Efficient Edge Anonymization of Large Social Graphs

Lijie Zhang and Weining Zhang
Department of Computer Science, University of Texas at San Antonio
{lijez, wzhang}@cs.utsa.edu
April 29, 2011

Abstract

Edges in a social graph may represent private information that needs to be protected. Due to their graph partition schemes, existing edge anonymization methods have several drawbacks, such as high utility loss and high computational overhead. In this paper, we present a new edge anonymization method, which partitions a graph using a new vertex equivalence relation called the neighbor-set equivalence. We show that whenever the resulting graph partition satisfies a graph confidence requirement, as the privacy measure, so does the graph partition based on automorphism. Our algorithm maintains the graph partition explicitly as a partition map and performs edge addition/removal in groups as merges of vertex classes on the partition map. To improve performance, our algorithm constructs a plan heuristically for a sequence of vertex class merges and executes the plan using a number of strategies. It checks for graph confidence only after the execution of the entire merge plan. We evaluate our method through extensive experiments using two real-world large social graphs and several graph utility measures. Our results show that the techniques used by our method effectively improve graph utility and reduce computational overhead.

1 Introduction

A social graph is a graph in which vertexes represent people and edges represent some relationships among people, such as friendship, collaboration, communication, and trust. Social graphs have been used in many areas of study, for example, anthropology [16], biology [3], communication [7], and criminology [6], to model and to analyze data obtained from many different sources, including clinical trials, telecommunication networks, on-line social networks, and retail stores.

Thanks to the advance of computing technology in hardware, software, networking, wireless communication, databases, and web technology, there has been a tremendous growth of applications in e-commerce, e-health, and social computing, resulting in an unprecedented amount of data about people and their social activities. There is an ever growing interest from research communities to study this data in various forms of social graphs. On the other hand, there is also real incentives for organization to release social graphs to researchers or to the public. However, since social graphs may contain sensitive personal information, privacy becomes a major concern for publishing of social graphs.
One approach to protect privacy is to publish anonymized social graphs [17, 21, 5, 19, 10, 22, 1, 11, 24, 20]. A naive anonymization is to simply remove personal identities from the graph. However, as shown in [5, 10, 22, 20], this does not provide sufficient protection because the adversary may still be able to infer sensitive information with some background knowledge about a target person. Such background knowledge may define graph features, such as the degree [10], neighborhood [22] and some other complex structure [5, 20, 24], of the vertex that represents the target person.

Existing graph anonymization methods can be broadly classified into vertex anonymization [5, 10, 22, 24] and edge anonymization [17, 21, 19, 20, 11, 1] methods. Vertex anonymization methods are designed to prevent so called vertex re-identification attack by which the adversary uniquely identifies the vertex representing a target person. Methods such as vertex $k$-anonymity [10] and $k$-automorphism [24] use various graph features to partition vertexes into groups and guarantee that each group contains at least $k$ vertexes.

On the other hand, edge anonymization methods are designed to protect sensitive relationships represented by edges. It has been recognized that vertex anonymization methods do not sufficiently protect sensitive edges. For example, although each vertex in a $k$-automorphism graph is hidden in a group of at least $k$ vertexes, the adversary can still infer with high confidence that two target persons, hiding in two groups, have a sensitive relationship, if every vertex in one group has a sensitive edge with every vertex in the other group. Therefore, edge anonymization deserves more research.

As described in Section 1.2, some existing edge anonymization methods, such as [17, 19, 11], perform graph partition and others do not. Since the adversary typically has some background knowledge to partition the graph, the attack will be focus on inferring sensitive edges among a small number of groups of vertexes. Without taking such graph partitions into consideration, these attacks can not be successfully prevented.

