paths and decrease the frequency of short paths. The reverse is true for adding edges. The figure also shows that opt_del causes more significant perturbations overall.

Figure 6, displays the CC per node, the results reflect the perturbation behavior of the different approaches. opt_add

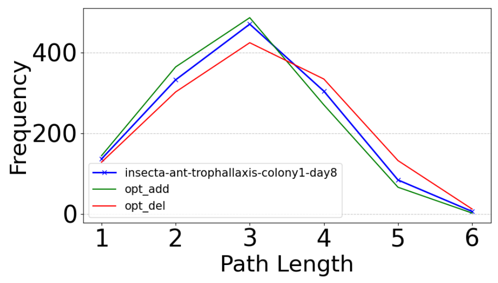


Figure 5: perturbation in SP for opt_add and opt_del

Dataset	Approach	Running time (s)	Avg Mem (GB)	Max Mem (GB)	$\|E\|$	GT
insecta-ant-trophallaxis-colony1-day1	delete	0.0970	0.00200	0.00800	27	**0.13**
	add	630	1.33	5.27	32	0.22
	heuristic	10.8	0.017	0.053	32	0.19
	original	-	-	-	30	0.14
insecta-ant-trophallaxis-colony1-day8	delete	2.96	0.00400	0.0170	33	0.13
	add	1830	5.32	20.8	41	**0.17**
	heuristic	421	0.405	0.807	41	0.21
	original	-	-	-	37	0.17
mammalia-baboon-grooming-group07	delete	1.04	0.004	0.021	13	0.22
	add	1900	1.29	1.85	20	**0.29**
	heuristic	2140	1.10	1.45	22	0.35
	original	-	-	-	16	0.27
mammalia-raccoon-proximity-24	delete	22.1	0.109	0.177	9	**0.64**
	add	14100	1.50	2.47	22	0.70
	heuristic	935	0.587	0.893	22	0.72
	original	-	-	-	14	0.66
mammalia-raccoon-proximity-3	delete	49.6	0.120	0.195	45	0.38
	add	8530	1.64	3.03	55	**0.46**
	heuristic	1530	0.635	0.919	56	0.52
	original	-	-	-	50	0.48

Table 2: Results from the selected social networks, averaged over 10 runs.

increases the CC by creating shorter paths, while opt_del destroys paths, decreasing this value. In Figure 6a, we notice that both approaches generally follow the structure of the original network, although both exhibit noticeable fluctuations in CC. For the networks with a size large than thirty nodes, opt_add performs better. opt_add, has less noticeable and smaller changes in CC, an example of this is illustrated in Figure 6b.

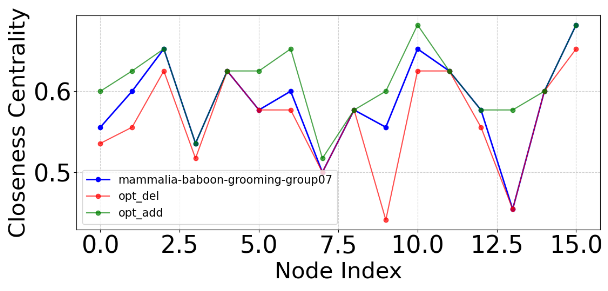
Table 2, presents the results for GT, which do not reveal a consistent trend. Consequently, no definitive conclusions can be drawn from this.

Q4: Heuristic comparison. We compare heur_add with opt_add on the selected networks. We assess them on running time, memory usage and solution quality.

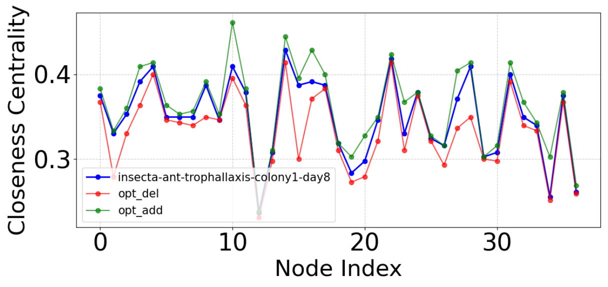
In Table 2 when looking at $|E|$, we observe that the heur_add is able to find solutions with optimal edge perturbations. However, as a heuristic it does return non-optimal solutions.

