A Constraint Programming Approach to Optimal Network Anonymization

Andrei Ionita[1]

Supervisor & Responsible Professor: Dr. Anna L.D. Latour[1]

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

June 22, 2025

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract

In the age of the internet, social networks are being used to study different phenomena, such as segregation, disease spread, or even peer influence. This introduces the need to protect the privacy of the individuals that are part of these networks, a problem known in the field of network science by the name of network anonymization. This involves altering the initial network using different methods such as adding, removing, or altering edges, in order to prevent attackers from identifying specific individuals. One aspect that has been studied is the data utility of the anonymized network: if we alter the initial network too much, its usability for studies is lowered. Thus, in this work, we propose a constraint programming approach that minimizes the anonymization cost, in turn maximizing the data utility of the final network. We introduce a new *isomorphic_neighborhoods* constraint and show that, for small($n < 20$) or highly dense graphs, we can guarantee the maximum data utility, by adding the fewest number of edges that guarantee anonymity.

1 Introduction

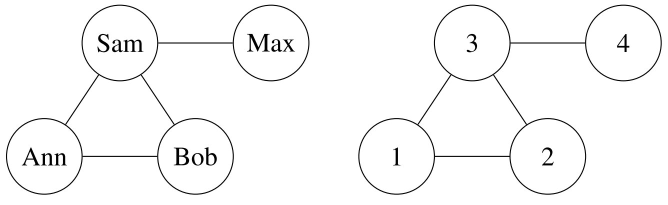
The term of open science is being promoted increasingly in modern times. One of the fields that is tasked with the transition to this concept is network science. Given the current times, social networks have become ubiquitous in people's daily life. Thus, data in the form of networks exists and continues to grow. A high variety of different fields, such as healthcare, psychology, economics, and of course, computer science, stand to benefit from the study and analysis of networks.

Thus, there is a growing need of the great amount of data to become public and open to regular use. Psychologists can study different social phenomena such as segregation or group dynamics; doctors and scientists can model disease transmission patterns; analysts can map global trade networks in order to detect dependencies and improve the economy - and the list goes on.

However, one can not directly make public all existing network data. There are major ethical concerns that center around the privacy of individuals represented in these networks. Personal and even highly confidential information exists in these networks, such as a person's salary or medical history. In general, even the identity of an individual ought to be protected from attackers that have different goals and attacking strategies.

The problem of network anonymization aims to address this challenge by ensuring that a network is first anonymized, protecting an individual's identity and information, before sharing the network publicly. The approaches that are used to solve this problem are called *anonymization methods*. There are different classes of methods, *i.e.*exact and heuristic, all concerned with solving the same problem.

An initial idea towards anonymization is to hide the names of the individuals in the network, and to assign each node a random *id*. This method is called naive user identity removal. However, the work of [Backstrom *et al.*, 2007] showed that this measure is not enough to prevent attackers from re-identifying specific individuals, depending on the knowledge that they posses.



(a) Original social graph (b) Anonymized version

Figure 1: A 4-person friendship graph. Nodes 3 and 4 have unique degrees, enabling re-identification in spite of the random IDs.

We briefly show how naive user identity removal cannot ensure anonymity in Figure 1. Assume an attacker has prior knowledge of the number of friends of each individual in the network, a number equivalent to the degree of a node in the graph. Even after removing the names of each individual in Graph 1a, node 5 is the only one that has a degree of 3, so Bob's identity is compromised to our attacker. Thus, more complex anonymization methods exist in the literature.

An important aspect of these methods is the usability of a network after it has been anonymized. If a method alters the initial network too much, no relevant statistics or results can be inferred from the resulting network. Thus, the concept of *data utility* aims to define and measure how much a network's general properties are affected by an anonymization method. In this paper, we focus on adding as few edges as possible to an initial network, in order to maximize the resulting data utility of the network.

Different attackers have access to different types of information about networks. For example, in the previous example that shows why naive user identity removal does not guarantee anonymization, we assumed that our attacker has prior knowledge of each individual's number of friends. In the literature, other assumptions regarding the attacker's knowledge include knowledge of partial or complete neighborhoods of individual nodes in a network [Zhou and Pei, 2011], or other structural attributes of the network [Narayanan and Shmatikov, 2009]. This motivates the existence of *anonymity measures*, which represent ways of quantifying the anonymity of each node in a network based on the assumed knowledge of an attacker. Some of the most commonly used include measures which look at the neighborhood of a node, or its degree[Solonets *et al.*, 2018].