For edge anonymization methods [1, 20, 21] that do partition the graph, there are several ways to create the graph partition and to present the anonymized graph. For example, a graph may be partitioned by an equivalence relation based on some graph feature [20] or by some other non-equivalence based criteria [21, 1]. Also, the anonymized graph may keep individual vertexes [1, 20] or represent groups of vertexes as supernodes [21, 1]. Since non-equivalence based graph partition and the use of supernodes may cause significant loss of utility, it seems more desirable to partition graph by equivalence of graph features and to keep individual vertexes without aggregation.

One important issue of (equivalence-based) graph partition is the choice a graph feature (to define the equivalence relation). Unlike table partition performed by data publishing methods [15, 13] where the fixed set of attributes of a data table is divided into sensitive attribute (SA) and quasi-identifier (QI), and the table is partitioned on values in QI, there is no direct correspondence between graph features and QI in a graph. First, it is possible to define many different graph features for vertexes. Second, different graph features model different types and levels of strength of adversary background knowledge, and are not completely independent. For example, the neighborhood of a vertex is somewhat dependent on its degree and its neighbors degrees.

Without an explicit notion of a QI feature, plausible choices of a graph feature is either a data publisher specified feature or the strongest feature, such as automorphism [24]. However, it is difficult for a data publisher to determine an appropriate graph feature because there may be many potential and unknown adversary with different type of background knowledge. Thus, it is desirable to partition graph using a strong graph feature. Although it has been used for vertex
anonymization, automorphism has not been used for edge anonymization.

Another important issue of graph anonymization methods is the computational cost. Existing methods anonymize graphs by adding (counterfeit) edges [19], removing (original sensitive) edges [20], or doing both [10]. For a given graph feature and privacy requirement, the anonymization may involve a complex computation to create a graph partition and may have to add/remove many edges. The strategy taken by many existing methods will typically add/remove one edge at a time and follow it immediately by a check to see if the privacy requirement is satisfied. Since the check of privacy typically requires an update of the graph partition, this edge-at-a-time strategy can involve a high computational overhead.

1.1 Our Contributions

In this paper, we consider edge anonymity on undirected unlabeled social graph. We use graph confidence [20] as the privacy measure. Given a graph confidence requirement $\tau$, we want to produce an anonymized graph in which the probability for an edge to be re-identified is no more than $1 - \tau$. Unlike [20], which assumes all edges are sensitive, we assume that not all edges are sensitive. The assumption is justified since we observe that in on-line social networks, not all members consider their relationships with other members sensitive.

Since computing graph partition based on automorphism is expensive, we use a new vertex equivalence relation called the neighbor-set equivalence to partition a graph. This equivalence relation is stronger than automorphism, but is much easier to compute and as shown by our experiments, does not cause much more utility loss. To further reduce the overhead, our algorithms takes a set-at-a-time strategy, which checks the privacy once after adding/removing a group of edges. Specifically, we make following contributes in this paper.

1. We define a neighbor-set equivalence relation for graph partition based on a structural equivalence concept in sociology [12]. We show that whenever the resulting graph partition satisfies a graph confidence requirement, so does the graph partition based on automorphism.

2. We represent the neighbor-based graph partition explicitly as a partition map, which has a condensed graph and two mappings that map vertexes and edges between the original graph and the condensed graph. This makes it possible to succinctly define changes to the graph partition resulted from groups of edge addition/removal.

3. We define the merge of two or more equivalence classes of vertexes as a basic operation and present algorithms to perform two special types of vertex class merge on the partition map. These two special types of vertex class merges perform the set-at-a-time edge addition and edge removal, respectively.

4. We show that the number of edges added or deleted by the two vertex class merges is actually the earth-move-distance between the degree distributions of the original and the modified graphs, thus, providing a basis for using heuristics in our algorithms to minimize/reduce the number of edges changed.

5. We present a heuristic edge anonymity algorithm, which constructs a plan of a series of vertex class merges using the number of edges changed as a utility measure, and executes
the merge plan using several strategies. The algorithm checks the graph confidence only after the completion of the entire merge plan. By doing so, it can significantly reduce the computational overhead. The algorithm can also be easily adapted to protect the vertex identity[24, 5]. We also formally prove the correctness of the algorithm.

