Social Network Analysis and Mining (2023) 13:63
https://doi.org/10.1007/s13278-023-01064-1

ORIGINAL ARTICLE

HAkAu: hybrid algorithm for effective *k*-automorphism anonymization of social networks

Jana Medková[1] · **Josef Hynek**[1]

Received: 29 December 2022 / Revised: 3 March 2023 / Accepted: 4 March 2023

Abstract
Online social network datasets contain a large amount of various information about their users. Preserving users' privacy while publishing or sharing datasets with third parties has become a challenging problem. The *k*-automorphism is the anonymization method that protects the social network dataset against any passive structural attack. It provides a higher level of protection than other *k*-anonymity methods, including *k*-degree or *k*-neighborhood techniques. In this paper, we propose a hybrid algorithm that effectively modifies the social network to the *k*-automorphism one. The proposed algorithm is based on the structure of the previously published *k*-automorphism KM algorithm. However, it solves the NP-hard subtask of finding isomorphic graph extensions with a genetic algorithm and employs the GraMi algorithm for finding frequent subgraphs. In the design of the genetic algorithm, we introduce the novel chromosome representation in which the length of the chromosome is independent of the size of the input network, and each individual in each generation leads to the *k*-automorphism solution. Moreover, we present a heuristic method for selecting the set of vertex disjoint subgraphs. To test the algorithm, we run experiments on a set of real social networks and use the SecGraph tool to evaluate our results in terms of protection against deanonymization attacks and preserving data utility. It makes our experimental results comparable with any future research.

Keywords Privacy · Anonymization · *k*-automorphism · Genetic algorithm · Graph isomorphism · Disjoint subgraphs

1 Introduction

The growing number of users participating in online social networks (SNs) leads to supplying social network datasets with more and more information. Since each provider collects various kinds of data about their users, the amount of information stored in SNs is enormous. Social network datasets have become a precious source of information about human behavior, establishing relationships, shopping habits, and mobility patterns for academic (Macià and García 2016), medical (Kanai et al 2012; Myneni et al 2020), and marketing (Harvey et al 2020) research all over the world.

Anonymization enables providers to share or publish their datasets and preserve the individual's privacy. It aims to modify the original datasets to satisfy the required level of anonymity and keep as much data utility as possible at the same time. Modifying datasets to anonymized ones requires making semantic and structural changes, and each modification causes information loss. Hence, the goal of the anonymization method is to find the optimal processes to achieve a high level of privacy while causing minimal information loss in the dataset.

The anonymization problem can be viewed as the optimization problem of minimizing information loss where the constraint is the required level of privacy. The required level of privacy affects the complexity of the problem. The problem of finding the *k*-degree anonymous graph is proved to be NP-hard by Hartung et al (2014), the problems finding *k*-neighborhood and *k*-symmetry graphs are proved to be NP-complete by Chester et al (2013), and the problem of finding *k*-automorphism graph is proved to be NP-hard by Zou et al (2009).

The *k*-automorphism anonymization method proposed by Zou et al (2009) is the edge editing *k*-anonymity-based method protecting against any structural attack. Together with *k*-isomorphism (Cheng et al 2010) and *k*-symmetry (Wu et al 2010), it provides the highest security level among

✉ Jana Medková
jana.medkova@uhk.cz

Josef Hynek
josef.hynek@uhk.cz

[1] University of Hradec Králové, Rokitanského 62, 500 03 Hradec Králové, Czech Republic

Published online: 04 April 2023

the edge editing SN anonymization methods based on the *k*-anonymity (Zou et al 2009). However, providing a high level of privacy protection entails extensive modifications in the graph structure during the anonymization procedure. Due to the significant information loss, the *k*-automorphism methods have not been widely developed. The recent research in edge editing SN anonymization methods focused on *k*-degree and *k*-neighborhood methods. However, these methods were proven vulnerable to deanonymization attacks by Ji et al (2015).

In this paper, we demonstrate that the *k*-automorphism is a competitive SN anonymization approach providing a high-security level by proposing Hybrid Algorithm for *k*-Authomorphism anonymization (HAkAu). HAkAu is based on the structure of the KM algorithm published by Zou et al (2009). It combines the original approach with the genetic algorithm (GA).

The motivation for applying the GA arises from the fact that the *k*-automorphism anonymization problem is proved to be NP-hard, and GA is a powerful tool for solving search-based optimization NP-hard problems. Using our novel representation, finding the *k*-automorphism graph is an optimization problem of minimizing information loss. Moreover, the solution space can be modeled very well, and the particular solutions are easily comparable with each other with respect to the information loss function. Thus, GA is a good fit for finding the optimal solution to such a problem. Moreover, GA has been successfully used to improve the *k*-degree anonymization method (Rajabzadeh et al 2020), *k*-neighborhood method (Alavi et al 2019) and clustering *k*-anonymization methods (Srivatsan and Maheswari 2022; Yazdanjue et al 2020; Sihag 2012). Hence, it motivated us to use it to improve the *k*-automorphism method too. Applying GA enhances the computations and increases the quality of the final solution. The resulting hybrid algorithm is more efficient and preserves the data utility better than the KM algorithm.

Except for the employment of GA in the *k*-automorphism method, we highlight the following three novel aspects in the proposed solution: chromosome representation in GA, the algorithm for effective selection of disjoint subgraphs, and the procedure of copying crossing edges.

We propose a new chromosome representation in which the chromosome length is independent of the size of the input network. Each chromosome describes not the whole graph but the set of isomorphic subgraphs. The set of isomorphic subgraphs can be represented with a single adjacency matrix and a list of vertices. The chromosome consists of specific elements of the adjacency matrix and the list of vertices. Moreover, the length of the chromosome can be regulated by setting the maximal size of the subgraphs. Thus, the length of the chromosome and the size of the solution space of GA are independent of the size of the input network. It increases the applicability of HAkAu on larger networks and the effectivity of the computation. It is a novelty in the usage of GA on the *k*-anonymization problems. In the previously published solutions (Sihag 2012; Yazdanjue et al 2020; Rajabzadeh et al 2020; Alavi et al 2019), the chromosome always represents the whole graph. Therefore, the length of the chromosome and the solution space always grew with the size of the input network.

Furthermore, the proposed chromosome representation keeps the isomorphism between the subgraphs "by design." Each chromosome in each generation leads to the *k*-automorphism anonymized output graph. Thus, it is not necessary to check whether the solution leads to *k*-automorphism output graph by the fitness or selection function. Usually, the requirement to check *k*-anonymity and minimize the anonymization cost by fitness or selection function makes the *k*-anonymization optimization problem two-dimensional and more challenging to be solved by GA. Our representation enables the fitness function to aim only at the problem of minimizing the anonymization cost.

In the preprocessing stage of HAkAu, it is required to find *k* vertex disjoint isomorphic subgraphs in the input graph. At first, the problem of finding the frequent subgraph with minimal support higher than *k* is addressed with the GraMi algorithm proposed by Elseidy et al (2014). The output of GraMi is the list of matches of the frequent subgraph in the input graph, which are not vertex disjoint. Since GA in HAkAu requires the list of *k* vertex disjoint subgraphs, we have to address the problem of selecting *k* vertex disjoint subgraphs from the set of subgraphs. We prove that the problem is NP-hard and introduce the "divide and conquer" algorithm with effective heuristics for minimizing the computational time in average cases.

We improve the procedure of copying crossing edges. While the KM algorithm adds *k* copies for each crossing edge, the HAkAu only copies the subsection of selected crossing edges. Hence, the effect of anonymization on the degree distribution is significantly smaller.

Furthermore, we show the usability of the HAkAu by running experiments on real-world SN datasets. We perform three kinds of experiments. At first, we run HAkAu on the Prefuse network to compare the results of HAkAu with the results of KM published by Zou et al (2009). Then, we run HAkAu on three SN datasets differing in size to provide the data utility analysis. Finally, we tested the resistance of the data anonymized by HAkAu against the deanonymization attacks. The SecGraph evaluation tool is used to compute utility and network metrics and perform the deanonymization attacks.

In summary, the innovation of this paper is in the following aspects. The employment of GA in the *k*-automorphism method is innovative since it enables the reduction in two NP-hard problems into a single one. In the original solution, the isomorphic graphs were found, then they were extended in such a way that the isomorphism was lost, and then, it was necessary to make them isomorphic again. We make the process more efficient by extending the isomorphic subgraphs "isomorphically" with GA. Hence, the algorithm solves only one NP-hard problem instead of two.

The original KM algorithm was based on the application of edge addition operation. Furthermore, its copying edges procedure created a large number of dummy edges. Employing the edge deletion together with a more efficient copying crossing edges procedure in HAkAu causes the degree distribution of the anonymized dataset to be much closer to the input one. Hence, the final solution keeps more data utility.

The proposed chromosome representation is inventive since it increases the applicability of the anonymization method based on GA. As far as we know, this is the first anonymization method based on GA where chromosomes do not represent the whole graph.

The minor innovations in our solutions are the proposal of the algorithm for selecting the subset of vertex disjoint graphs from the set of edge disjoint graphs and the emphasis on the comparability of the produced results. Nearly all published studies have their own methodology for data utility measurement. Hence, it is difficult to compare the particular results with each other. We show that applying the proposed algorithm on the whole available dataset and measuring the data utility with the external evaluation tool enable generating results that can be easily compared with the results of any future research. Hence, the main contributions of our paper are as follows

1. the proposal of the novel hybrid *k*-automorphism anonymization algorithm HAkAu which improves the previous *k*-automorphism method by employing GA
2. introducing the novel chromosome representation for *k*-automorphism problem. The chromosome representation preserves the *k*-anonymity property of the chromosomes "by design"; hence, it is not necessary to test the *k*-anonymity property with fitness or selection function.
3. the proposal of the novel "divide and conquer" algorithm for effective selection of vertex disjoint subgraphs
4. the application of the GraMi algorithm in the anonymization method. As far as we know, this is the first paper where the GraMi algorithm is used in the anonymization procedure.
5. running experiments on the real-world SN datasets and evaluating the experimental results with SecGraph evaluation tool

2 Related work

This section summarizes the studies introducing *k*-anonymity methods that address the identity disclosure problem in undirected SN datasets. The comprehensive survey of all privacy-preserving solutions developed for SN is given by Majeed et al (2022).

The well-known *k*-anonymity method was introduced by Samarati and Sweeney (1998). The process anonymizes relational datasets such that at least k individuals have the same value of quasi-identifiers, attributes that can re-identify the individual in the records. The *k*-anonymity ensures that any individual is indistinguishable from the $k-1$ other individuals in the anonymized dataset.

Since background knowledge about the network structure can lead to revealing the identity of the individual, social ties in SNs are also considered to be quasi-identifiers. Modifying the edge set of the original graph such that there exist at least k nodes with the same degree for each degree value in the graph is the ground of SN anonymization methods based on *k*-anonymity: *k*-degree anonymity (Liu and Terzi 2008; Casas-Roma et al 2017), *k*-neighborhood anonymity (Zhou and Pei 2008), $k(d)$-neighborhood (Alavi et al 2019), *k*-symmetry (Wu et al 2010), *k*-isomorphism (Cheng et al 2010) and *k*-automorphism (Zou et al 2009).

The *k*-anonymity was extended to the *k*-degree anonymity by Liu and Terzi (2008). In the *k*-degree anonymous graph, there are at least k nodes with the same degree for every degree value in the graph. Hence, the probability that the attacker distinguishes their target node in the anonymized graph is $\frac{1}{k}$ if the attacker knows only the degree of their target node. The proposed algorithm was improved in later published studies in terms of speed and preserving data utility (Hartung et al 2014; Lu et al 2012; Medková 2020) or its usability in larger networks (Casas-Roma et al 2017). The *k*-degree method was recently extended with the node-based differential privacy approach to provide the protection against mutual friend attack by Shakeel et al (2021). However, the model of the attacker in the *k*-degree anonymization method is limited.