More promising results are observed in terms of running time and memory usage. According to Table 2, Memory usage is consistently lower for heur_add. This observation can be explained by the fact that heur_add considers fewer edges, resulting in fewer variables and constraints. Similar results are observed for the running time. However, there is one instance where heur_add performs worse in terms of running time. In this case, the memory usages of heur_add remains better.

When assessing the quality of the solution, heur_add tends to stick to the overal structure of the original network. This is a consistent trend for SP, due to the heuristic only adding edges within a node's 2-neighborhood. However, Figure 7 illustrates that heur_add does not always perform better than opt_add. In Figure 8 again notice heur_add to not



(a) Node size smaller than 30



(b) Node size bigger than 30

Figure 6: Perturbation in CC for opt_add and opt.

deviate too much from the original values. Regarding GT, opt_add generally introduces less perturbation compared to heur_add.

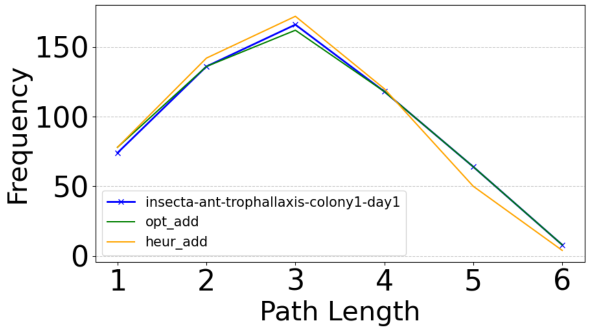


Figure 7: Perturbation in SP for heur_add and opt_add

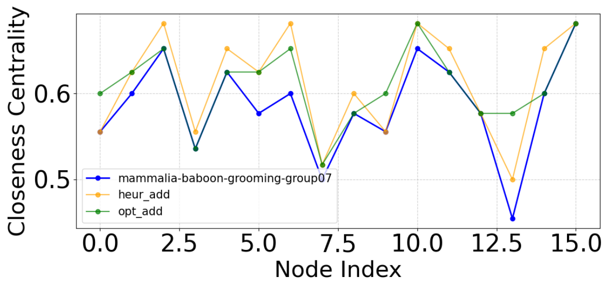


Figure 8: Perturbation in CC for heur_add and opt_add

6 Conclusion

In this paper, we focused on exact solutions for anonymizing social networks, building on an existing Integer Linear Programming (ILP) encoding based on the (n, m)-k-anonymity measure. Our primary goal was to gain insight into two different anonymization settings, edge addition and edge deletion. While prior work considers edge deletion computationally more efficient, edge addition is often regarded as the better privacy-preserving technique, as it avoids removing original data. To address the performance limitation of edge addition, we introduced a heuristic that restricts edge addition to a node's 2-neighborhood.

The experimental results on synthetic networks show that running time and memory usage are heavily influenced by network size and network topology. Even revealing a turning point where one method becomes more efficient than the other. Given that social networks follow a power-law edge distribution, we suspect that social networks fall into the region where edge deletion performs better. However, we noticed, edge deletion tends to introduce greater perturbations in SP. The utility metrics, GT and CC did not show consistent trends across methods.

Our heuristic for edge addition generally showed significant improvements in computational performance while not introducing major perturbations. However, one instance showed in increase in running time, suggesting that further investigation is needed to understand this behavior.

For future work, a natural next step would be to further investigate the heuristic, since this showed promising improvements in computational performance. Additional directions include improving efficiency by enhancing exact methods, either by refining the ILP model or by tuning GUROBI parameters. Another interesting direction is to further examine the "easy-hard-easy" pattern observed across different network topologies, as it may offer deeper insights into computational behavior.

Acknowledgments

I want to express my gratitude towards Anna Latour, for her supervision throughout the Research Project course. I also want to thank the people from Leiden, Rachel de Jong and Frank Takes, for their input and support. I want to thank Tu Delft for access to resources such as the Delftblue supercomputer and the Gurobi license.

Responsible Research

When anonymizing networks the main objective is mitigating privacy risks. If done incorrectly or not at all, users can be uniquely identified. Consequently, sensitive information about the users can be revealed. This raises serious ethical concerns. With our research we aim to reduce these risks by providing anonymization techniques and analyzing their behavior.