6. We empirically study our edge anonymization algorithms through extensive experiments using two large social graphs obtained from real-world social networks and using degree distribution, earth-move-distance, and clustering coefficient as graph utility measures. Our results show that the techniques used by our method effectively improve graph utility and reduce computational overhead.

1.2 Related Work

We discuss a number of previous studies that are related to our work in this paper. Interested readers may refer to [9, 23] for more detailed survey on privacy-preserving social graph publishing.

There are a number of methods for protecting vertex identity, which vary in the types of graph features used to partition vertexes.

Liu and Terzi [10] considered the problem of vertex identity disclosure of a simple undirected unlabeled graph. Their method creates a \( k \)-degree anonymous graph in which each degree in the degree sequence is held by at least \( k \) vertexes. The method first constructs a \( k \)-anonymous degree sequence that minimizes the number of edges to be added, and then constructs a graph from the \( k \)-anonymous degree sequence using a heuristic algorithm.

Zhou and Pei [22] studied a neighborhood attack on simple undirected graphs with vertex labels. They define the privacy to be the identification of vertexes based on isomorphism of their neighborhoods and define utility in terms of aggregate network queries. Their method guarantees that each vertex in the anonymized graph is isomorphic to at least \( k - 1 \) other vertexes, by generalizing vertex labels and adding edges. To measure the utility loss that is caused by making the neighborhoods of two vertexes isomorphic, they take into consideration the amount of generalization of vertex labels, the number of added edges, and the number of vertexes newly included into neighborhoods. Their algorithm heuristically processes vertex in degree descending order and greedily partitions vertexes into groups of \( k \) to minimize the utility loss.

Hay et al. [5] considered simple undirected unlabeled graph. Their definition of privacy is the re-identification of a known individual in a naively anonymized graph. They formalize special sets of graph features that can be used to partition a graph as a class of vertex queries, of increasing power, over the graph. These graph features are less powerful than automorphism and are limited by user specified parameters. Their method generates a vertex \( k \)-anonymous supergraph by partitioning the graph and replacing vertex groups by supernodes and edge groups by superedges. The graph partition method is a simulated annealing that modifies the graph partition trying to maximize a special likelihood, which measures how well the anonymized graph fits the original graph. This graph partition is not based on any specific graph feature.

Zou et al. [24] also considered the vertex re-identification problem on simple undirected unlabeled graph. They use the number of changed (added and removed) edges as a measure of utility loss. Their \( K \)-Match (KM) algorithm creates a \( k \)-automorphic graph from an original graph by partitioning the graph into blocks and clustering blocks into groups of \( k \). For each group, the algorithm aligns all blocks in the group and replaces individual blocks by the alignment block. The
graph partition and block clustering is by finding frequent sub-graph patterns [8] that minimizes the utility loss. The block alignment is NP-hard and is done using a heuristic algorithm, which traverses all blocks in the cluster in a synchronized breach-first traversal, introducing new vertexes and edges if needed.

Campan and Truta [2] also considered the vertex re-identification attack on a simple undirected graph in which vertexes are labeled by tuples with QI and SA attributes. Their greedy algorithm partitions vertexes into groups by repeatedly selecting $k$ vertexes that minimize the utility loss to form the next group. The utility loss is measured by the amount of generalization to the vertex labels and a special distance that measures the similarity between neighborhoods of two vertexes.

There are also a number of methods for protecting edge anonymity. These methods differ on several aspects, including the types of graphs, type of information considered private, and the approaches to anonymization.

Ying and Wu [19] proposed a randomization method to prevent edge re-identification attack on a simple undirected unlabeled graph. Their method represents graph by adjacency matrix and measures privacy in terms of the posterior probability of edges. Their algorithms randomly add/delete or switch edges while trying to keep the largest eigenvalue of the adjacency matrix and the second largest eigenvalue of the graph’s Laplacian matrix unchanged. Their methods obtain an anonymized graph by performing random edge perturbations. However, the anonymized graphs obtained by these methods may not provide sufficient protection to edge anonymity, due to the small random noise.