The *k*-neighborhood anonymity was introduced by Zhou and Pei (2008, 2011). The proposed *k*-neighborhood anonymity model protects against the attacker who knows the target's neighbors and their connections. The model was extended to $k(d)$-neighborhood anonymity by Alavi et al (2019). The $k(d)$-neighborhood anonymity protects against the attacker who has the background structural knowledge about nodes in the path distances up to d from the target node.

There were published three anonymization methods that protected against any structural attack: *k*-isomorphism

(Cheng et al 2010), *k*-symmetry (Wu et al 2010) and *k*-automorphism (Zou et al 2009). The definition of the *k*-isomorphic anonymized network in Cheng et al (2010), as well as the definition of the *k*-symmetry anonymized network in Wu et al (2010), is equivalent to the definition of the *k*-automorphic anonymized network in Zou et al (2009). The *k*-isomorphism anonymization algorithm, proposed by Cheng et al (2010), is very similar to the KM algorithm introduced by Zou et al (2009). Both algorithms aim to partition the input graph into *k* subgraphs and make them isomorphic to each other with edge additions. Neither Cheng et al. nor Wu et al. did not compare their experimental results with the results of other known *k*-anonymity algorithms as Zou et al. did.

Zou et al. presented the KM algorithm that ensured *k*-automorphism anonymization. The KM algorithm transforms the input graph G to the *k*-automorphism graph G^*. The algorithm starts with finding the frequent subgraph $g_s(k)$ with the given minimal support k in G by running the grow-and-store *SiGraM* algorithm (Kuramochi and Karypis 2005). Then, the KM algorithm finds k matches of $g_s(k)$ in G. The found isomorphic subgraphs are expanded, meaning some neighborhood vertices and edges are added to each of them. The expansion is necessary for decreasing the total anonymization cost. Thus, the subgraphs are larger, but they are not isomorphic to each other anymore. However, since the aim of the algorithm is to obtain *k*-automorphic graph G^*, its subgraphs have to be isomorphic. Thus, dummy edges are added to make the expanded subgraphs isomorphic to each other again.

Contrarily, HAkAu expands the subgraphs "isomorphically." It means that the expanded graphs are isomorphic after the expansion and do not require further modification. Moreover, HAkAu employs a more effective algorithm for finding frequent subgraphs and improves the crossing edges procedure where fewer edges are added to the final solution. The KM algorithm solved the task as two separate NP-hard problems: finding the optimal graph partitioning and finding the optimal graph alignment. On the other hand, HAkAu merges these problems and solves them together with a single GA.

GAs have been recently exploited in the SN analysis for community detection in large networks (Azaouzi et al 2019), graph clustering (Bello-Orgaz et al 2012; Cai et al 2015), and predicting dynamics of SN (Caschera et al 2019). Several SN anonymization methods based on GA have been recently presented as well.

Sihag (2012) proposed GA that anonymized SN by clustering nodes into supernodes. However, the algorithm was tested only on small SNs (up to 67 nodes) and cost more significant information loss than the previously published deterministic clustering algorithm Sangreea (Campan and Truta 2008).

Another genetic clustering algorithm was proposed by Yazdanjue et al (2020). They optimized the clustering procedure in the *k*-anonymity method employing particle swarm optimization. They presented a hybrid solution that combined particle swarm optimization with GAs. Their solutions were represented with binary matrices describing which node belonged to which supernode. Each chromosome contained all nodes and all supernodes of the whole graph. Hence, while anonymizing large networks, it can be used only large values of the anonymizing parameter k since larger k meant fewer supernodes and smaller chromosomes.

The genetic *k*-degree edge modification was introduced by Rajabzadeh et al (2020). At first, the algorithm detected communities in the SN graph. Then, it modified the edge set in each community with GA. Hence, the SN graph was anonymized by adding edges between vertices inside detected communities.

Alavi et al (2019) introduced the $k(d)$-neighborhood anonymity approach and presented GA for graph anonymization called GAGA. The GAGA was the edge editing algorithm that prioritized edge switching over edge adding or removing. They showed that GA was an efficient tool for anonymizing large SNs. Using the SecGraph framework (Ji et al 2015), GAGA algorithm was proved to be resistant against five deanonymization attacks. Moreover, the SecGraph was used to measure information loss after anonymization and compare GAGA with existing approaches. We agree that evaluating anonymization algorithms with an independent tool like SecGraph can lead to a better comparison of proposed algorithms. However, GAGA is tested only on the subgraph of the DBLP co-authorship network (Yang and Leskovec 2015), not on the whole DBLP dataset. Since we can not find which subgraph they used, we can not compare our results with those published by Alavi et al (2019).

Other evolutionary methods have rarely been employed in social network anonymization. As mentioned above, the particle swarm optimization was employed in Yazdanjue et al (2020); however, the final solution combined it with the genetic algorithm. The ant colony optimization was employed in Bhattacharya and Roy (2015), where the method for preventing the walked-based attack was presented. The method focused only on preventing one kind of attack, while we present a more universal approach protecting against any structural attack. Kiabod et al (2021) aimed to increase the anonymization speed and enhance the usability of anonymization in big data SN datasets. They combined the univariate micro-aggregation anonymization with the firefly algorithm with neighborhood attraction. All

Table 1 Summary of the notation

Notation	Definition
G	A social network graph
G^*	An anonymized social network graph
$\tilde{G}$	A released social network
$E(G)$	The edge set of G
$V(G)$	The vertex set of G
(v_i, v_j)	The edge between nodes v_i and v_j
k	An anonymization parameter
F_j	An automorphism on G^*
$\mathcal{Q}$	A structural query
$g_f(s)$	A frequent subgraph with the minimal support s
H, P_{ij}	Subgraphs of G
P'_{ij}	Supergraphs of P_{ij}
Q_{ij}	Subgraphs of P'_{ij}: $E(Q_{ij}) = E(P'_{ij}) \setminus E(P_{ij})$
M	A matrix
r_i(M)	The ith row of M
rc(**M**)	The number of rows of M
$\mathbf{Adj}_i$	The adjacency matrix of $P'_{i1}, \ldots, P'_{ik}$
CH	The bit part of the chromosome
$varCH$	The part of the chromosome representing the list of vertices
$Cost(G, G^*)$	The total anonymization cost
$VCost(G, G^*)$	The anonymization cost caused by vertex edits
$ExCost(G, G^*)$	The extension cost
$ExCost_i(H)$	The extension cost caused in the i-th round
$CECost(G, G^*)$	The crossing edges cost
$FF(I)$	The fitness function on the individual I
GA	Genetic algorithm
SN	Social network

mentioned and recently published studies focused mainly on data utility and omitted the level of privacy protection. The datasets anonymized by them satisfy only the k-degree anonymity, which is not a sufficient level of privacy.

Finding subgraphs with the given minimal support in the single graph is one of the subtasks that have to be solved by the k-automorphism anonymization algorithms. It is an NP-hard problem that has already been studied (Elseidy et al 2014; Kuramochi and Karypis 2005). It is possible to exploit some of the previously presented algorithms to solve the subtask. Zou at el. used the SiGraM algorithm proposed by Kuramochi and Karypis (2005). The SiGraM algorithm is based on the grow-and-store method. The SiGraM stored all appearances of each examined subgraph which requires much space and computational time.

Recently, some more effective algorithms have been proposed. The HAkAu uses the GraMi algorithm proposed by Elseidy et al (2014). GraMi stores only the templates of the frequent subgraphs, not the whole subgraphs, and models the frequency evaluation as the constraint satisfaction problem. The GraMi algorithm is proved to be faster than the SiGraM algorithm (Elseidy et al 2014).

3 Preliminaries

In this section, we formalize the system and define the basic terms from the graph theory and genetic algorithms. For reference, the summary of notation used throughout this paper is presented in Table 1.

3.1 Graph theory

Definition 1 (*Isomorphic graphs*, Zou et al 2009) Given two graphs P_1 and P_2, P_1 is *isomorphic to* P_2 (denoted by $P_1 \backsimeq P_2$), if and only if there exists at least one bijective function $F : V(P_1) \longrightarrow V(P_2)$ such that for any edge

$(u, v) \in E(P_1)$, there is an edge $(F(u), F(v)) \in E(P_2)$. The function F is called the *isomorphism* of P_1 and P_2.

Definition 2 (*Graph automorphism*) Let G be a graph, and the bijective function $F : V(G) \longrightarrow V(G)$ be the isomorphism. Then F is the *automorphism* on G.

The terms isomorphism and automorphism are frequently used in Sect. 6. The same function F is marked as isomorphism in one paragraph and automorphism in another. To explain the double marking, Lemma 1 is formulated below.

Lemma 1 *Let $P_1, P_2 \subset G$ be vertex disjoint subgraphs of G, $V(P_1) \cap V(P_2) = \emptyset$. Let $F : G \longrightarrow G$ be an automorphism on G such that $F(P_1) = P_2$. Then, the restriction $F|_{P_1}$ of F on P_1 $F|_{P_1} : P_1 \longrightarrow P_2$ is the isomorphism from P_1 to P_2.*

Proof If F is the automorphism of G and P_1 is the subgraph of G, then for all edges $(u, v) \in E(P_1)$ holds that $(F(u), F(v)) \in E(P_2)$. The same condition holds under the restriction of F: for all edges $(u, v) \in E(P_1)$ holds that $(F|_{P_1}(u), F|_{P_1}(v)) \in E(P_2)$. Moreover, since $F(P_1) = P_2$, then for all $u \in V(P_1)$ its image $F(u) \in V(P_2)$ and for all $v \in V(P_2)$ its preimage $F^{-1}(v) \in V(P_1)$. Hence, the restriction $F|_{P_1}$ maps P_1 to P_2, and it is a bijection since F is a bijection. Thus, $F|_{P_1}$ is the isomorphism from P_1 to P_2. □

Thus, if F is the automorphism of G, then its restriction of a particular subgraph P_1 is the isomorphism of P_1 and P_2. To simplify the notation in Sect. 6, the notation for the restriction is omitted, and the same functions $F_{i,j}$ are characterized both as automorphisms and isomorphisms. Note that $F_{i,j}$ are automorphisms on G, but when we focus on subgraphs that are mapped with $F_{i,j}$, then $F_{i,j}$ are isomorphisms of these subgraphs.

Definition 3 (*Isomorphism extension*) Let P_1 and P_2 be two vertex disjoint graphs. Let there be two graph isomorphisms $F_1 : V(P_1) \longrightarrow V(H_1)$ and $F_2 : V(P_2) \longrightarrow V(H_2)$, where $V(H_1) \cap V(H_2) = \emptyset$. Then, $F : V(P_1) \cup V(P_2) \longrightarrow V(H_1) \cup V(H_2)$, where $F|_{V(P_1)} = F_1$ and $F|_{V(P_2)} = F_2$, is called the *extension of* F_1 and F_2. The operation of extension is denoted by $F = F_1 \oplus F_2$. Clearly, F is also the graph isomorphism.

Definition 4 (*Frequent subgraph with minimal support*, Kuramochi and Karypis 2005) Given a graph G and the minimum support s, a graph $g_f(s)$ is called a *frequent subgraph* of G if and only if there exist s subgraphs of G, $P_1, \ldots, P_s$, that are isomorphic to $g_f(s)$ and

$$E(P_i) \cap E(P_j) = \emptyset \qquad i \neq j \qquad \forall i, j \in \mathcal{N} : 1 \leq i < j \leq s.$$

The graphs $P_1, \ldots, P_s$ are called the *matches* of $g_f(s)$ in G.

Note that the above definition assures that the matches are edge disjoint but not vertex disjoint. In large graphs, there can be several frequent subgraphs with given support s. In that case, the k-automorphism algorithms take the frequent subgraph with the largest number of edges.

Definition 5 (*Social network*) A social network is represented by the graph $G = (V(G), E(G))$, where $V(G) = \{v_1, \ldots, v_n\}$ is the set of nodes representing the participating users and $E(G) = \{e_1, \ldots, e_m\}$ is the set of edges representing the social relationships between users. The edge between nodes v_i and v_j is denoted by (v_i, v_j). The network is considered to be unlabeled.