This paper approaches the network anonymization problem with the objective of finding a fully anonymized network. We aim to maximize the data utility of the resulting network by adding as few edges as possible to fully anonymize the network. Thus, we present a novel approach from the field of constraint programming, and show that while slower than existing methods, the guarantee of optimal data utility makes our solution feasible on small or highly dense graphs.

In the following chapter, the exact problem that we are discussing in this paper will be defined, along with the other necessary preliminaries. The following chapter contains information about the related work, and Chapter 4 contains the main contributions of this paper. In Chapter 5, we discuss the experimental setup and the results, followed by a discussion about the ethical aspects and reproducibility of our research in Chapter 6. Lastly, the conclusion and our final remarks are presented in Chapter 7.

2 Preliminaries

In this section, we define and describe the concepts and notation used throughout the whole text, concerning general graph terminology, anonymization measures and methods, and constraint programming. We refer the reader de Jong *et al.*'s 2023 paper on the topic of efficient algorithms for computing the anonymity of nodes [de Jong *et al.*, 2023], where a more in-depth description and presentation of terms related to graphs can be found.

2.1 Graphs

Let $G = (V, E)$ denote an undirected, connected graph, where V is the set of nodes, and E is the set of edges. In the context of social networks, an edge $\{u, v\}$ represents a relation between two individuals, which are represented by the nodes u, v. In addition, as is the case in social networks, G does not contain self-loops. The distance between two nodes $u, v \in V$, denoted $dist(u, v)$, is equal to the number of edges in the shortest path from one to the other. If there is no path between two nodes, we represent this through $dist(u, v) = \infty$. As we do not allow self-loops, we also represent the distance between a node and itself as $dist(v, v) = \infty$.

The d-neighborhood of a node u is the subgraph that contains all of the nodes $v \in V$, such that $dist(u, v) \leq d$, and all of the edges between these nodes. The d-neighborhood of v is sometimes denoted as $N_d(v)$.

We say that two graphs are structurally indistinguishable if they are isomorphic.

Definition 2.1. (*Graph isomorphism*) Given two graphs $G = (V, E)$ and $G' = (V', E')$. We call the two *isomorphic*, or write $G \simeq G'$, if there exists a bijective function $\phi : V \to V'$ such that for all pairs $(u, v) \in V \times V$, $(u, v) \in E \leftrightarrow (\phi(u), \phi(v)) \in E'$.

In the literature, the d-neighborhoods of two nodes are used in order to check if the two are equivalent through an *isomorphism test* between the two subgraphs.

2.2 Anonymity measures and methods

When choosing an anonymity measure, what one actually chooses is the way the equivalence between two nodes is measured. This ties into the concept of k-anonymity, which requires all nodes to have at least $k - 1$ other nodes they are indistinguishable from. In this case, k-anonymity requires nodes to be equivalent to at least $k - 1$ other nodes.

Definition 2.2. (*k-anonymity*) Given a graph $G = (V, E)$ and a function $f : V \to P$, where f is a function describing the assignment of each node to an equivalence class p. We call G *k-anonymous* if $\forall v \in V$, it holds that there are at least $k - 1$ other nodes $\{v_1, \ldots, v_{k-1}\}$, such that $f(v) = f(v_1) = \cdots = f(v_{k-1})$.

For example, in the case of d-k-anonymity, two nodes are equivalent if their d-neighborhoods are isomorphic:

Definition 2.3. (*Node equivalence based on neighborhoods*) Given two nodes $u, v \in V$, we say that u is equivalent to v, or write $u \simeq v$, if $a) N_d(u)$ and $N_d(v)$ are isomorphic and $b)$ there is an isomorphism $\phi : N_d(u) \to N_d(v)$ such that $\phi(u) = v$.

In this paper, we are studying undirected, connected graphs, and assuming that an attacker has perfect knowledge of the complete 1-neighborhoods of every node in a graph. As we are focusing on k-anonymity methods, for each node in our graph, we are trying to find at least $k - 1$ other equivalent nodes. Based on our problem definition, two nodes are considered equivalent if their 1-neighborhoods are isomorphic. We write $N(u)$, omitting the subscript, to represent the 1-neighborhood of node u.