Singh and Zhan [17] considered topological anonymity in simple undirected unlabeled graph. They measure topological anonymity of a graph in terms of degree sets and binary clustering coefficients. They applied the anonymity measure on random graphs and on a real-world social graph, but did not give any anonymization method.

Liu et al. [11] considered simple undirected graph in which edges have weights and proposed a method that perturbing the edge weights while trying to preserve the shortest paths between pairs of vertexes without adding or deleting vertex or edge. Their method modifies the weights of each edge by adding or subtracting a random value depending on whether the edge is visited by a set of user specified shortest paths.

Zhang and Zhang. [20] studied edge anonymity in a simple undirected unlabeled graph in which all edges are considered sensitive. They define privacy to be the existence of a sensitive relationship between target individuals. They measure privacy by a notion of graph confidence and partition the graph by vertex degree. The algorithm creates vertex groups by removing or swapping edges, one at a time, and heuristically minimizes utility loss.

Bhagat et al. [1] considered an interaction graph, a bipartite graph with one type of labeled vertexes representing persons and another type of labeled vertexes representing types of interaction. Edges between the two types of vertexes are undirected and unlabeled. They define privacy to be the true labels of persons in an interaction. They propose two methods, both are based on a graph partition that heuristically adds a vertex into a class as long as it shares no common neighbor with other vertexes in the same class. Their methods either label each vertex by its class or represent each class as a supernode.

Zheleva et al. [21] considered a simple undirected unlabeled graph in which there are sensitive and non-sensitive edges, and a link re-identification attack with which the adversary infers sensitive edges from non-sensitive ones. Their methods include removing all sensitive edges, and aggregating vertexes/edges into clustered nodes/edges.
1.3 Roadmap

The rest of the paper is organized as follows. In Section 2, we define the notion of graph confidence based on studying the sensitive edge detection attack. In Section 3, we define the neighbor-set equivalence and partition map. Section 4 defines special types of vertex classes merges as basic operations, and present their algorithms. Section 5 presents an edge anonymization algorithm that constructs and executes a merge plans. We present an experimental study in Section 6. The detailed proofs of major theorems are given in Appendix A.

2 Basic Concepts

In this paper, a social graph is a simple undirected unlabeled graph. We denote the set of vertexes of a graph $G$ by $V(G)$ and the set of edges by $E(G)$. We assume that in the original graph, some edges are sensitive and other edges non-sensitive. An edge may be sensitive because it represents a special relationship or because it is designated by the data owner. Whenever the context is clear, we refer vertexes as persons and vice verse.

2.1 A Measure of Edge Anonymity

We model a sensitive edge detection attack as an edge query of which the answer may be used to infer an sensitive edge between two target persons.

**Definition 2.1. (Edge Query)** An edge query describes some structural characteristics of two target vertexes in a graph $G$. The answer to the query on the graph consists of two groups $V_1$ and $V_2$ of vertexes in $V(G)$ that satisfy the structural characteristics of the query and all edges in $E(G)$ that connect $V_1$ and $V_2$.

We assume that the structural information in an edge query defines an equivalence relation over vertexes of the graph, therefore, induces a partition of the graph.

**Definition 2.2. (Graph Partition)** Let $R$ be an equivalence relation over vertexes of a graph $G$. The partition of $G$ based on $R$ (or $R$-partition) is $P(G, R)$ in which any two vertexes $v$ and $u$ belong to the same vertex equivalence class (or VEC) iff $vRu$. For any VECs $C_i, C_j \in P(G, R)$, any edge connecting a vertex in $C_i$ with a vertex in $C_j$ belongs to the same edge equivalence class (or EEC), to which $C_i$ and $C_j$ are the end VECs.

Thus, the answer to an edge query consists of two VECs, one for each target person, and one EEC containing all edges between the two VECs. We measure the probability of disclosing a sensitive edge as follows.