To guarantee privacy against any structural attack, the given SN graph G is anonymized to become k-automorphic graph G^*, where k is the anonymization parameter.

Definition 6 (k-automorphic graph (Zou et al 2009)) Let G^* be a graph. If there exist at least $k-1$ automorphisms F_j in G^*, $j = 1, \ldots, k-1$, and

$$\forall v \in V(G^*) : F_{j_1}(v) \neq F_{j_2}(v) \qquad \forall j_1, j_2 \in \mathcal{N} : 1 \leq j_1 < j_2 \leq k-1,$$

then G^* is called a k-automorphic graph.

Definition 7 (Crossing edge) Let G be a graph and P the subgraph of G. Each edge $(u, v) \in E(G)$ such that $u \in V(P)$ and $v \in V(G) \backslash V(P)$ is called a *crossing edge* between P and G.

The crossing edges are essential when subgraph P is separated from G. In case two graphs P_1, P_2 are separated from G, then the set of crossing edges can be defined as $\{(u, v) \in E(G); \exists j \in \{1, 2\} : u \in V(P_j) \wedge v \notin V(P_j)\}$. A similar situation happens when more than two graphs are separated from G.

Since the description of the proposed algorithms uses matrices and manipulations with their rows, the following definition introduces the necessary notation.

Definition 8 (Matrix notation) Let $\mathbf{M}(m, n)$ be a matrix with m rows and n columns. Then $rc(\mathbf{M})$ denotes the number of rows of $\mathbf{M}$ and $r_i(\mathbf{M})$ denotes the i-th row of $\mathbf{M}$. The fact that the element e is contained in the i-th row of $\mathbf{M}$ is denoted by $e \in r_i(\mathbf{M})$. Adding the vector r as the last row if $\mathbf{M}$ is denoted by $\mathbf{M} \cup r$. Removing the row r from $\mathbf{M}$ is denoted by $\mathbf{M} \setminus r$.

3.2 Genetic algorithm

GAs originally developed by Holland (1973) are based on mimicking the processes of natural evolution: the Mendelian

principles of inheritance, the Darwinian theory of the survival of the fittest, and the genetic recombination operators allowing the inheritance of specific features and traits as well as the introduction of small but necessary changes. GAs are very variable; here, we outline the pattern of the GA used in HAkAu. The specifications of the proposed functions, operators, and chromosome representation are given in Sect. 7.

The search for a suitable solution to a given problem is viewed as the competition amongst the whole population of *individuals* representing potential solutions to the problem. These individuals are encoded as *chromosomes* of fixed length. At the beginning of the process, the *initial population* of individuals is randomly generated. All individuals are evaluated with the *fitness function*. The evaluation of individuals forms the basis of their chance to be selected for survival and reproduction. The *selection function* is employed to emulate the processes of natural selection where the fitter individuals are given a higher chance to be selected. Consequently, the selected individuals go through the process of reproduction. The reproduction operators used in this paper are crossover and mutation. *Crossover* is two-parent operation. It combines two parent individuals into a single child individual. *Mutation* is an asexual recombination operator maintaining genetic diversity between generations. After producing offspring, a new population of individuals is created. The repetition of this process, the selection of fitter individuals, the inheritance of their potentially favourable features, and newly introduced random changes are the main powers enabling the subsequent generations of potential solutions might include one or even several individuals presenting a suitable or even optimal solution to the given problem (Hynek 2002).

4 Problem definition

In this paper, we focus on the problem of privacy-preserving protection against structural attacks. Additionally, the attacker could have other non-structural information like vertex labels or edge labels. However, this paper does not address the problem of preserving protection against non-structural attacks.

More precisely, the privacy-preserving issue addressed in this paper is the *identity disclosure problem* in SN datasets. Assume the provider possesses the graph G representing the SN network. The provider desires to share or publish the data. The version of G that is shared or published is called the *released graph* $\tilde{G}$. The released graph $\tilde{G}$ equals G or any anonymized version of G depending on the privacy level guaranteed by the provider. The identity disclosure occurs if an attacker can identify the target individual in the released dataset $\tilde{G}$. In other words, the identity is disclosed if the attacker can link $v \in V(\tilde{G})$ with the particular individual that is represented with v.

Definition 9 (Query (Zou et al 2009)) Given a social network G, a query $\mathcal{Q}$ represents any information that the attacker can exploit to extract private information from G. The result of $\mathcal{Q}$ is a set of vertices $W \subset V(G)$. Each $w \in W$ is called a match vertex.

Definition 10 (Structural attack (Zou et al 2009)) Given a released network $\tilde{G}$, if a query $\mathcal{Q}$ over $\tilde{G}$ launched by an attacker has a limited number of match vertices in $\tilde{G}$, then target individual t might be uniquely identified. If $\mathcal{Q}$ is based on the structural information about t in $\tilde{G}$, this is called a *structural attack*.

Structural attacks include degree attacks, subgraph attacks, neighbor graph attacks and hub fingerprint attacks (Zou et al 2009). The presented k-automorphism concept defends against all kinds of structural attacks since for every vertex $v \in V(G^*)$, there are at least other $k-1$ vertices with the same l-neighborhood in G^* for any $l \in \mathcal{N}$. Therefore, the result of any structural query $\mathcal{Q}$ on G^* contains at least k vertices. The attacker cannot identify their target node with a higher probability than $\frac{1}{k}$ in the k-automorphic graph G^*.

5 Theoretical part

Before we describe the proposed solution to the issue stated in the previous section, we present some theoretical aspects of our solution.

5.1 Anonymization cost

The information loss caused by the anonymization is called the *anonymization cost*. In the edge editing anonymization methods, the anonymization cost corresponds to the number of edge edits made during the anonymization process. Since making a graph k-automorphic also requires node edits in the input graph, the total anonymization cost of HAkAu equals the sum of edge edits and node edits.

The number of node edits depends on $|V(G)|$. If G^* is k-automorphic, then for each node v there exist $k-1$ nodes that are isomorphic to v. Hence, $|V(G^*)|$ is divisible by k. If $|V(G)|$ is not divisible by k, adding or removing some vertices is necessary. Let us denote $z := mod(|V(G)|, k)$. In case $z \leq \frac{k}{2}$, then z vertices are removed by HAkAu; otherwise, $k-z$ dummy vertices are added. Thus, the number of node edits, denoted by $VCost(G, G^*)$, is equal to or less than $\frac{k}{2}$.

Edge edits have a more significant impact on the anonymization cost. HAkAu uses both edge-removing and edge-adding operations. Edge modifications are applied in two parts of HAkAu: when chosen subgraphs of G are extended by GA and in the adding crossing edges procedure. Hence, the total anonymization cost caused by modifying G to G^* with HAkAu is

$$Cost(G, G^*) = VCost(G, G^*) + ExCost(G, G^*) + CECost(G, G^*)$$

where $ExCost(G, G^*)$ is the number of edge edits made by GA and $CECost(G, G^*)$ is the number of edge edits made in the adding crossing edges procedure. Both costs, $ExCost(G, G^*)$ and $CECost(G, G^*)$, are computed in Sect. 6, after HAkAu is explained.

5.2 NP-hard problems

The issue addressed in this paper contains NP-hard problems. In this section, we present the problems and prove their NP-hardness. Methods for solving them are proposed in the next section.

Problem 1 Let G be a graph and M be a set of subgraphs of G. Select $S \subseteq M$ such that $\forall P_1, P_2 \in S : |V(P_1)| \cap |V(P_2)| = \emptyset$.

We prove the NP-hardness of Problem 1. Since all subgraphs in M are isomorphic, they have the same number of vertices. We define the matrix $\mathbf{M}$ such that the i-th row of $\mathbf{M}$ is the list of vertices of the i-th subgraph of M. The selection of the subset S corresponds to the selection of the set of rows in $\mathbf{M}$ that do not contain any identical number. The selected rows represent the isomorphic subgraphs with the mutually vertex disjoint set of vertices.

Now we show that the problem of selecting matrix rows that do not contain any identical number is polynomially reducible to the maximum independent set problem. This problem is known to be a strongly NP-hard (Garey and Johnson 1978). Let us suppose that the individual rows in $\mathbf{M}$ will be represented by vertices of a completely new graph (without any assumption of what kind of graph is represented by the specific row). Let us add edges in our newly created graph in such a way that there will be an edge between two vertices just when the intersection between the respective rows is non-empty. This simple transformation changed our problem into the maximum independent set problem. Hence, Problem 1 is NP-hard.

Problem 2 Let G be a graph and G^* be a k-automorphism graph such that $Cost(G, G^*)$ is minimal. Let $P_{i,1}, \ldots, P_{i,k}$, be subgraphs of G such that $P_{i,j}$ is isomorphic to $P_{i,l}$, $\forall j, l := 1, \ldots, k$. For $P_{i,1}, \ldots, P_{i,k}$ find graphs $P'_{i,1}, \ldots, P'_{i,k}$ and isomorphism $F_{i,j}$ such that

- $P'_{i,j}$ is subgraph of G^*
- $P'_{i,j}$ is the supergraph of $P_{i,j}$
- $F_{i,j}(P'_{i,j}) = P'_{i,j+1}$, $j = 1, \ldots, k-1$,
- $F_{i,k}(P'_{i,k}) = P'_{i,1}$

Problem 2 is the core of finding k-automorphic graph G^* for the given G. Naturally, G^* is not known before the anonymization procedure, and the real goal is to find G^* such that $Cost(G, G^*)$ is minimal. The problem repeatedly arises in HAkAu, and the index i denotes the number of its repetitions. It is a combination of two NP-hard problems proposed by Zou et al (2009): finding optimal graph partitioning of G and finding the optimal graph alignment for the set of subgraphs. We show how to reduce one of those problems to Problem 2 to show its NP-hardness.

Zou et al (2009) firstly find the frequent subgraph of G with the minimal support k and its matches in G. The found matches form the set of subgraphs denoted by U_i. Then subgraphs in U_i are extended to graphs $\overline{P_{i,j}}$, which are still subgraphs of G but are not isomorphic to each other. The extension is made so that $Cost(G, G^*)$ is minimal. The procedure is repeated until G is completely partitioned into sets U_i. This issue is called finding the optimal graph partitioning. Then, for each set U_i, some edges are added into $\overline{P_{i,j}}$ to create new graphs $\overline{P'_{ij}}$ that are isomorphic to each other. The edge addition procedure requires minimization of $Cost(G, G^*)$. The alignment vertex table defined in Zou et al (2009) described how vertices are mapped to each other under the isomorphism. This issue is called finding the optimal graph alignment.

To reduce the finding of the optimal graph alignment problem to Problem 2, we set $P_{i,1}, \ldots, P_{i,k}$ to be the matches of the frequent subgraph with the support k. Then $U_i := \{P_{i,j}; j := 1, \ldots, k\}$ before the extension is applied. After all $P'_{i,j}$ and $F_{i,j}$ are found for each i, j with HAkAu, then subgraphs $\overline{P'_{i,j}}$ requested in finding the optimal graph alignment problem are $\overline{P'_{i,j}} := P'_{i,j}$. The graphs $\overline{P'_{i,j}}$ are isomorphic. Moreover, the isomorphisms $F_{i,j}$ give the alignment vertex table.

6 HAkAu algorithm

In this section, we give a detailed description of the novel Hybrid Algorithm for k-Automorphism anonymization. The proposed HAkAu modifies the graph G representing the given SN to the anonymized k-automorphism graph G^*. The algorithm addressed the privacy-preserving problem outlined in Sect. 4. The final G^* is resistant to any structural attack.

The crucial idea of our approach is as follows. The k isomorphic subgraphs are found in G. They are isomorphically extended to minimize $Cost(G, G^*)$. Then, the extended isomorphic subgraphs are removed from the input graph, and

the process is rerun on the smaller graph. After the whole input graph is processed, we get the set of disconnected graphs such that for every graph, there are at least $k-1$ other graphs that are isomorphic to it. The disconnected graphs are linked together, making the final graph k-automorphic.