There are multiple ways of anonymizing a graph, with the main ones being adding, deleting or altering edges or nodes. We assume that no edges/nodes in the original network are deleted, thus only considering the case of edge additions. This choice comes with two benefits: first, no relationships/individuals in the original network are lost in the final, anonymized network. Secondly, for a given $k \leq |N|$, where $|N|$ is the number of nodes in a graph, there is also a guarantee that a k-anonymous network exists. In the worst case, this is the clique made of all of the nodes in the network.

After anonymizing a graph based on the chosen measure and method, the resulting graph is used in various data analysis processes or studies. Thus, we aim to keep the anonymized graph as close as possible to the original graph. This "closeness" is quantified by how we choose to define the concept of data utility.

In this study, we propose an approach that maximizes the data utility of the resulting graph, by minimizing the amount of added edges. Other, more complex functions of measuring the data utilities can be defined, but we choose the most straight-forward and easy to compute measure, as it is also used in [Zhou and Pei, 2011].

Definition 2.4. (*Data utility based on number of added edges*) Given an initial graph $G = (V, E)$ and the graph $G' = (V, E')$, resulted after applying our approach, we calculate the *data utility* of G' as $\frac{1}{|E'| - |E|}$.

2.3 Constraint programming

Constraint programming is a programming paradigm mainly used for solving combinatorial problems. Usually, using a declarative language, users state what consists of a feasible solution by making use of constraints. These constraints, compared to imperative programming, represent the properties of the solutions, not specific steps to be executed.

Formally, a constraint is a relation between one or multiple variables that specifies a limit of the values that the variables can take.

The user also specifies a solver which searches for the described solution. The solver usually makes use of methods such as backtracking, pruning of the search space, or propagation. Each time the solver assigns values to variables, it re-evaluates the list of constraints defined by the user. If the solution made up of the current assignments satisfies all of the constraints, it may return the solution if the user's objective is to find any solution. Otherwise, if the objective is to optimize a certain function, the solver keeps searching for solutions that satisfy all of the constraints, only returning the best solution in the end.

We propose a new *isomorphic_neighborhoods* constraint that we use to perform isomorphism tests between the 1-neighborhoods of each node in the graph.

3 Related Work

To address the problem of network anonymization, multiple approaches have been proposed. The two most widely used methods are differential privacy - a method described in [Dwork and Smith, 2010], which involves sharing partial graphs instead of the full data, and *k-anonymization*. The term k-anonymity was proposed in [Samarati and Sweeney, 1998], where the authors also propose a first method for ensuring anonymity for information in the form of tables. Our work also focuses on k-anonymity methods.

In this paper, we are studying networks where the only attribute that we aim to anonymize and protect from attackers is the identity of an individual.

In modern social networks such as *LinkedIn* or *Facebook*, when accessing a person's profile, one only has access to their immediate connections/friends. Thus, there are multiple works such as [Zhou and Pei, 2011], or [de Jong *et al.*, 2024], where the authors argue that, in order to model the problem as close to real-life scenarios as possible, the 1-neighborhood of a node is a relevant choice in terms of anonymity measures. We also focus on the scenario where an attacker only has access to the complete 1-neighborhood of any given node.

The 2011 paper by Zhou and Pei [Zhou and Pei, 2011] introduces a greedy algorithm for anonymizing a graph, assuming the same knowledge of an attacker. The authors motivate the usefulness of the algorithm by showing that the anonymized network can be used to respond accurately to aggregation queries. We introduce a complete algorithm that guarantees optimality through the use of a constraint programming approach.

This approach has been previously used in [Solonets *et al.*, 2018], where the authors introduce a new constraint, used to provide the optimal anonymized network. The difference comes from the assumption of an attacker's knowledge. The authors use a much simpler anonymity measure, namely the degree count of the nodes. The definition of optimality used by the authors is based on minimizing the anonymization cost that comes with adding dummy edges to the initial network. This cost can be represented through multiple formulas, with the most straightforward representation being simply the number of added edges.