During this research, we used animal social networks. However, these are not fully representable of human social networks. As such, results may not translate accurately, since the selected social networks may miss key structural characteristics of human social networks.

ChatGPT was used to support during the writing of this paper. To help with the overall structure of the text. AI was not involved during the implementation of the methods and analyzing the results.

Upon reproducing this research the results should lead to similar conclusions. However, a few problems might arise. Firstly, the code is not publicly available at the time of writing. Although the experiments cannot be replicated exactly, the paper outlines the main concepts of the implementation. As such, researchers are able to create their own version with similar behavior. Another potential problem is access to TU Delft specific resources, which allows access to resources such as Delft Blue and Gurbobi licenses. Additionally, a fixed random seed was used throughout the research, further supporting reproducibility.

References

[Backstrom *et al.*, 2007] Lars Backstrom, Cynthia Dwork, and Jon Kleinberg. Wherefore art thou r3579x? anonymized social networks, hidden patterns, and structural steganography. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 181–190, New York, NY, USA, 2007. Association for Computing Machinery.

[Barabási and Albert, 1999] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[Casas-Roma, 2015] Jordi Casas-Roma. An evaluation of edge modification techniques for privacy-preserving on graphs. In *Modeling Decisions for Artificial Intelligence: 12th International Conference, MDAI 2015, Skövde, Sweden, September 21-23, 2015, Proceedings 12*, pages 180–191. Springer, 2015.

[de Jong *et al.*, 2023] Rachel G. de Jong, Mark P. J. van der Loo, and Frank W. Takes. Algorithms for efficiently computing structural anonymity in complex networks. *ACM J. Exp. Algorithmics*, 28, August 2023.

[de Jong *et al.*, 2024] Rachel G. de Jong, Mark P. J. van der Loo, and Frank W. Takes. A systematic comparison of measures for k-anonymity in networks, 2024.

[de Jong *et al.*, 2025] Rachel G. de Jong, Mark P. J. van der Loo, and Frank W. Takes. The anonymization problem in social networks, 2025.

[Delft High Performance Computing Centre (DHPC), 2024] Delft High Performance Computing Centre (DHPC). *DelftBlue Supercomputer (Phase 2)*, 2024. https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2.

[Erdös and Rényi, 2006] P. Erdös and A. Rényi. *On the evolution of random graphs*, pages 38–82. Princeton University Press, Princeton, 2006.

[Latour, 2024] Anna L. D. Latour. Research note - anonymisation - ilp encoding. personal communication (unpublished), 2024.

[Li *et al.*, 2014] Chao Li, Michael Hay, Gerome Miklau, and Yue Wang. A data- and workload-aware algorithm for range queries under differential privacy, 2014.

[Liu and Terzi, 2008] Kun Liu and Evimaria Terzi. Towards identity anonymization on graphs. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, page 93–106, New York, NY, USA, 2008. Association for Computing Machinery.

[Rossi and Ahmed, 2015] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.

[Samarati and Sweeney, 1998] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. 1998.

[Thompson and Yao, 2009] Brian Thompson and Danfeng Yao. The union-split algorithm and cluster-based anonymization of social networks. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, page 218–227, New York, NY, USA, 2009. Association for Computing Machinery.

[Xie, 2023] Xinyue Xie. Anonymization algorithms for privacy-sensitive networks. Master's thesis, LIACS, Leiden University, 2023.

[Zhang *et al.*, 2019] Yang Zhang, Mathias Humbert, Bartlomiej Surma, Praveen Manoharan, Jilles Vreeken, and Michael Backes. Towards plausible graph anonymization, 2019.

[Zhang *et al.*, 2021] Cheng Zhang, Honglu Jiang, Xiuzhen Cheng, Feng Zhao, Zhipeng Cai, and Zhi Tian. Utility analysis on privacy-preservation algorithms for online social networks: an empirical study. *Personal and Ubiquitous Computing*, 25:1063–1079, 2021.