**Definition 2.3. (Linking Probability)** Let two sets of vertexes in the answer of an edge query be $C_i$ and $C_j$. Let $X$ and $Y$ be random variables so that $X = 1$ iff the target pair has an edge in the original graph and $Y = 1$ iff the edge in the original graph is sensitive. The probability that an $s$-edge between the target pair is disclosed is given as follows.
\[
P[C_i, C_j] = P[X = 1, Y = 1] = P[Y = 1|X = 1] \cdot P[X = 1] = \frac{\alpha_{ij} \cdot \delta_{ij}}{\delta_{ij} \cdot \beta_{ij}} \cdot \frac{\delta_{ij}}{\beta_{ij}}\] (1)

where \(\alpha_{ij}\) is the number of s-edges, \(\gamma_{ij}\) the number of original edges, \(\delta_{ij}\) the number of edges, and

\[
\beta_{ij} = \begin{cases} 
\frac{|C_i| \times |(C_i|-1)}{2}, & i = j; \\
|C_i| \times |C_j|, & i < j. 
\end{cases} \tag{3}
\]

the number of pairs of vertexes between \(C_i\) and \(C_j\), assuming each pair of vertexes between \(C_i\) and \(C_j\) is equally likely to have an edge, each edge is equally likely to be in the original graph, and each edge in the original graph is equally likely to be sensitive.

Using the linking probability, the privacy of a graph can be measured in terms of graph confidence.

**Definition 2.4. (Graph Confidence)** Let \(G\) be a graph and \(R\) be a given structural equivalence relation. The confidence of edge anonymity in \(G\) under \(R\) is

\[
\text{conf}(G) = 1 - \max\{P[C_i, C_j]|C_i, C_j \in P(G, R)\}. \]

Graph \(G\) is said to be \(\tau\)-confident for some \(0 \leq \tau \leq 1\), if \(\text{conf}(G) \geq \tau\).

**Remark.** With the structural information, the adversary only knows the VECs that contain the target persons, but does not know which vertexes are the target persons. Without additional information, the adversary can at best determine with a probability \(\frac{1}{\beta_{ij}}\) that the pair of target persons is one of the \(\beta_{ij}\) pairs of vertexes. Thus, the probability that there is an edge in the original graph between the target persons is \(P[X = 1] = \frac{\delta_{ij}}{\beta_{ij}}\). But even if there is an edge, the privacy is not breached if the edge is non-sensitive. Eq. 1 further incorporates the probability for an edge to be sensitive, i.e., \(P[Y = 1|X = 1] = \frac{\alpha_{ij}}{\delta_{ij}}\). Since the answer to the edge query is obtained from an anonymized graph, Eq. 2 also indicates with \(\gamma_{ij}\) that the anonymized graph may contain counterfeit edges and only original edges (representing real relationships) can be sensitive.

**Remark.** The linking probability defined here generalizes the similar concept in [20] in two aspects. First, the partition of graph can now be based on any structural equivalence relation, not limited to vertex degree equivalence. Second, the original graph is allowed to have both sensitive and non-sensitive edges, instead of all edges must be sensitive. It is straightforward to see that the linking probability in [20] is the special case of Eq. 1 in which all edges are sensitive (i.e., \(\alpha_{ij} = \delta_{ij}\)).

### 2.2 Utility of Anonymized Graphs

Many graph properties [18] can be used to measure utility of an anonymized graph. Some of these measures, such as graph spectral properties and aggregate network queries quality, are very
inefficient to compute, therefore, are not suitable for the use in anonymization algorithms. In this paper, we consider two utility measures, clustering coefficients and degree distribution. Both these measures evaluate the graph utility globally, that is, by going through the entire graph.

The clustering coefficient \([4]\) of a graph \(G\) is

\[
cc(G) = \frac{3 N \Delta(G)}{N^3(G)},
\]

where \(N \Delta(G)\) is the number of triangles in the graph and \(N^3(G)\) is the number of triples in the graph.