4 Approach

This section presents a novel constraint programming approach for the previously defined problem of anonymizing a graph, assuming that the attacker has perfect knowledge of each node's complete 1-neighborhood. We introduce an *isomorphic_neighborhoods*$(E, isomorphic)$ constraint that ensures that based on the set of edges E, the assignments of isomorphic or non-isomorphic pairs of nodes are correct. We ensure that the constraint holds by performing isomorphism tests between the 1-neighborhoods of all pairs of two nodes in our graph.

Definition 4.1. (*The isomorphic_neighborhoods constraint*) Let $G = (V, E)$ be an undirected, connected graph, and $isomorphic$ a matrix of boolean values, where $isomorphic[i, j]$ states whether the nodes i and j's 1-neighborhoods are isomorphic or not, or not yet fixed by the solver. The constraint $isomorphic\_neighborhoods$ holds *iff*:

1. $\forall i, j \in V, isomorphic[i, j] = 1 \implies N(i) \simeq N(j)$
2. $\forall i, j \in V, isomorphic[i, j] = 0 \implies N(i) \not\simeq N(j)$

Informally, the constraint fails if the solver assigns two nodes as having isomorphic 1-neighborhoods while this is not actually the case, or vice-versa.

A constraint programming model that does not make use of any custom constraints has to use expensive global constraints in order to simulate isomorphism tests. For example, in order to find a bijective function from graph A to graph B, it would have to make use of the *all_different* constraint in order to ensure the injectivity of the function. Moreover, finding the actual function assignments involves letting the solver reason over at least n^3 variables: for each pair of two nodes i, j, the solver needs to find an assignment k' for every node $k \in N(i)$ such that $k' \in N(j)$.

Thus, we move the logic behind isomorphism tests away from the solver, and into the form of a constraint. Our *isomorphic_neighborhoods* constraint calls a propagator with the same name that performs isomorphism tests every time the set of edges E or the pair of assignments *isomorphic* change. Instead of having the model dynamically compute the neighborhoods of each node and simulate isomorphism tests through expensive global constraints, we perform these steps in the solver's backend by implementing our own constraint.

Every time the set of edges or the assignment of isomorphic or non-isomorphic pairs of nodes change, a propagator enforces the *isomorphic_neighborhoods* constraint. For each fixed assignment of an isomorphism between the neighborhoods of two nodes, we perform an isomorphism test and check whether the graphs are actually isomorphic. If the test fails, the propagator raises a conflict, failing the constraint in turn. The opposite happens if a pair of two nodes are marked as non-isomorphic but an isomorphism test shows that their neighborhoods are isomorphic.

In order to address the problem of symmetry, and to avoid computing an isomorphism test between both $(u, v), (v, u)$, for any fixed assignment $isomorphic[i, j]$, we also propagate its value to $isomorphic[j, i]$, if unfixed.

We describe the resulting algorithm in Figure 1.

5 Experimental Results

In this section, we aim to answer the following two research questions:

Algorithm 1 Propagation algorithm

```
Require: $E, isomorphic$ as described above
Ensure: $\forall isomorphic[i,j] = 1, N(i) \simeq N(j)$ and
        $\forall isomorphic[i,j] = 0, N(i) \not\simeq N(j)$
 1: let $n$ = maximum node index in $E$    ▷ Number of nodes
 2: for $i \leftarrow 1$ to $n-1$ do
 3:     for $j \leftarrow i+1$ to $n$ do
 4:         if $isomorphic[i,j]$ fixed then
 5:             run isomorphism test between $N(i), N(j)$
 6:             let $flag = 1$ if $N(i) \simeq N(j)$, $0$ otherwise
 7:             if $flag = 1$ and $isomorphic[i,j] = 0$ then
 8:                 raise propagator conflict
 9:             else if $flag = 0$ and $isomorphic[i,j] = 1$ then
10:                 raise propagator conflict
11:             else if $isomorphic[j,i]$ not fixed then
12:                 propagate $isomorphic[j,i] = flag$
13:             end if
14:         end if
15:     end for
16: end for
```

1. How does a model using our *isomorphic_neighborhoods* compare to a simple model in terms of running time?
2. How does the anonymity factor k affect the running time and data utility of our solutions?
3. Are constraint programming approaches feasible in terms of running time?

In order to answer each question, we tested and compared two different solutions. We call BasicModel the constraint programming model only using global constraints, and IsoModel a model that makes use of the *isomorphic_neighborhoods* constraint.