Algorithm 1 HAkAu algorithm

Require: anonymization parameter k, input network G, minimal support s
Ensure: k-automorphism network G^*

1: Set $i := 1$, $H := G$, $C := \emptyset$, $M := \emptyset$, $S := \emptyset$ and G^* to be an empty graph. Set $F_j := 0$, $j := 1, \ldots, k$.
2: **while** $|V(H)| \geq k$ **do**
3: Run GraMi on (H, s) to find the frequent subgraph $g_f(s)$ and the set $M := \{I_1, \ldots, I_s;\ I_j$ is a match of $g_f(s)$ in $H, j := 1, \ldots, s\}$.
4: Run *Algorithm 2* on (M, s, k) to find the set of k vertex-disjoint matches $S := \{P_{ij} \in M;\ V(P_{ij}) \cap V(P_{il}) = \emptyset, j \neq l, j, l = 1, \ldots, k\}$ (see *Section 6.1*)
5: Run GA on (S, H, k) to find graphs P'_{ij} and isomorphism F_{ij} such that
- P'_{ij} is the supergraph of P_{ij}
- $F_{ij}(P'_{i,j}) = P'_{i,j+1},\ j = 1, \ldots, k-1,$
- $F_{ik}(P'_{i,k}) = P'_{i,1}$
- $Cost(G, G^*)$ is minimal

(see *Sections 7*)
6: $C_i := \{(u, v) \in H : \exists j \in \{1, \ldots, k\} : u \in V(P'_{ij}) \wedge v \notin V(P'_{ij})\}$
7: $C := C \cup C_i$
8: $V(H) := V(H) \setminus \bigcup_{j=1}^{k}(V(P'_{ij}) \cap V(H))$
9: $E(H) := E(H) \setminus (C \cup \bigcup_{j=1}^{k}(E(P'_{ij}) \cap E(H)))$
10: $F_j := F_j \oplus F_{ij}\ j := 1, \ldots, k$
11: $i := i + 1$.
12: **end while**
13: $m := i - 1$
14: $C := C \cup E(H)$
15: **if** $|V(H)| \geq \frac{k}{2}$ **then**
16: $m := m + 1$
17: Add $k - |V(H)|$ dummy edges in $V(H)$.
18: **for** $j := 1, \ldots, k$ **do**
19: Select $v \in V(H)$ and set $P'_{mj} := v$
20: $V(H) := V(H) \setminus \{v\}$
21: **end for**
22: **end if**
23: $G^* := \bigcup_{i=1}^{m} \bigcup_{j=1}^{k} P'_{ij}$
24: Run *Algorithm 3* on (C, G, G^*, k, F_j) to add selected crossing edges and their isomorphic copies in G^* (see *Section 6.2*).
25: Return G^*.

The detailed description of HAkAu is given in Algorithm 1. The graph H is the rest of the input graph after the ith round of the *while* cycle (see line 2). While there are at least k vertices in H, H is partitioned. GraMi algorithm is run to find the frequent subgraph $g_f(s)$ and s matches of $g_f(s)$ in H (see line 3). If GraMi finds more than one frequent subgraph, $g_f(s)$ is the largest. The set of subgraphs matching $g_f(s)$ in H is denoted by M. After selecting k vertex disjoint subgraphs P_{ij} from M (see line 4), the GA is run to find isomorphic supergraphs P'_{ij} (see line 5). The proposed GA solves the NP-hard Problem 2. The graphs P'_{ij} are removed from H (see lines 8 and 9) and the crossing edges between P'_{ij} and H are added into the set of crossing edges C (see line 7). When $|V(H)| < k$, the edges remaining in H are added into C, and the remaining vertices become the last found isomorphic subgraphs (see lines 14–20). All the found subgraphs P'_{ij} are the core of the anonymized graph G^*. To make G^* connected, selected crossing edges and their copies are isomorphically added into G^* (see line 24). The detailed description of adding crossing edges procedure is given in Sect. 6.2.

Note the following interesting aspects of the algorithm. At first, we explain the issue of finding isomorphisms F_j. In the first round of the *while* cycle GA finds k isomorphic graphs $P'_{11}, \ldots, P'_{1k}$. The set of k isomorphic graphs determines k isomorphisms $F_{11}, \ldots, F_{1k}$ as shown in line 10.

Since F_j is a zero homomorphism at the beginning, $F_j := F_{1j}$ after the first round. After the second round, F_j is the extension of F_{1j} and F_{2j} and it is still the isomorphism, since $V(P'_{1j}) \cap V(P'_{2j}) = \emptyset$, $\forall j = 1, \ldots, k$. Similarly, after the i-th round F_j is extended with F_{ij} (see line 10). After the *while* cycle ends, there are $m \cdot k$ graphs P'_{ij}, $i = 1, \ldots, m$, $j = 1, \ldots, k$, and k isomorphisms F_j:

- $F_j(P'_{i,j}) = P'_{i,j+1}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, k-1,$
- $F_k(P'_{i,k}) = P'_{i,1}, \quad i = 1 \ldots, m$

Secondly, note that P_{ij} are subgraphs of H, but P'_{ij} are not subgraphs of H. Naturally, $V(P_{ij}) \subseteq V(P'_{ij})$ and $E(P_{ij}) \subseteq E(P'_{ij})$, since P_{ij} are supergraphs of P_{ij}. Moreover, $V(P'_{ij}) \subseteq V(H)$, but $E(P'_{ij}) \nsubseteq E(H)$.[1] Some edges from H are preserved in P'_{ij}, some new edges can be added between nodes of $V(P'_{ij})$ by GA, some edges that were between some nodes of $V(P'_{ij})$ in H do not exist in P'_{ij}.

Thirdly, the aim of GA is to set up graphs P'_{ij} such that they are isomorphic to each other and $Cost(G, G^*)$ is minimal. Putting back a single crossing edge caused adding other $k-1$ edges that are isomorphic to the graph. By finding larger graphs P'_{ij} and replacing them with smaller P_{ij}, the amount of crossing edges is reduced; hence, $CECost(G, G^*)$ is significantly reduced.

Finally, the HAkAu algorithm requires the input dataset and two parameters: the anonymization parameter k and the minimal support parameter s. The anonymization parameter is the independent parameter corresponding to the required anonymization level. The minimal support s is the dependent parameter. It holds that $s > k$ since we select k vertex disjoint subgraphs from the set of s edge disjoint subgraphs.

6.1 Finding the subset of vertex disjoint subgraphs

GraMi algorithm has been utilized to find the largest frequent subgraph $g_f(s)$ and its matches in H. We denoted

$$M := \{I_1, \ldots, I_s; I_j \text{ is a match of } g_f(s) \text{ in } H, j := 1, \ldots, s\}.$$

Naturally, there might be partially overlapping subgraphs in M having one or more common vertices. To proceed further, we need to find k mutually vertex disjoint subgraphs in M.

Algorithm 2 Finding the subset of mutually vertex-disjoint subgraphs

Require: the set of s isomorphic subgraphs $M = \{I_1, \ldots, I_s\}$, the anonymization parameter k
Ensure: the set of k vertex-disjoint isomorphic subgraphs S
1: Set $\{v_1^j, \ldots, v_n^j\} := V(I_j)$ and $\mathbf{M} := \{v_i^j\}_{i,j}$ to be a matrix, $i := 1, \ldots, |V(I_1)|$, $j := 1, \ldots, s$.
2: $\mathbf{R} := \emptyset$, $\mathbf{S} := \emptyset$.
3: **while** $rc(\mathbf{M}) > 1$ **do**
4: Calculate the frequency of all vertices contained in $\mathbf{M}$. Let v be the vertex with the highest frequency of occurrence in $\mathbf{M}$.
5: **for** $j := 1, \ldots, rc(M)$ **do**
6: **if** $v \in r_j(M)$ **then**
7: $\mathbf{M} := \mathbf{M} \setminus r_j(\mathbf{M})$
8: $\mathbf{R} := \mathbf{R} \cup r_j(\mathbf{M})$
9: **end if**
10: **end for**
11: **if** $rc(\mathbf{M}) = 1$ **then**
12: $\mathbf{S} := \mathbf{M}$.
13: **end if**
14: **end while**
15: **repeat**
16: Calculate the frequency of vertices contained in $\mathbf{S}$.
17: Find $r := r(\mathbf{R})$ such that $\forall v \in r(\mathbf{R})$ frequency of v in $\mathbf{S}$ equals 0.
18: $\mathbf{R} := \mathbf{R} \setminus r$
19: $\mathbf{S} := \mathbf{S} \cup r$
20: **until** $r = \emptyset$ or $rc(\mathbf{S}) = k$
21: Return $S := \{I \in M;\ \exists i : r_i(S) = V(I)\}$

[1] The statement $V(P'_{ij}) \subseteq V(H)$ simplifies slightly the real situation. In some cases, a few new nodes have to be added, and for some $j : V(P'_{ij}) \nsubseteq V(H)$ (see Sect. 7.2).

The output of the GraMi algorithm is in tabular form with s rows, where each row represents one subgraph I_j as the list of its vertices. Since $I_1, \ldots, I_s$ are isomorphic, $|V(I_1)| = \cdots = |V(I_s)|$. The set of the rows can be represented by a matrix $\mathbf{M}$ with s rows and $|V(I_1)|$ columns. The issue is formulated in Problem 1 and is proved to be NP-hard in Sect. 5.

Therefore, a naive approach based on a brute-force algorithm that successively compares each row with another one is impractical, and we need some heuristic approach to speed up the search. Our idea is based on the "divide and conquer" design paradigm where the frequency of the individual vertices (numbers) in matrix $\mathbf{M}$ is utilized to break down the main task into two subtasks. In the first subtask, the rows of $\mathbf{M}$ containing the most frequent vertices are step by step removed and stored in auxiliary matrix $\mathbf{R}$. The last row remaining in $\mathbf{M}$ is the cornerstone of the solution represented by the matrix $\mathbf{S}$. In the second subtask, we select rows from $\mathbf{R}$ that are disjoint with rows already stored in $\mathbf{S}$ and add them step by stem into $\mathbf{S}$. The algorithm terminates if it can find no such a row in $\mathbf{R}$ or $\mathbf{S}$ have k rows. The algorithm described in detail in Algorithm 2 delivers a subset of mutually vertex disjoint subgraphs.

6.2 Adding crossing edges

The crossing edges are edges stored in the set C. Note that a crossing edge (v, w) is the edge that connects vertices from different subgraphs, $v \in V(P'_{ij})$, $w \in V(P'_{ab})$, where $i \neq a$ or $j \neq b$. If we add (v, w) into G^*, then it is necessary to add other $k-1$ edges $(F_j(v), F_j(w))$ into G^*, $j = 1, \ldots, k-1$, to keep G^* k-automorphic.

The crossing edge (v, w) is added into G^* if there are at least $\frac{k}{2} - 1$ other crossing edges in C that are isomorphic to (v, w). Hence, adding (v, w) into G^* causes fewer edge edits than not adding it. Except that, HAkAu adds crossing edges that are significant for matching the structure of G^* to the structure of G. The procedure, including also the computation of $CECost(G, G^*)$, is described in detail in Algorithm 3.