A Extra Comparisons

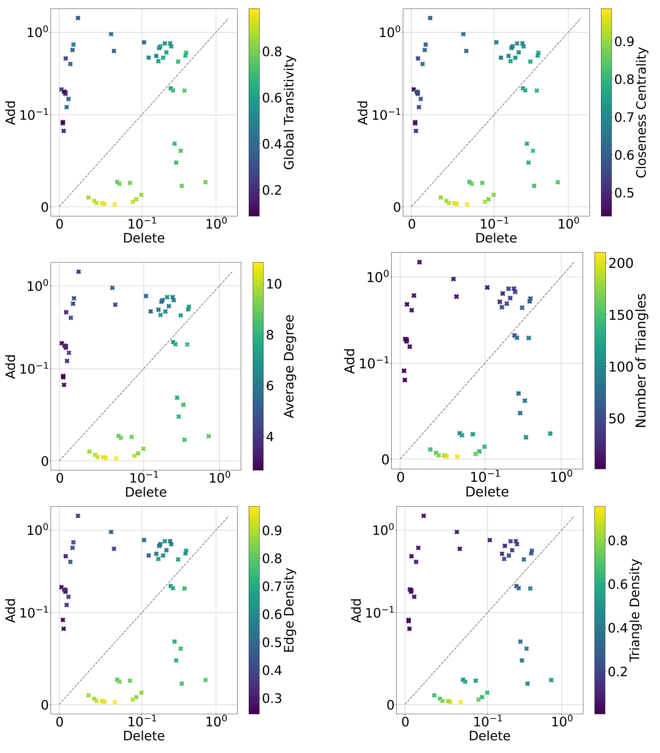


Figure 9: Max memory (GB).

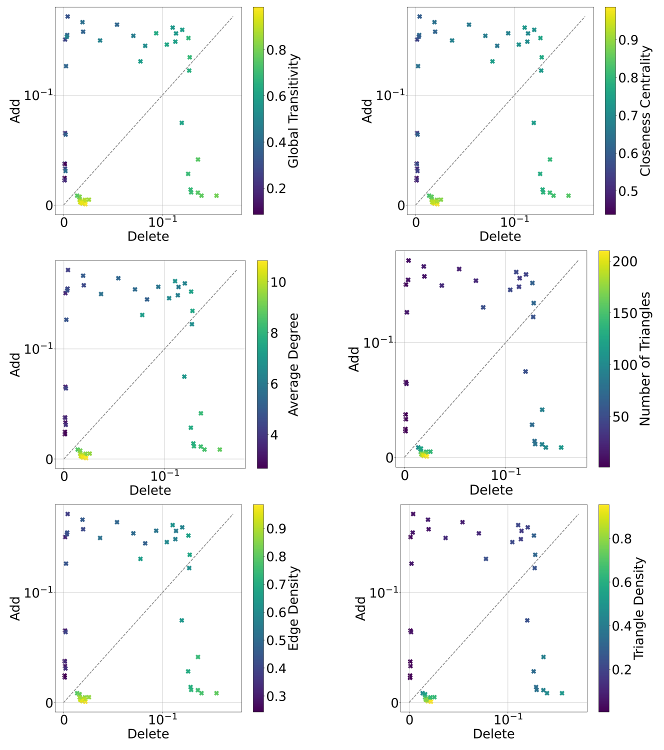


Figure 10: Average memory (GB).

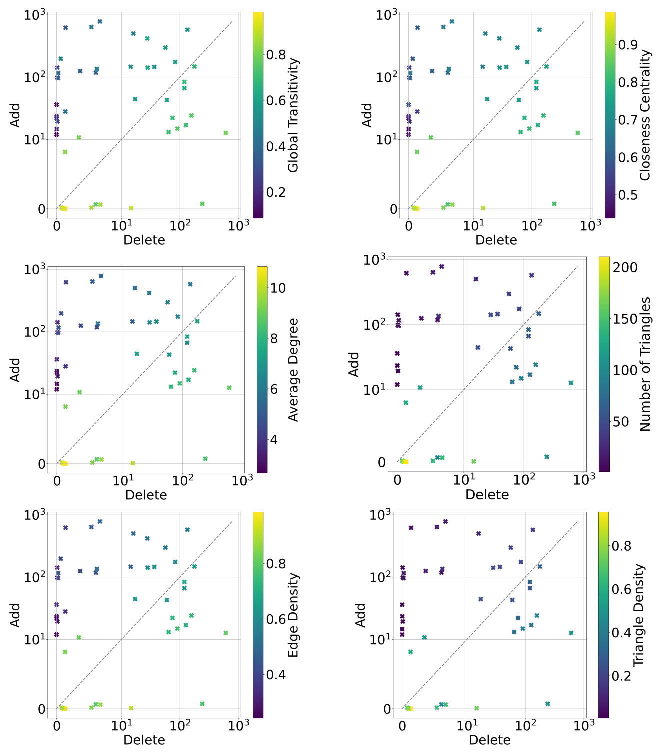


Figure 11: Runtime (S).