5.1 Datasets

The graph instances used come from 3 different sources: first, we selected datasets that represent real networks from the KONECT collection of networks [Kunegis, 2013], and from the Network Repository [Rossi and Ahmed, 2015]. The number of nodes and edges of each selected instance can be found in Table 1. We pre-processed the original graphs to remove duplicated edges - which do not affect the model - for consistency reasons, as well as indexed all of the nodes from 1 to the number of nodes in each graph.

We also generated undirected, connected graphs using the Erdős–Rényi model, as introduced in [Erdős and Rényi, 1960]. We adapted the model's idea of adding all possible edges between a number of nodes N, with a probability p, to generate graphs. As we will show, our constraint programming approaches are only feasible on small graphs($n < 20$) or highly dense graphs (with density ≥ 0.9).

We generated 41 undirected graph instances, to reach a total of 50 instances (together with the 9 real life networks). The graphs have a maximum of 20 nodes and have a density of over 0.8.

We mention the density and the fact that our instances are highly dense because of one reason: none of our two models perform partial assignments or propagations to stop the search early in the search tree. In order to find the optimal solution in terms of adding the least amount of edges, IsoModel needs to try all possible permutations of unassigned edges. For a graph G with n nodes and m initial edges, there are $\frac{(n-1)*n}{2} - m$ edges that can be added to G. We call these candidate edges. Thus, IsoModel will try $2^{\frac{(n-1)*n}{2} - m}$ configurations, and for each of them will perform, through our constraint and propagator, isomorphism tests between each of the graph's nodes.

Based on our experiments, with a time limit of 5 minutes, IsoModel can handle graphs that have at most 20 candidate edges, hence the high density of our instances.

On the other hand, BasicModel is limited by the need to find a bijective function for each pair of two nodes in the graph. BasicModel is tasked with adding edges to our graph, while at the same time finding isomorphisms between neighborhoods. The model performs the latter in the most expensive way possible: by randomly assigning values and then checking them through expensive constraints.

While conceptually, the idea of performing isomorphism tests is the same, we rely on an implementation of these tests from the *petgraph* crate in our *isomorphic_neighborhoods* propagator. This allows the solver to only search and assign candidate edges, while *petgraph* handles the logic related to isomorphisms.

Network	Nodes	Edges
South African companies	6	8
Southern women	5	8
Rhesus macaques	16	111
mammalia-voles-kcs-trapping-43	5	4
mammalia-baboon-association-group17	9	15
mammalia-baboon-association-group18	10	27
aves-weaver-social-14	11	38
aves-weaver-social-01	7	12
aves-barn-swallow-non-physical	17	122

Table 1: Real datasets used in experiments, and their number of nodes and edges [Rossi and Ahmed, 2015; Kunegis, 2013]

5.2 Experimental setup

All of the experiments were conducted on a laptop running Windows 11 Home with an Intel Core i5-11400H CPU at 2.7 GHz and 16 GB RAM. The models were implemented in the MiniZinc modeling language [Nethercote *et al.*, 2007]. The constraint and propagator were implemented inside of the Pumpkin solver[1]. The solver used for BasicModel is Gecode [Gecode Team, 2006], as for models that do not have custom propagators, it currently outperforms Pumpkin.

There were two main aspects that we measured throughout our experiments: first, the resulting data utility of the anonymized graph, based on Definition 2.4. We also measured the running time of our solutions in seconds. We compared the results in order to show that a constraint program-

[1]https://github.com/ConSol-Lab/pumpkin

ming approach is feasible on small or highly dense graphs, from the perspective of time, and provides an optimal result, from the perspective of data utility.