Algorithm 3 Adding crossing edges

Require: the set of crossing edges C, input network G, anonymized network G^*, anonymized parameter k, isomorphisms F_j $(j := 1, \ldots, k)$
Ensure: k-automorphism network G^*, $CECost(G, G^*)$

```
CECost(G, G*) := 0
while C ≠ ∅ do
    Select (v, w) from C.
    Set C̃ := {(F_j(v), F_j(w)) ∈ C}.
    if |C̃| ≥ k/2 − 1 or (deg_G*(v) < deg_G(v) or deg_G*(w) < deg_G(w)) then
        E(G*) := E(G*) ∪ {(F_j(v), F_j(w)), j := 1, ..., k}
        C := C \ {C̃ ∪ (v, w)}
        CECost(G, G*) := CECost(G, G*) + k − 1 − |C̃|
    else
        C := C \ {(v, w)}
        CECost(G, G*) := CECost(G, G*) + 1
    end if
end while
Return G*, CECost(G, G*).
```

6.3 Computing the extension cost

The extension cost $ExCost(G, G^*)$ is the anonymization cost caused by replacing P_{ij} with P'_{ij}, $i := 1, \ldots, k$, $j := 1, \ldots, m$. Let $ExCost_i(H)$ denote the extension cost caused in the i-th round of *while* cycle in Algorithm 1 where H is processed. Then

$$ExCost(G, G^*) := \sum_{i=1}^{m} ExCost_i(H)$$

$$ExCost_i(H) := \sum_{j=1}^{k} ExCost(P_{ij}, P'_{ij})$$

where $ExCost(P_{ij}, P'_{ij})$ is the anonymization cost caused by extending P_{ij} to P'_{ij}, i, j fixed. The cost $ExCost(P_{ij}, P'_{ij})$ equals

to the number of edges that exist in P'_{ij} and not exist in G plus the number of edges $(u, v) \in E(G)$ such that $u, v \in V(P'_{ij})$ and $(u, v) \notin E(P'_{ij})$. More precisely,

$$ExCost(P_{ij}, P'_{ij}) := |E(P'_{ij}) \setminus (E(Pij') \cap E(G))| + |\{(u, v) \in E(G); u, v \in V(P'_{ij}) \wedge (u, v) \notin E(P'_{ij})\}|$$

7 Genetic algorithm

We used the model of the genetic algorithm described in Sect. 3.2. In this section, we describe the chromosome representation, fitness and selection function, and how genetic operators are applied to the used representation.

7.1 Chromosome representation

In this section, we introduce the novel chromosome representation used in GA. The goal of GA is to find k graphs P'_{ij}, $j := 1, \ldots, k$, i fixed that are isomorphic to each other (line 5 in Algorithm 1). Therefore, in the rest of this section, the index i is fixed, and indices $j, l := 1, \ldots, k$.

Each individual in GA represents one solution; hence, each individual represents all graphs P'_{ij}. For all j, the graph P'_{ij} is the supergraph of P_{ij}. We denote $Q_{ij} = P'_{ij} \setminus P_{ij}$ to be the subgraph of P'_{ij} with $V(Q_{ij}) = V(P'_{ij})$ and $E(Q_{ij}) = E(P'_{ij}) \setminus E(P_{ij})$ (see Fig. 1). The graphs P_{ij} are found with Algorithm 2, and they do not change during the run of GA. Thus, GA aims to find the optimal Q_{ij} for each j. Since $P'_{ij} \simeq P'_{il}$ and $P_{ij} \simeq P_{il}$, then $Q_{ij} \simeq Q_{il}$.

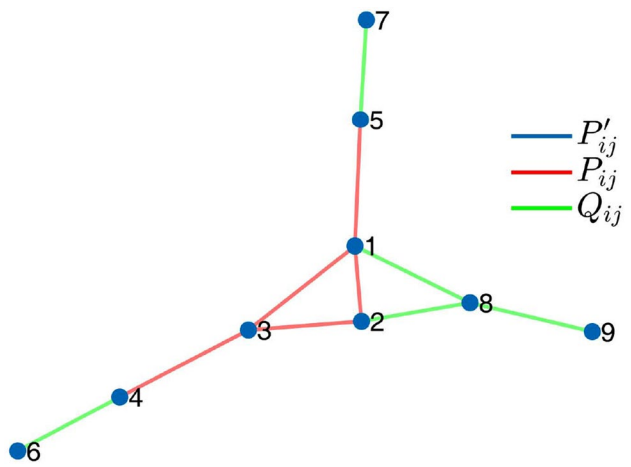


Fig. 1 Subgraphs of P'_{ij}. Vertex sets of particular subgraphs are $V(P'_{ij}) = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $V(P_{ij}) = \{1, 2, 3, 4, 6\}$, $V(Q_{ij}) = \{1, 2, 4, 5, 6, 7, 8, 9\}$

Furthermore, since $P'_{ij} \simeq P'_{il}$, then $P'_{i1}, \ldots, P'_{ik}$ have the same adjacency matrix (see the example with P'_{i1} and P'_{i2} in Fig. 2). Let us denote the adjacency matrix of $P'_{i1}, \ldots, P'_{ik}$ by $\mathbf{Adj}_i$. The part of $\mathbf{Adj}_i$ representing edges of P_{ij} is known before GA is run and corresponds to $g_f(s)$ found with GraMi (line 3 in Algorithm 1). Thus, the part of $\mathbf{Adj}_i$ representing edges of P_{ij} is constant in all possible solutions of GA and all individuals in all generations. Therefore, to encode the individuals in chromosomes, it is enough to encode some part of $\mathbf{Adj}_i$ and the ordered lists of vertices of Q_{ij}. Hence, each chromosome consists of two parts

- CH = bits representing elements of $\mathbf{Adj}_i$
- $varCH$ = ordered lists of vertices from $V(Q_{i1}), \ldots, V(Q_{ik})$

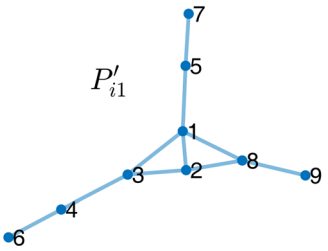
More precisely, CH represents the elements of $\mathbf{Adj}_i$ corresponding to $E(Q_{ij})$ and $(u_{ij}, v_{ij}) \in E(P'_{ij}) : u_{ij} \in V(Q_{ij}) \wedge v_{ij} \in V(P_{ij})$ (see Fig. 3). The chromosome representation guarantees that each individual corresponds to graphs $P'_{i1}, \ldots, P'_{ik}$ that are supergraphs of $P_{i1}, \ldots, P_{ik}$ and $P'_{ij} \simeq P'_{il}$. Hence, the representation keeps the k-automorphism in the final solution G^*, and it is unnecessary to check the k-anonymity property during GA processing.

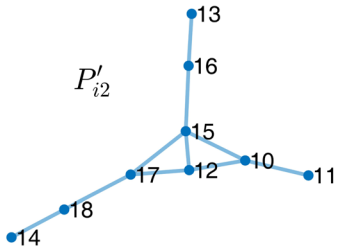
7.2 Fitness function

A fitness function $FF(I)$ evaluates how close the solution represented by the individual I is to the optimal solution of the problem. The aim of GA in HAkAu is to find the solution with minimal $Cost(G, G^*)$. GA runs several times in HAkAu (see line 5 in Algorithm 1). In each run, GA finds the optimal graphs $P'_{i1}, \ldots P'_{ik}$ for fixed i. Thus, minimizing $ExCost_i(H)$ in each run of GA is necessary. Except that, $CECost(G, G^*)$ should be minimized. However, during the particular run of GA, it is impossible to compute $CECost(G, G^*)$ properly since the decision, whether to add a crossing edge in G^* or not is made after all runs of GA. $CECost(G, G^*)$ is directly proportional to the number of crossing edges between $V(P'_{ij})$ and other nodes of H. Hence, instead of minimizing $CECost(G, G^*)$ in GA, we minimize the number of crossing edges. Thus, the fitness function of the individual I is defined as the two-tuple

$$FF(I) = \left[nCE(H, I); ExCost(H, I)\right]$$

where $ExCost(H, I)$ means $ExCost_i(H)$ and $nCE(H, I)$ means the number of crossing edges between P'_{ij} and H, $j := 1, \ldots, k$, i fixed, where P'_{ij} **are constructed according to the individual** I. Referring to the line 6 in Algorithm 1





Adjacency matrix of P'_{i1}:

	1	2	3	4	5	6	7	8	9
1	0	1	1	0	1	0	0	1	0
2	1	0	1	0	0	0	0	1	0
3	1	1	0	1	0	0	0	0	0
4	0	0	1	0	0	1	0	0	0
5	1	0	0	0	0	0	1	0	0
6	0	0	0	1	0	0	0	0	0
7	0	0	0	0	1	0	0	0	0
8	1	1	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	1	0

Adjacency matrix of P'_{i2}:

	15	12	17	18	16	14	13	10	11
15	0	1	1	0	1	0	0	1	0
12	1	0	1	0	0	0	0	1	0
17	1	1	0	1	0	0	0	0	0
18	0	0	1	0	0	1	0	0	0
16	1	0	0	0	0	0	1	0	0
14	0	0	0	1	0	0	0	0	0
13	0	0	0	0	1	0	0	0	0
10	1	1	0	0	0	0	0	0	1
11	0	0	0	0	0	0	0	1	0

Graph isomorphism $F_1 : P'_{i1} \longrightarrow P'_{i2}$:

$F_1(1) = 15$ $F_1(4) = 18$ $F_1(7) = 13$
$F_1(2) = 12$ $F_1(5) = 16$ $F_1(8) = 10$
$F_1(3) = 17$ $F_1(6) = 14$ $F_1(9) = 11$

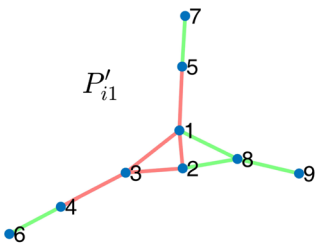
Fig. 2 Adjacency matrix of isomorphic graphs P'_{i1} and P'_{i2} ($k = 2$)

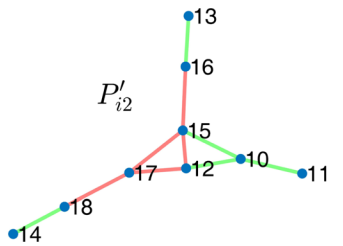
$nCE(H, I) = |C_i|$. If two values $FF(I_1)$ and $FF(I_2)$ are compared in GA, then

$$FF(I_1) \leq FF(I_2)$$

if $nCE(H, I_1) < nCE(H, I_2)$
or $nCE(H, I_1) = nCE(H, I_2) \wedge ExCost(H, I_1) \leq ExCost(H, I_2)$

The number of crossing edges has more weight while comparing FF values. It has three reasons. Firstly, the aim of GA is to search for the expansion of subgraphs found with GraMi. Why are the subgraphs P_{ij} found with GraMi not used? By the expansion of P_{ij}, $CECost(G, G^*)$ is reduced significantly. Thus, we aim to minimize $CECost(G, G^*)$ as much as possible during GA runs. Moreover, we experimentally found that $ExCost(H, I)$ is much smaller than $nCE(H, I)$.





$\mathbf{Adj}_i$	1	2	3	4	5	**6**	**7**	**8**	**9**
1	0	1	1	0	1	0	0	1	0
2	1	0	1	0	0	0	0	1	0
3	1	1	0	1	0	0	0	0	0
4	0	0	1	0	0	1	0	0	0
5	1	0	0	0	0	0	1	0	0
6	0	0	0	1	0	0	0	0	0
7	0	0	0	0	1	0	0	0	0
8	1	1	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	1	0

$\mathbf{Adj}_i$	15	12	17	18	16	**14**	**13**	**10**	**11**
15	0	1	1	0	1	0	0	1	0
12	1	0	1	0	0	0	0	1	0
17	1	1	0	1	0	0	0	0	0
18	0	0	1	0	0	1	0	0	0
16	1	0	0	0	0	0	1	0	0
14	0	0	0	1	0	0	0	0	0
13	0	0	0	0	1	0	0	0	0
10	1	1	0	0	0	0	0	0	1
11	0	0	0	0	0	0	0	1	0

$CH = 00010 \quad 000010 \quad 1100000 \quad 00000001$
$varCH = 6\,7\,8\,9 \quad 14\,13\,10\,11$
Whole chromosome: 000100000101100000000000016 7 8 9 14 13 10 11

Fig. 3 Chromosome representation for $k = 2$. The part of $\mathbf{Adj}_i$ representing $E(P_{ij})$ is highlighted with red (dark) color, and the part of $\mathbf{Adj}_i$ representing $E(Q_{ij})$ is highlighted with green (light) color. The part of $\mathbf{Adj}_i$ in the black tetragon makes the bit part of the chromosome CH. The bits are taken in columns. Ordered lists of nodes from $V(Q_{i1})$ and $V(Q_{i2})$ makes $varCH$

Finally, if one crossing edge is added in G^*, then other $k-1$ copies of it have to be added in G^* to keep G^* k-automorphic. On the other hand, increasing $ExCost(H, I)$ by one corresponds only to the single change in G^*.