The change of degree distributions can be measured by the earth move distance (EMD). Let \(H = \{(b_1, p_1), \ldots, (b_n, p_n)\}\) and \(H' = \{(c_1, q_1), \ldots, (c_m, q_m)\}\) be vertex degree histograms of the input and the output graphs, respectively, of an NEC merge, where \(b_i\) (resp. \(c_j\)) is a degree (referred to as a bin) and \(p_i\) (resp. \(q_j\)) is the count of vertexes with degree \(b_i\) (resp. \(c_j\)). The difference between the two degree distributions is typically measured by the earth move distance (EMD), which was originally defined in the context of a linear programming problem, called the transportation problem[14]. In this problem, the change from \(H\) to \(H'\) is viewed as moving earth weight from bins \(b_i\) into bins \(c_j\). Let \(f_{ij}\) be a flow (i.e., a part of the count viewed as earth weight) moved from bin \(b_i\) in histogram \(H\) to bin \(c_j\) in histogram \(H'\), and each flow is associated with a cost \(d_{ij}\). The optimal flow should minimize the total cost defined by

\[
\sum_{i=1}^{n} \sum_{j=1}^{m} f_{ij} \cdot d_{ij} \quad (4)
\]

subject to the following constraints:

\[
\begin{align*}
    f_{ij} & \geq 0 & \forall i \in [1, n], j \in [1, m] \\
    \sum_{j=1}^{m} f_{ij} & \leq p_i & \forall i \in [1, n] \\
    \sum_{i=1}^{n} f_{ij} & \leq q_j & \forall j \in [1, m] \\
    \sum_{i=1}^{n} \sum_{j=1}^{m} f_{ij} & = \min\{\sum_{i=1}^{n} p_i, \sum_{j=1}^{m} q_j\} \\
\end{align*}
\]

Given the optimal flow, the EMD is the average cost per unit flow, i.e.,

\[
\frac{\sum_{i=1}^{n} \sum_{j=1}^{m} f_{ij} \cdot d_{ij}}{\sum_{i=1}^{n} \sum_{j=1}^{m} f_{ij}}.
\]

2.3 Problem Definition

We now can define our edge anonymity problem as follows.

Given a social graph \(G\), an equivalence relation \(R\), a threshold \(\tau\), and a utility measure. Find a graph \(G'\) such that, \(G'\) is \(\tau\)-confident based on its \(R\)-partition and minimizes the utility loss.

This problem is \(NP\)-hard. In this paper, we present heuristic algorithms to solve a specific version of this problem, where the graph partition is based on a new equivalence relation defined in Section 3.1, and study empirically the performance and utility of our method.

3 Neighbor-Set Equivalence and N-Partition Map

In this section, we define a new vertex equivalence relation, called the neighbor-set equivalence, which induces a new type of graph partition, N-partition. We show that this partition is stronger than the partition induced by the automorphism. For efficient graph anonymization, we represent the N-partition explicitly using a partition map.
3.1 Neighbor-Set Equivalence

As mentioned in the Section 1, different graph features can induce different graph partitions and some are stronger than others. We now formalize the strength of a graph partition.

**Definition 3.1. (Stronger Partition)** Let $P_1$ and $P_2$ be two distinct partitions of the same graph. We say that $P_1$ is stronger than $P_2$ if every VEC of $P_1$ is contained in a VEC of $P_2$.

In previous studies, several vertex equivalence relations have been considered, including equivalence based on vertex degree, neighborhood of various radius, neighborhood of arbitrary shape, and distance to hubs (i.e., vertexes with very high degrees). In terms of graph confidence, the strongest vertex equivalence relation used in graph anonymization is the automorphism equivalence.