5.3 Results

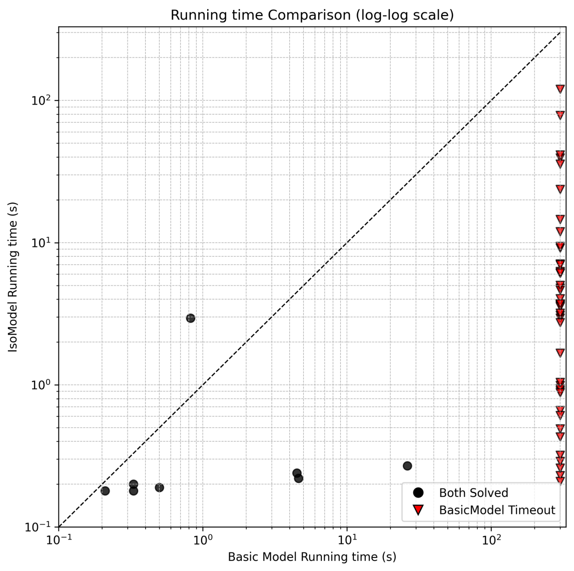


Figure 2: Comparison of running times of *IsoModel* and *BasicModel* across 50 instances. Timeout values are capped at 5 minutes and marked with red.

We start with a comparison of the two models. Figure 2 reports the running time of the two models for all 50 different instances. We initially started with an anonymization factor $k = 2$. Despite the low value of k, BasicModel only finds the optimal solution in 8 of the 50 instances, timing out for all the others.

We did not plot the amount of added edges in Figure 2, as for the instances where both models successfully ran, the anonymization cost was equal. This is to be expected due to the goal of both BasicModel and IsoModel being to find the solution that adds as few edges as possible. In the same figure, we can see that BasicModel outperforms IsoModel for 7 instances. This can be explained by the idea that on really small instances, assigning random values to variables can sometimes be faster than performing actual isomorphism tests, as is the case with our *isomorphic_neighborhoods* constraint.

Since for $k = 2$, BasicModel only runs on a fraction of the instances without timing out, we only analyze the performance of IsoModel when k increases in Figure 3. We observe that, when k increases, the anonymization cost, represented in our case by the amount of added edges, also increases, or at the very least stays the same. This is to be expected, due to higher values of k requiring additional anonymization to be reached. However, due to the nature of constraint programming, the optimal, minimum cost is still guaranteed.

Despite being able to obtain the minimum possible anonymization cost, our solution, IsoModel, has clear limitations when it comes to the size of the graph to be anonymized. Other approaches, while not able to guarantee the maximal resulting data utility, can anonymize graphs of up to tens or even hundreds of thousands of nodes and edges. While we can guarantee our anonymized graph has the least amount of added edges from the original one, we can only do so for small graphs($n < 20$), or highly dense graphs, where the amount of candidate edges is small enough. Other methods such as greedy approaches [Zhou and Pei, 2011] or Genetic Algorithms [Alavi *et al.*, 2019] are better suited for larger graphs, or graphs that have a lower density.

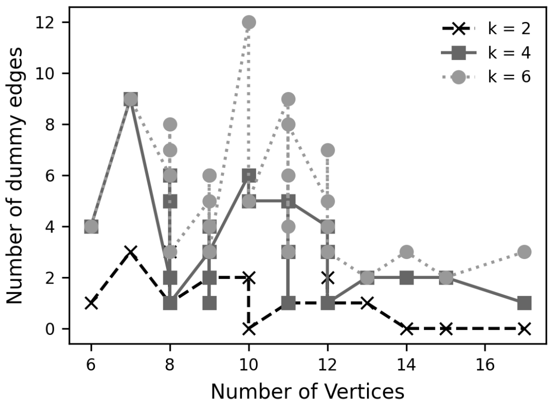


Figure 3: The influence of the anonymization factor k over the anonymization cost in terms of dummy edges added to the graph.

6 Responsible Research

The network anonymization problem stems from the need to protect the privacy of the individuals. It promotes the sharing of data, and making it public and free to use, while at the same time, trying to ensure the privacy and rights of all individuals. Ensuring that anonymized networks are free, yet safe from any de-anonymization techniques employed by various attackers, is an ongoing mission that deals with the previously mentioned ethical aspects. We, as the authors, and researchers in general, need to be careful with the data and information that they study, an idea closely related to network science.

This work also ensures that the methods and experiments are reproducible, in order to promote open and public research. We make all of the code used for our solutions available online [2], code which is written in C++ and Rust, two programming languages that are widely used and free to install on any computer.

In Section 4, we amply describe and explain the algorithms used, ensuring that even without the publicly available code, any reader experienced with a programming language can reproduce the code. The datasets used in our experiments also come from the KONECT project [Kunegis, 2013], a project specifically created in order to share public networks for free use. The solver that we used to run the model on, Pumpkin, has also recently become open source.