7.3 Selection function

The selection function chooses individuals from the current population to create the next generation. We propose a two-step selection function. Let N be the number of requested parents. In the first step, $2N$ individuals are selected from the whole current population using the roulette wheel selection where the expectations are computed with $nCE(H, I)$. In the second step, N parents are selected from the $2N$ chosen individuals using tournament selection where expectations are computed with $ExCost(H, I)$.

7.4 Genetic operators

We use the usual genetic operations: crossover and mutation. Crossover combines two parent individuals I_1 and I_2 into a single child individual I_{1+2}. Let CH_1, CH_2 and CH_{1+2} be the bit parts of chromosomes corresponding to I_1, I_2 and I_{1+2}, respectively, and $varCH_1$, $varCH_2$ and $varCH_{1+2}$ be the "node" parts of chromosomes corresponding to I_1, I_2 and I_{1+2}, respectively. CH's and $varCH$'s are crossed over separately.

The two-point crossover is applied on CH_1 and CH_2 (Hynek 2008). CH_1 and CH_2 have the same length l_{CH}. We select two random integers $a, b \in (1; l_{CH})$. The bit part of the child individual CH_{1+2} gets 1st, … , a-th bit from CH_1, $(a+1)$ -th, … , b-th bit from CH_2 and $(b+1)$-th, … , l_{CH}-th bit from CH_1 (see Fig. 4).

Before $varCH_1$ and $varCH_2$ are crossed over, $varCH_1$ and $varCH_2$ are cut into k segments corresponding to $Q_{i1}, \dots, Q_{ik}$. The two-point crossover is applied to every segment. The i-th segment of $varCH_1$ is crossed with the i-th segment of $varCH_2$, $i = 1, \dots, k$. We select two random integers c, d between 1 and the length of the segment. Then all segments are crossed at the same points corresponding to c and d (see Fig. 4).

The probability that an individual is mutated is given by the mutation rate. Then it is randomly decided whether CH or $varCH$ is mutated. When CH is mutated, one random bit is reversed in CH. When $varCH$ is mutated, one random vertex from $varCH$ is replaced with a new one.

7.5 Heuristic for selecting new vertices proportionally to their degree

If it is necessary to add a new vertex in $varCH$, then the vertex is selected from the set of unused vertices $V(H) \setminus V(P'_{ij})$, i fixed, $j = 1, \dots, k$. There are three situations where vertices are added to $varCH$:

- Creating initial population. The bit parts of chromosomes CH are generated randomly. The vertices are added one by one to $varCH$ from the unused vertices.
- When CH_1 and CH_2 are crossed over, $\mathbf{Adj}_i$ corresponding to CH_{1+2} can lead to graphs with more vertices than the parents' graphs. Hence, it is necessary to add new vertices to $varCH_{1+2}$. Similarly, CH after mutation can lead to a graph with more vertices.
- $varCH$ is mutated.

Selecting new vertices proportionally to their degree reduces $nCE(H, I)$. The bit string CH is usually spare after several generations of GA. Most vertices in $varCH$ are connected with P_{ij} with a single edge. Hence, their other edges become crossing edges. The fewer links the vertices in $varCH$ have, the fewer crossing edges are produced. Thus, we use a stochastic selection method where the probability of selecting the vertex is proportional to its degree. More precisely, we choose the new vertex to $varCH$ with roulette selection where expectations are computed with the metric $\frac{1}{deg(v)}$, where $deg(v)$ is the vertex's degree.

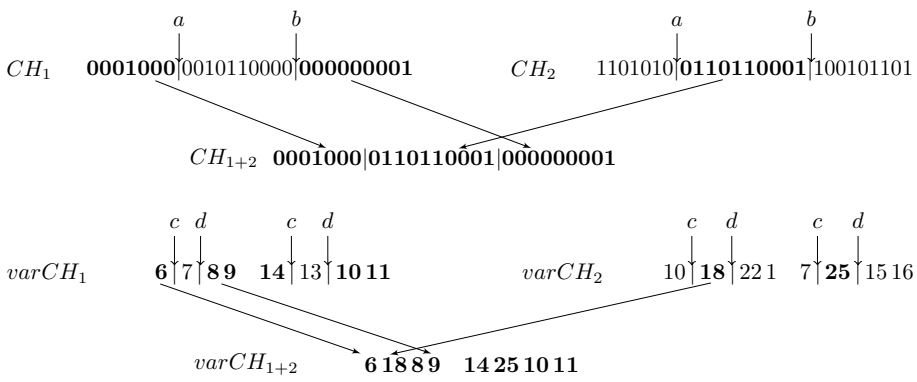


Fig. 4 Crossover operation in GA

8 Experimental results

In this section, the results of accomplished experiments with real-world networks are presented. All experiments were performed on a Windows 10 operating system PC with 8 GB RAM and a 3.2 GHz processor. The programs were written in Matlab 9.7.0.1261785 (R2019b). The used implementation of the GraMi algorithm is available at Elseidy and Abdelhamid (2014). The evaluation tool SecGraph is available at Ji and Li (2015). In all experiments, the mutation rate was set to 0.02 and the minimal support parameter $s = 2k$.

8.1 Tested datasets

We tested HAkAu on three real-world datasets of different sizes that are free to use: Prefuse (Heer et al 2005), Polblogs (Adamic and Glance 2005) and WikiVote (Leskovec et al 2010). We produced three kinds of results. At first, we compared HAkAu with the KM algorithm. We provided the comparison on the Prefuse dataset since the results of KM on Prefuse are presented by Zou et al (2009). Then, we provided data utility measurements on all three datasets. Finally, we tested the resistance against deanonymization attacks. Since the Prefuse dataset was too small to accomplish this testing, we provided results only on Polblogs and WikiVote. The features of all three datasets are summarized in Table 2.

8.2 The comparison of HAkAu and KM algorithm

We compared the performance of the HAkAu algorithm with the performance of the original k-automorphism algorithm KM algorithm. To compare our results with the ones presented by Zou et al (2009), we ran the experiments on the Prefuse dataset with the anonymization parameter $k \in \{5, 10, 15, 20\}$ and computed the following network metrics in the anonymized network:

- **Average clustering coefficient ACC** (Kemper 2009) the average of local clustering coefficients of all nodes in the graph, where the local clustering coefficient is the ratio of the number of triangles connected to the node and the number of triplets centered on the node

Table 2 Tested datasets

Datasets	#Nodes	#Edges	Repository
Prefuse	121	169	Heer et al (2007)
Polblogs	1225	1675	Rossi and Ahmed (2015)
WikiVote	7115	103, 689	Leskovec and Krevl (2014)

- **Average shortest path length APL** (Casas-Roma et al 2017)

$$APL = \frac{\sum_{i,j=1}^{n} dist(v_i, v_j)}{\binom{n}{2}}$$

where $dist(v_i, v_j)$ is the length of the shortest path from v_i to v_j, meaning the number of edges along the path

- **Total degree difference** the sum of the difference between the node degree in the original graph and its degree in the anonymized one

Since HAkAu is a non-deterministic algorithm, it was run ten times on each parameter setting. The experimental results are shown in Fig. 5. There is the mean of the ten metric values as well as the metric value of the best run.

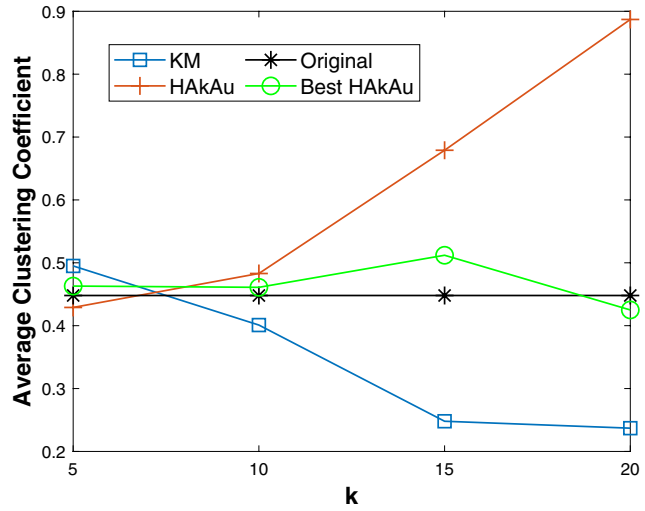
The clustering coefficient describes how well the neighborhood of a node is connected. If it is fully connected, the clustering coefficient is 1, whereas a value close to 0 implies hardly any connection (Kemper 2009). Hence, when k is larger, more edges are added by HAkAu, and ACC increases on average. However, in the best case, the algorithm can compile an anonymized graph with ACC very close to the original network. Interestingly, the ACC values of KM get lower with larger k even though KM only adds edges.

While the original graph is modified by adding edges, the distance between each pair of nodes is reduced (see Fig. 5b). If $k \geq 15$, the GraMi algorithm finds no frequent subgraph appearing at least 15 times in the original graph of the Prefuse network since the dataset is small. Hence, the subgraphs $P_{1,1}, \ldots, P_{1,15}$ inputting GA are only isolated vertices selected randomly.

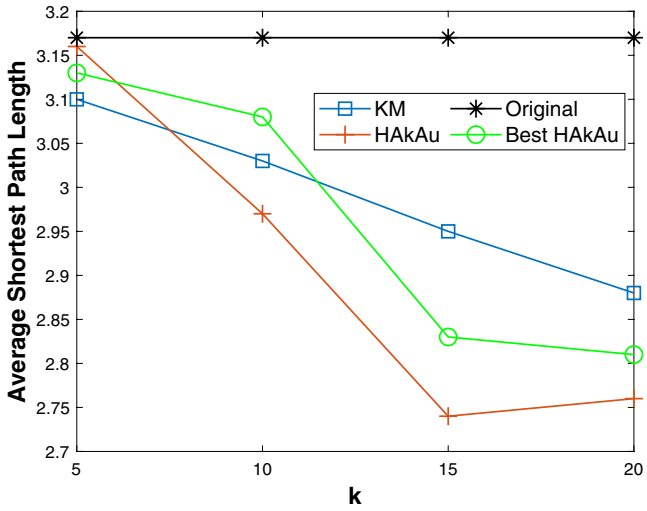
8.3 Data utility measurement

The HAkAu algorithm was further evaluated in preserving other network and application metrics. Three real social networks were anonymized with HAkAu, and the anonymized networks were evaluated using the SecGraph tool (Ji et al 2015; Ji and Li 2015).

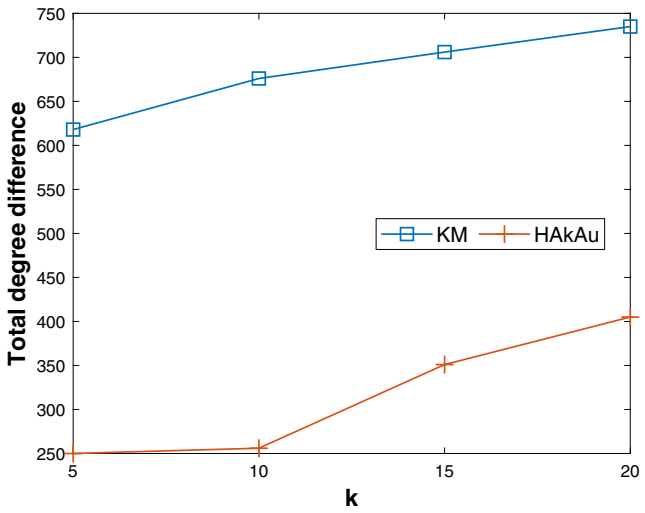
SecGraph is an independent evaluation tool that researchers can use to analyze the performance of their anonymization algorithms. The SecGraph has three modules: anonymization, utility, and deanonymization. In the anonymization module, graph data anonymization schemes are implemented. The module can be used to anonymize raw graph data. The utility module can evaluate anonymized data utility concerning utility and application metrics. Therefore, it can evaluate how an anonymization algorithm preserves data utility. In the deanonymization module, data security can be evaluated with real-world deanonymization algorithms before publishing or sharing. The effectiveness of an



a) Average clustering coefficient.



b) Average shortest path length.



c) Total degree difference.