**Definition 3.2. (Automorphism Equivalence)** An automorphism of a graph $G$ is a bijection $\sigma : V(G) \rightarrow V(G)$, such that $(u, v) \in E(G)$ iff $(\sigma(u), \sigma(v)) \in E(G)$. Two vertexes $u$ and $v$ of $G$ are automorphism equivalent, written as $u \sim_A v$, iff there exists an automorphism mapping $\sigma$ such that $u = \sigma(v)$ or $v = \sigma(u)$.

The automorphism equivalence induces an automorphic partition (or simply A-partition), denoted by $P(G, \sim_A)$, in which a pair of vertexes $u$ and $v$ belongs to the same automorphism equivalence class (or AEC) iff $u \sim_A v$. That is, vertexes in an AEC must be indistinguishable based on the complete structural information.

One problem of the A-partition is that it requires to compute all automorphic mappings, which is prohibitively expensive on large graphs. To avoid the computational complexity of an A-partition, we consider the following new vertex equivalence relation.

**Definition 3.3. (N-Equivalence)** Two vertexes $u$ and $v$ of a graph $G$ are neighbor-set equivalent (or simply N-equivalent), written as $u \sim_N v$, iff $NS(u) - \{v\} = NS(v) - \{u\}$, where $NS(u) = \{v \mid \exists v \in V(G), (u, v) \in E(G)\}$ is the set of neighbors of vertex $u$ in $G$.

The N-equivalence induces an N-partition, denoted by $P(G, \sim_N)$, in which a pairs of vertexes $u$ and $v$ belongs to the same neighbor-set equivalence class (or NEC) iff $u \sim_N v$.

**Example 3.1.** We will use Figure 1 as a running example in this paper. Figure 1a shows a simple graph $G$, which has a single sensitive edge $(v_1, v_5)$, labeled by “s”. Its N-partition contains NECs

![Figure 1: A Running Example](image-url)
\{v_1, v_2\}, \{v_3\}, \{v_4\}, \{v_5\}, \{v_6\}, and EECs \{(v_1, v_5), (v_2, v_5)\}, \{(v_3, v_5)\}, \{(v_3, v_6)\}, and \{(v_4, v_6)\}. The linking probabilities are \(P(\{v_1, v_2\}, \{v_3\}) = 0.5, P(\{v_3\}, \{v_5\}) = 0, P(\{v_3\}, \{v_6\}) = 0,\) and \(P(\{v_4\}, \{v_6\}) = 0.\) Thus, the graph confidence of \(G\) is 0.5.

Unlike the A-partition, the N-partition can be computed in \(O(|V|^2|E|)\) because in the worst case, every vertex needs to compare the neighbor set with all other vertexes and the complexity of neighbor set comparison is \(O(|E|)\). On the other hand, making a graph \(\tau\)-confident based on the N-partition is sufficient for making the same graph \(\tau\)-confident based on the A-partition.

**Theorem 3.1.** If a graph is \(\tau\)-confident based on the N-partition, it is also \(\tau\)-confident based on the A-partition.

The proof is given in Appendix A.1.

### 3.2 N-Partition Map

In order to measure the graph confidence during anonymization, it is necessary to compute the N-partition. It will be more efficient to keep track of the changes than re-computing the entire N-partition. For this, we need an explicit representation of the N-partition.

**Definition 3.4. (N-map)** The N-partition map (N-map for short) of a graph \(G\) is a triple \((H, \phi, \psi)\), where \(H\) is another graph, \(\phi : V(G) \rightarrow V(H)\) and \(\psi : V(H) \rightarrow 2^{V(G)}\) are two mappings such that the following properties hold.

1. **Semantics of \(\phi\).** For any \(v, v' \in V(G)\) s.t. \(v \neq v'\), \(\phi(v) = \phi(v')\) iff \(v \sim_N v'\).

2. **Semantics of \(\psi\).** For each \(u \in V(H)\), \(\psi(u) \neq \emptyset\). For any \(u, u' \in V(H)\) s.t. \(u \neq u'\), \(\psi(u) \cap \psi(u') = \emptyset\), \(\cup_{u \in V(H)} \psi(u) = V(G)\). And for any \(v \in V(G)\), \(v \in \psi(u)\) iff \(\phi(v) = u\).