[2]https://github.com/AndreiLeIttu/network-anonymization-RP

Use of Generative AI. During this study, we made limited use of OpenAI's ChatGPT (GPT-4) to help with suggestions of English synonyms, and for help with installing an old C++ library. The authors are the ones who, without any external help, came up with all core algorithmic ideas, ran the experiments on the benchmark instances, and wrote this paper. All of the English suggestions from ChatGPT were taken into consideration, edited and only used when considered accurate.

By choosing a constraint programming approach, we chose a complete solution that provides a network that is guaranteed to be anonymous based on our definition of anonymity. Thus, our work promotes the protection of an individual's identity and privacy, which are the main ethical concerns related to this problem. Lastly, we adhere to the Netherlands Code of Conduct for Research Integrity [3].

7 Conclusions

In this paper, we studied the problem of network anonymization in the form of graphs, with the goal of altering a network as little as possible, in order to maximize the data utility of the anonymized network. We specifically looked at the 1-neighborhood of each node in order to decide whether it can be considered anonymous or not. We proposed a constraint programming approach that guarantees that we can find the most optimal solution based on our goal, and introduced a *isomorphic_neighborhood* constraint for the purpose of performing isomorphism tests. By implementing a new constraint, we also leave open the implementation of custom propagators enforcing this constraint in future work. Our experiments show that our approach can successfully anonymize small or highly dense graphs, while guaranteeing the optimal data utility.

Acknowledgements

I am grateful for the continuous support of my supervisor, Anna Latour. I would like to thank Rachel de Jong and Frank Takes for introducing the problem and for their support, and my four other team mates, for their great input and collaboration.

References

[Alavi *et al.*, 2019] Arash Alavi, Rajiv Gupta, and Zhiyun Qian. When the attacker knows a lot: The gaga graph anonymizer. In *Information Security: 22nd International Conference, ISC 2019*, pages 211–230, New York, NY, USA, 2019. Springer.

[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)*, pages 181–190, Banff, Canada, 2007. ACM.

[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 Journal of Experimental Algorithmics*, 28:1–22, 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. *arXiv preprint*, 2407.02290, 2024.

[Dwork and Smith, 2010] Cynthia Dwork and Adam Smith. Differential privacy for statistics: What we know and what we want to learn. *Journal of Privacy and Confidentiality*, 1(2), 2010.

[Erdős and Rényi, 1960] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17–61, 1960.

[Gecode Team, 2006] Gecode Team. Gecode: Generic Constraint Development Environment, 2006. Available at http://www.gecode.org.

[Kunegis, 2013] Jérôme Kunegis. Koblenz network collection (Konect). In *Proceedings of the 22nd International World Wide Web Conference (WWW '13)*, pages 1343–1350, Rio de Janeiro, Brazil, 2013. ACM.

[Narayanan and Shmatikov, 2009] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *2009 IEEE Symposium on Security and Privacy (SP '09)*, pages 173–187, Oakland, CA, USA, 2009. IEEE.

[Nethercote *et al.*, 2007] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, pages 529–543, Berlin, Germany, 2007. Springer.

[Rossi and Ahmed, 2015] Ryan A. Rossi and Nesreen K. Ahmed. The Network Data Repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI '15)*, pages 4292–4293, 2015.

[Samarati and Sweeney, 1998] Pierangela Samarati and Latanya Sweeney. Generalizing data to provide anonymity when disclosing information. In *Proceedings of the 24th ACM SIGMOD–SIGACT–SIGART Symposium on Principles of Database Systems (PODS '98)*, pages 188–195, Seattle, WA, USA, 1998. ACM.

[Solonets *et al.*, 2018] Sergei Solonets, Victor Drobny, Victor Rivera, and JooYoung Lee. Introducing ADegree: Anonymisation of social networks through constraint programming. In *Mobility Analytics for Spatio-Temporal and Social Data: First International Workshop (MATES '17)*, pages 73–86, Munich, Germany, 2018. Springer.

[Zhou and Pei, 2011] Bin Zhou and Jian Pei. The k-anonymity and l-diversity approaches for privacy preservation in social networks against neighborhood attacks. *Knowledge and Information Systems*, 28(1):47–77, 2011.

[3]https://www.nwo.nl/en/netherlands-code-conduct-research-integrity