Fig. 5 Comparison of HAkAu and KM algorithm while anonymizing Prefuse network with the anonymization parameter $k \in \{5, 10, 15, 20\}$

anonymization algorithm can be examined in this module as well. Researchers can test whether the anonymized data of an anonymization algorithm is resistant to attacks. We refer to Ji et al (2015) for the detailed description of the implemented anonymization schemes, utility metrics and deanonymization algorithms in SecGraph. Table 3 summarizes the abbreviation of SecGraph terms. SecGraph has been recently improved by adding two other modules, recommendation and security quantification. The second version of the tool is called ShareSafe (Tang et al 2019).

We used the following methodology of experiments. We select the dataset $D \in \{\text{Prefuse, Polblogs, WikiVote}\}$ and the parameter $k \in \{5, 10, 15, 20\}$. HAkAu was run ten times on D with k. Utility metrics were measured in all ten output graphs by the SecGraph utility module, and the mean value of each metric is shown in Table 4. The dataset D was anonymized using three schemes from the SecGraph anonymization module. Since HAkAu is the k-anonymization method, we select k-degree anonymization method kDA (Liu and Terzi 2008) and clustering methods tMean and Union (Thompson and Yao 2009) with parameter $t = k$. Utility metrics were measured in the graphs anonymized with kDA, tMean and Union as well (see Table 4). The values in Table 4 describe how each metric is preserved in the anonymized graph compared to the original one. More precisely, SecGraph measures the cosine similarity between the

Table 3 SecGraph abbreviations

Abbreviation	Term
*k*DA	*k*-degree anonymity (Liu and Terzi 2008)
*t*Mean	*t*-means clustering (Thompson and Yao 2009)
Union	Union-split clustering (Thompson and Yao 2009)
AS	Authorities score
BC	Betweenness centrality
CC	Closeness centrality
CD	Community detection
Deg	Degree distribution
ED	Effective diameter
EV	Eigenvector
HS	Hubs score
Infe	Infectiousness
JD	Joint degree distribution
LCC	Local clustering coefficient
NC	Network constraint
PR	Page rank
RX	Role extraction
NS	Narayanan–Shmatikov's attack (Narayanan and Shmatikov 2009)
Per	Yartseva–Grossglauser's attack (Yartseva and Grossglauser 2013)
Rec	Korula–Lattanzzi's attack (Korula and Lattanzi 2014)

Table 4 Utility measurement

	Prefuse				Polblogs				WikiVote			
k = 15	kDA	tMean	Union	HAkAu	kDA	tMean	Union	HAkAu	kDA	tMean	Union	HAkAu
AS	0.157	0.199	**0.383**	0.219	0.849	0.858	**0.901**	0.608	0.834	**0.886**	0.881	0.799
BC	0.528	**0.834**	**0.834**	0.615	0.904	0.971	**0.994**	0.420	0.885	0.966	**0.991**	0.477
CC	**0.995**	0.913	0.943	**0.995**	0.999	**1.000**	**1.000**	0.993	0.999	**1.000**	**1.000**	0.981
CD	0.214	**0.810**	0.715	0.102	0.857	0.756	**1.000**	0.061	0.838	0.755	**0.960**	0.018
Deg	0.795	0.973	**0.983**	0.779	0.977	0.991	**0.998**	0.826	0.965	0.992	**0.999**	0.817
ED	0.802	1.374	1.381	**0.826**	0.955	1.132	**0.987**	0.844	0.982	**1.013**	0.984	0.911
EV	0.904	0.411	0.692	**0.905**	0.957	0.966	**0.997**	0.792	0.977	0.989	**0.998**	0.800
HS	0.276	0.201	**0.381**	0.041	0.814	0.816	**0.881**	0.806	**0.876**	0.846	0.865	0.674
Infe	**0.694**	0.617	0.602	0.646	0.914	0.896	0.895	**0.928**	0.848	0.835	0.850	**0.870**
JD	**0.660**	0.029	0.124	0.005	**0.232**	0.230	0.224	0.080	0.318	0.353	**0.397**	0.087
LCC	0.972	0.970	**0.996**	0.880	0.955	0.905	**0.983**	0.827	0.988	0.988	**1.000**	0.774
NC	0.982	0.982	**0.993**	0.957	0.998	0.947	**1.000**	0.781	**1.000**	0.996	**1.000**	0.712
PR	0.572	0.537	0.544	**0.949**	0.518	0.498	0.487	**0.666**	0.607	**0.690**	0.678	0.588
RX	0.460	0.937	**0.933**	0.490	0.335	0.332	0.346	**0.398**	0.757	0.889	**0.971**	0.379

The best result for each parameter setting is highlighted in bold

metric's distributions in the anonymized and the original graph (Ji et al 2015). Due to the space limits, Table 4 contains only metric values for $k = 15$.

As we see in the next section, HAkAu offers a much higher level of security than other algorithms. Since deanonymization attacks exploit structural metrics, keeping a high level of security is paid with worse data utility preservation. The metrics affected the most by HAkAu anonymization are metrics based on the degree of nodes, like degree and joint degree distributions (Deg, JD), since the degree of many nodes is changed significantly by HAkAu. Communities in the graph are also heavily altered with HAkAu, which

Table 5 Resistance against deanonymization attacks

	Polblogs				WikiVote			
	kDA	tMean	Union	HAkAu	kDA	tMean	Union	HAkAu
k = 5								
NS	92.42%	90.61%	92.42%	**4.64%**	72.43%	72.61%	72.90%	**0.81%**
Per	48.83%	11.64%	60.02%	**4.96%**	9.57%	14.46%	37.40%	**0.92%**
Rec	96.00%	48.94%	98.94%	**4.74%**	98.30%	30.02%	99.55%	**0.79%**
k = 10								
NS	90.88%	91.88%	92.33%	**4.53%**	72.03%	72.67%	72.90%	**0.80%**
Per	18.77%	53.16%	63.00%	**4.65%**	2.18%	7.97%	33.11%	**0.89%**
Rec	91.99%	63.48%	97.14%	**4.28%**	92.92%	43.22%	98.40%	**0.75%**
k = 15								
NS	91.79%	91.25%	92.42%	**4.56%**	71.45%	72.84%	72.90%	**0.81%**
Per	20.76%	43.23%	36.73%	**4.65%**	8.32%	4.67%	35.42%	**0.85%**
Rec	81.54%	27.70%	84.72%	**4.29%**	90.89%	22.98%	97.44%	**0.75%**
k = 20								
NS	90.97%	88.09%	92.15%	**4.51%**	71.31%	72.53%	72.90%	**0.81%**
Per	17.51%	61.19%	56.32%	**4.59%**	2.83%	22.28%	8.47%	**0.84%**
Rec	69.12%	25.65%	86.36%	**4.23%**	81.26%	24.53%	94.35%	**0.74%**

The best result for each parameter setting is highlighted in bold

was proved by the low values of the community detection metric (CD).

On the other hand, HAkAu is best in preserving infectiousness (Infe) in 9 parameter settings (Prefuse: $k = 10$, Polblogs for all k, WikiVote for all k) and page rank (PR) in 6 parameter settings (Prefuse for all k, Polblogs: $k \in \{5, 15\}$).

Since HAkAu is non-deterministic, the metric values varied in ten anonymized networks with the same parameter settings obtained in ten runs. The coefficient of variation was lower in the metrics values that HAkAu better preserved. Except for JD and CD metrics, the coefficient of variation was up to 10%. Furthermore, the coefficient of variation was lower in larger networks.

8.4 Resistance to deanonymization attacks

In this section, we show that HAkAu is resistant to deanonymization techniques, and the resistance is guaranteed in every algorithm run. The methodology of experiments is similar to the one in the previous measurement. We select dataset $D \in \{\text{Polblogs}, \text{WikiVote}\}$ and parameter $k \in \{5, 10, 15, 20\}$. The HAkAu was run ten times on each parameter setting, and each output network was attacked with three deanonymization algorithms implemented in SecGraph: Narayanan–Shmatikov's attack (NS) (Narayanan and Shmatikov 2009), Yartseva–Grossglauser's attack (Per.) (Yartseva and Grossglauser 2013), and Korula–Lattanzzi's attack (Rec.) (Korula and Lattanzi 2014).

These seed-based passive attacks employ the structural similarity between the anonymized graph and the auxiliary graph to break the anonymity. The input of their procedures is the anonymized network G^*, auxiliary network G_{aux}, and seed mapping s. The auxiliary network G_{aux} is a fraction of the original network gained by the attacker before the passive attack. The seed mapping s is the mapping that links some nodes from G_{aux} with the ones in G^*. In our experiment, G_{aux} was sampled randomly with the probability of 90% from the original network. The seed s was set as 50 links between randomly selected nodes from G_{aux} and G^*.

The output of deanonymization procedures in SecGraph is the ratio of successfully deanonymized users. We present results in percentages in Table 5. The HAkAu values are the average ratios in the ten runs with the same parameter settings. We also tested the graphs anonymized with kDA, tMean and Union.

All results prove that the security level of the HAkAu algorithm is much higher than the security level of other tested algorithms. In the WikiVote network, the percentage of deanonymized users was up to 1% for all k. The resistance level is the same for all k values in both networks. The k-automorphism approach achieves better resistance against attacks compared to k-degree and clustering methods.

Although HAkAu is heuristic and non-deterministic, the percentage of deanonymized users did not vary in the ten runs with the same parameter settings. The coefficient of variation within the ten runs was up to 4% in all cases, except the instance with D =WikiVote and $k = 10$, where the coefficient of variation equals 10%. However, the level of security was the same since the best value equalled 0.85% and the worst one to 1.09% when D =WikiVote and $k = 10$.

8.5 Discussion

The comparison of KM and HAkAu is limited since only two structural metrics and the total degree difference are measured in Zou et al (2009). The total degree difference is significantly lower while applying HAkAu. Using edge deletion operation in the procedure of adding crossing edges decreases the amount of added edges and the final degree of all nodes. The application of edge deletion operation was enabled by improving the design of the k-automorphism algorithm.

The experiments show that the fitness function is not designed to keep the APL property well. If APL was the critical property that should not change during the anonymization process, then the fitness function would have to be modified to consider distances between vertices.

More structural metrics are measured with the SecGraph tool. Even though HAkAu does not exceed in preserving all utility metrics, it keeps Page Rank and Infectiousness very well. Both metrics are centrality metrics that can identify influential users in the graph. Thus, the importance of nodes is preserved in the network anonymized by HAkAu. The final k-automorphism maps important nodes to each other. Preserving infectiousness indicates that the communication channels in the anonymized network are kept very well even though the graph structure is changed significantly by HAkAu.

Unlike other tested algorithms, HAkAu was shown to be resistant to deanonymization attacks. It proves that the k-automorphism approach provides a higher level of security than other solutions. Moreover, HAkAu provides the same level of resistance for all kinds of tested attacks and all values of the anonymization parameter.

9 Conclusion

In this paper, we proposed a hybrid anonymization algorithm HAkAu that modified the social network to the k-automorphism network. The core of HAkAu is the genetic algorithm with the novel chromosome representation in which the length of the chromosome is independent of the size of the input network, and each individual in each generation leads to the k-automorphism solution. Moreover, we presented the efficient algorithm for finding the largest subset of mutually vertex disjoint subgraphs that effectively prepare the input for the genetic algorithm. We also introduced heuristics that improved the postprocessing stage of the genetic algorithm.

HAkAu makes users indistinguishable with respect to all structural semantics with the probability of $1 - \frac{1}{k}$. The experiments on real social networks showed that HAkAu was resistant to deanonymization attacks and provided better security than other solutions. The structural changes enabling the high resistance caused worse data utility preservation in the anonymized network. However, HAkAu can preserve utility metrics measuring centrality properties in the graph and keep the position of important nodes in the anonymized network.