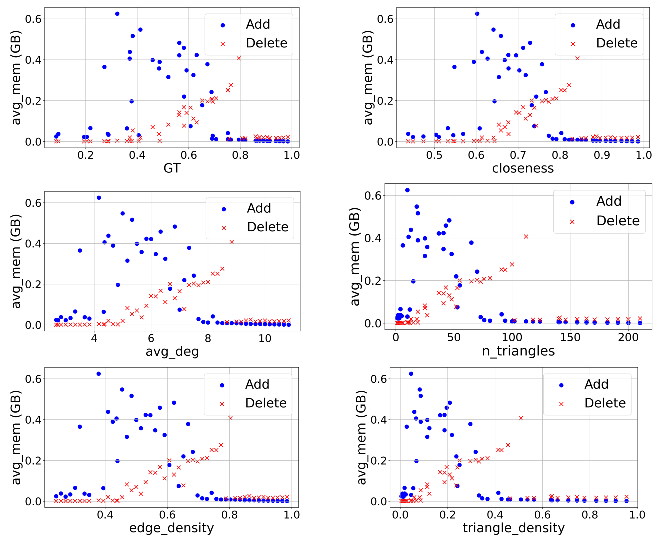


Figure 12: Average memory (GB).

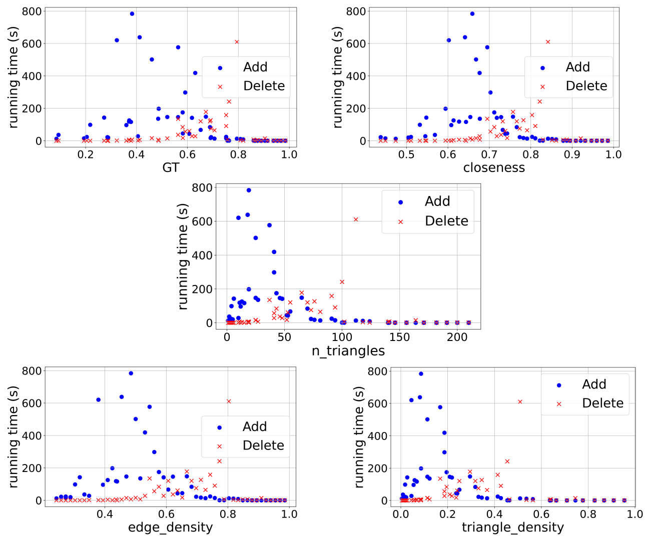


Figure 14: Running time (s).

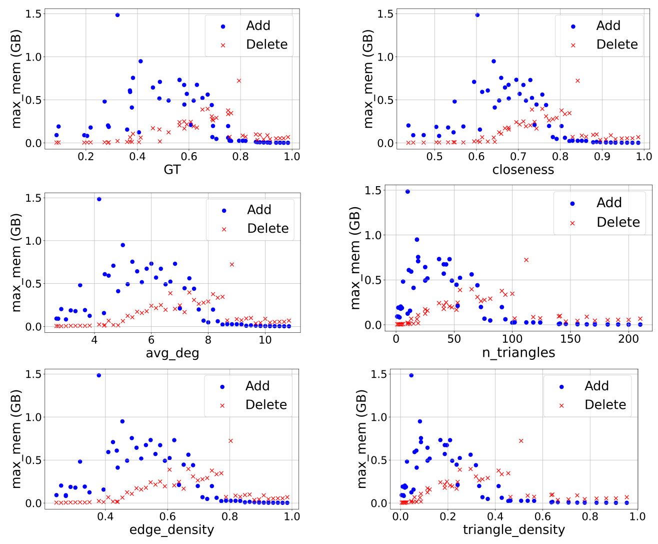


Figure 13: Max memory (GB).