While presenting our experimental results, we emphasize keeping the results comparable with any future studies. To keep experimental results comparable in the future, we tested HAkAu on datasets that are available to other researchers. We selected the datasets with the size corresponding to the tested algorithm's computation capabilities. Moreover, we used the evaluation tool SecGraph for evaluating experimental results.

The paper highlights several new avenues that could be explored in future research. Implementing the HAkAu algorithm can be further improved with more sophisticated parameter settings in the genetic algorithm. Estimating the optimal running strategy could raise the quality of the search process and the found results. The proposed chromosome representation can be applied to other genetic algorithms dealing with anonymization tasks. The possibility of exploiting the representation in other k-anonymization methods based on genetic algorithms can also be explored. Similarly, the procedure for finding the vertex disjoint subgraphs has the potential for broader application.

Acknowledgements The authors are grateful to the anonymous reviewers for their time and valuable comments. The authors are also grateful to researchers in developing SecGraph: Ada Fu, Michael Hay, Davide Proserpio, Qian Xiao, Shirin Nilizadeh, Jing S. He, Wei Chen, and Stanford SNAP developers. The authors are thankful to Mohammed Elseidy, Ehab Abdelhamid, Spiros Skiadopoulos, and Panos Kalnis, who developed and provided the GraMi source code. This study is supported by the SPEV 2023 project run at the Faculty of Informatics and Management, University of Hradec Kralove, Czech Republic.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

Adamic LA, Glance N (2005) The political blogosphere and the 2004 US election: divided they blog. In: Proceedings of the 3rd international workshop on link discovery. ACM Press, Chicago, pp 36–43. https://doi.org/10.1145/1134271.1134277

Alavi A, Gupta R, Qian Z (2019) When the attacker knows a lot: the GAGA graph anonymizer. In: Information security. Springer, Cham, pp 211–230. https://doi.org/10.1007/978-3-030-30215-3_11

Azaouzi M, Rhouma D, Ben Romdhane L (2019) Community detection in large-scale social networks: state-of-the-art and future directions. Soc Netw Anal Min 9(1):1–32. https://doi.org/10.1007/s13278-019-0566-x

Bello-Orgaz G, Menéndez HD, Camacho D (2012) Adaptive K-means algorithm for overlapped graph clustering. Int J Neural Syst. https://doi.org/10.1142/S0129065712500189

Bhattacharya M, Roy S (2015) Prevention of walk based attack on social network graphs using ant colony optimization. In: 2015 international conference and workshop on computing and communication (IEMCON), pp 1–5. https://doi.org/10.1109/IEMCON.2015.7344432

Cai Q, Gong M, Ma L et al (2015) Greedy discrete particle swarm optimization for large-scale social network clustering. Inf Sci 316:503–516. https://doi.org/10.1016/j.ins.2014.09.041

Campan A, Truta TM (2008) Data and structural k-anonymity in social networks. In: International workshop on privacy, security, and trust in KDD. Springer, Berlin, pp 33–54. https://doi.org/10.1007/978-3-642-01718-6_4

Casas-Roma J, Herrera-Joancomartí J, Torra V (2017) K-degree anonymity and edge selection: improving data utility in large networks. Knowl Inf Syst 50(2):447–474. https://doi.org/10.1007/s10115-016-0947-7

Caschera MC, D'Ulizia A, Ferri F et al (2019) MONDE: a method for predicting social network dynamics and evolution. Evol Syst 10(3):363–379. https://doi.org/10.1007/s12530-018-9242-z

Cheng J, Fu AWC, Liu J (2010) K-isomorphism: privacy preserving network publication against structural attacks. In: Proceedings of the ACM SIGMOD international conference on management of data. ACM Press, New York, pp 459–470. https://doi.org/10.1145/1807167.1807218

Chester S, Kapron BM, Srivastava G et al (2013) Complexity of social network anonymization. Soc Netw Anal Min 3:151–166. https://doi.org/10.1007/s13278-012-0059-7

Elseidy M, Abdelhamid E, Skiadopoulos S et al (2014) Grami: frequent subgraph and pattern mining in a single large graph. Proc VLDB Endow 7(7):517–528. https://doi.org/10.14778/2732286.2732289

Elseidy M, Abdelhamid E (2014) Grami. https://github.com/ehab-abdelhamid/GraMi. Accessed 13 Nov 2021

Garey MR, Johnson DS (1978) "Strong'' NP-completeness results: motivation, examples, and implications. J ACM 25(3):499–508

Hartung S, Hoffmann C, Nichterlein A (2014) Improved upper and lower bound heuristics for degree anonymization in social networks. In: International symposium on experimental algorithms. Springer, Copenhagen, pp 376–387. https://doi.org/10.1007/978-3-319-07959-2_32

Harvey J, Smith A, Goulding J et al (2020) Food sharing, redistribution, and waste reduction via mobile applications: a social network analysis. Ind Market Manag 88:437–448. https://doi.org/10.1016/j.indmarman.2019.02.019

Heer J, Card SK, Landay JA (2005) Prefuse: a toolkit for interactive information visualization. In: Proceedings of the SIGCHI conference on Human factors in computing systems. ACM Press, New York, pp 421–430. https://doi.org/10.1145/1054972.1055031

Heer J, Card SK, Landay JA (2007) Prefuse data. https://github.com/prefuse/Prefuse/blob/master/data/socialnet.xml. Accessed 13 Nov 2021

Holland JH (1973) Genetic algorithms and the optimal allocation of trials. SIAM J Comput 2(2):88–105

Hynek J (2002) Genetic algorithms in a nutshell. Econ Manag 5:48–54

Hynek J (2008) Genetické algoritmy a genetické programování. Grada Publishing, Prague

Ji S, Li W (2015) SecGraph home. https://nesa.zju.edu.cn/secgraph_pages/home.html. Accessed 13 Nov 2021

Ji S, Li W, Mittal P et al (2015) SecGraph: a uniform and open-source evaluation system for graph data anonymization and de-anonymization. In: 24th USENIX security symposium, pp 303–318

Kanai R, Bahrami B, Roylance R et al (2012) Online social network size is reflected in human brain structure. Proc R Soc B Biol Sci 279(1732):1327–1334. https://doi.org/10.1098/rspb.2011.1959

Kemper A (2009) Valuation of network effects in software markets: a complex networks approach. Springer, Berlin

Kiabod M, Naderi Dehkordi M, Barekatain B (2021) A fast graph modification method for social network anonymization. Expert Syst Appl 180(115):148. https://doi.org/10.1016/j.eswa.2021.115148

Korula N, Lattanzi S (2014) An efficient reconciliation algorithm for social networks. Proc VLDB Endow 7(5):377–388. https://doi.org/10.14778/2732269.2732274

Kuramochi M, Karypis G (2005) Finding frequent patterns in a large sparse graph. Data Min Knowl Disc 11(3):243–271. https://doi.org/10.1007/s10618-005-0003-9

Leskovec J, Huttenlocher D, Kleinberg J (2010) Predicting positive and negative links in online social networks. In: Proceedings of the 19th international conference on world wide web. ACM Press, New York, pp 641–650. https://doi.org/10.1145/1772690.1772756

Leskovec J, Krevl A (2014) SNAP datasets: Stanford large network dataset collection. http://snap.stanford.edu/data

Liu K, Terzi E (2008) Towards identity anonymization on graphs. In: Proceedings of the ACM SIGMOD international conference on management of data. ACM Press, Vancouver, Canada, pp 93–106. https://doi.org/10.1145/1376616.1376629

Lu X, Song Y, Bressan S (2012) Fast identity anonymization on graphs. In: Database and expert systems applications. Springer, Berlin, pp 281–295. https://doi.org/10.1007/978-3-642-32600-4_21

Macià M, García I (2016) Informal online communities and networks as a source of teacher professional development: a review. Teach Teach Educ 55:291–307. https://doi.org/10.1016/j.tate.2016.01.021

Majeed A, Khan S, Hwang SO (2022) A comprehensive analysis of privacy-preserving solutions developed for online social networks. Electronics. https://doi.org/10.3390/electronics11131931

Medková J (2020) High-degree noise addition method for the k-degree anonymization algorithm. In: 2020 Joint 11th International conference on soft computing and intelligent systems and 21st international symposium on advanced intelligent systems. IEEE, Hachijo Island, Japan, pp 1–6. https://doi.org/10.1109/scisisis50064.2020.9322670

Myneni S, Lewis B, Singh T et al (2020) Diabetes self-management in the age of social media: large-scale analysis of peer interactions using semiautomated methods. JMIR Med Inf 8(6):25. https://doi.org/10.2196/18441

Narayanan A, Shmatikov V (2009) De-anonymizing social networks. In: 2009 30th IEEE symposium on security and privacy. IEEE, Oakland, CA, USA, pp 173–187. https://doi.org/10.1109/SP.2009.22

Rajabzadeh S, Shahsafi P, Khoramnejadi M (2020) A graph modification approach for k-anonymity in social networks using the

genetic algorithm. Soc Netw Anal Min 10(1):1–17. https://doi.org/10.1007/s13278-020-00655-6

Rossi RA, Ahmed NK (2015) The network data repository with interactive graph analytics and visualization. In: AAAI. http://networkrepository.com

Samarati P, Sweeney L (1998) Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. In: Technical report SRI-CSL-98-04. Computer Science Laboratory, SRI International, Palo Alto, CA

Shakeel S, Anjum A, Asheralieva A et al (2021) *k*-NDDP: an efficient anonymization model for social network data release. Electronics. https://doi.org/10.3390/electronics10192440

Sihag VK (2012) A clustering approach for structural k-anonymity in social networks using genetic algorithm. In: Proceedings of the CUBE international information technology conference. ACM Press, Pune, India, pp 701–706. https://doi.org/10.1145/2381716.2381850

Srivatsan S, Maheswari N (2022) Privacy preservation in social network data using evolutionary model. Mater Today Proc 62:4732–4737. https://doi.org/10.1016/j.matpr.2022.03.251

Tang K, Han M, Gu Q et al (2019) ShareSafe: an improved version of SecGraph. KSII Trans Internet Inf Syst 13(11):5731–5754. https://doi.org/10.3837/tiis.2019.11.025

Thompson B, Yao D (2009) The union-split algorithm and cluster-based anonymization of social networks. In: Proceedings of the 4th international symposium on information, computer, and communications security. ACM Press, New York, NY, USA, pp 218–227. https://doi.org/10.1145/1533057.1533088

Wu W, Xiao Y, Wang W et al (2010) K-symmetry model for identity anonymization in social networks. In: Proceedings of the 13th international conference on extending database technology. ACM Press, Lausanne, Switzerland, pp 111–122. https://doi.org/10.1145/1739041.1739058

Yang J, Leskovec J (2015) Defining and evaluating network communities based on ground-truth. Knowl Inf Syst 42(1):181–213. https://doi.org/10.1007/s10115-013-0693-z

Yartseva L, Grossglauser M (2013) On the performance of percolation graph matching. In: Proceedings of the first ACM conference on Online social networks. ACM Press, New York, NY, USA, pp 119–130. https://doi.org/10.1145/2512938.2512952

Yazdanjue N, Fathian M, Amiri B (2020) Evolutionary algorithms for k-anonymity in social networks based on clustering approach. Comput J 63(7):1039–1062. https://doi.org/10.1093/comjnl/bxz069

Zhou B, Pei J (2011) The k-anonymity and l-diversity approaches for privacy preservation in social networks against neighborhood attacks. Knowl Inf Syst 28(1):47–77. https://doi.org/10.1007/s10115-010-0311-2

Zhou B, Pei J (2008) Preserving privacy in social networks against neighborhood attacks. In: 2008 IEEE 24th international conference on data engineering. IEEE, Cancun, Mexico, pp 506–515. https://doi.org/10.1109/icde.2008.4497459

Zou L, Chen L, Özsu MT (2009) K-automorphism: a general framework for privacy preserving network publication. Proc VLDB Endow 2(1):946–957. https://doi.org/10.14778/1687627.1687734