3. **Edge Correspondence.** For each edge \((u, u') \in E(H)\), there is an edge \((v, v') \in E(G)\) for every pair of vertexes \(v \in \psi(u)\) and \(v' \in \psi(u')\) s.t. \(v \neq v'\); for each edge \((v, v') \in E(G)\), there is an edge \((\phi(v), \phi(v')) \in E(H)\).

4. **Label Consistency.** Each vertex \(u \in V(H)\) has a label \(l_u\) that has the value \(|\psi(u)|\), and each edge \((u, u') \in E(H)\) has a label \(l_{(u, u')}\) that has the value \(|S(u, u')|\), where \(S(u, u')\) is the set of sensitive edges between \(\psi(u)\) and \(\psi(u')\).

In the rest of this paper, whenever the \(\phi\) and \(\psi\) mappings are understood, we simply refer \(H\) as the N-map. Notice that \(H\) may have some edge that connects a vertex to itself. To ease the presentation, we extend the notations of the two mappings to sets of vertexes, namely, for sets of vertexes \(S \subseteq V(G)\) and \(S' \subseteq V(H)\), \(\phi(S) = \cup_{w \in S} \{\phi(w)\}\) and \(\psi(S') = \cup_{w \in S'} \psi(w)\). According to edge correspondence, if two vertexes \(v \in \psi(u)\) and \(v' \in \psi(u')\) are neighbors in \(G\), vertexes \(\phi(v) = u\) and \(\phi(v') = u'\) will be neighbors in \(H\) and vice versa. So we use \(NS(\phi(v))\) to denote all neighbors of \(\phi(v)\) in the N-map.

Given a graph, it is straightforward to obtain its N-map. From the N-partition, we can create a unique vertex for each NEC, and add an edge between two vertexes in the map if there is an edge between a pair of vertexes in the NECs represented by the two vertexes in the N-map. The mapping \(\phi\) will then map each vertex in the graph to its NEC vertex in the map and the mapping \(\psi\) will map each NEC vertex in the map to the set of vertexes in the NEC in the graph. Obviously, it is also straightforward to obtain the underlying graph from a given N-map.
Example 3.2. The N-map of the graph in Figure 1a consists of the graph $H$ in Figure 1b. The number inside each circle is the vertex label and the number on the edge is the edge label. The mapping $\phi$ is \{ $\phi(v_1) = \phi(v_2) = u_1$, $\phi(v_3) = u_2$, $\phi(v_4) = u_3$, $\phi(v_5) = u_4$, $\phi(v_6) = u_5$ \}, and the mapping $\psi$ is \{ $\psi(u_1) = \{v_1, v_2\}$, $\psi(u_2) = \{v_3\}$, $\psi(u_3) = \{v_4\}$, $\psi(u_4) = \{v_5\}$, $\psi(u_5) = \{v_6\}$ \}.

The following definition extends the linking probability from being associated to an EEC in the graph to being associated to an edge in the N-map, so that, one can easily compute the confidence of a graph from its N-map.

Definition 3.5. (unsatisfied degree) Let $(H, \phi, \psi)$ be the N-map of a graph $G$, $e = (u, u')$ be an edge $(H, \phi, \psi)$, and $\tau$ be a threshold of the graph confidence. The linking probability of $e$ is $Pr(e) = Pr(\psi(u), \psi(u'))$, and based on Definitions 2.3 and 3.4,

$$Pr(e) = Pr(\psi(u), \psi(u')) = \begin{cases} \frac{0.5 \times l_e}{l_e 	imes (l_e - 1)}, & \text{if } u = u'; \\ \frac{l_e}{l_u \times l_{u'}}, & \text{otherwise}. \end{cases}$$

Edge $e$ is unsatisfied if $Pr(e) > 1 - \tau$. The unsatisfied degree of a vertex $u$ of the N-map, denoted by $ud(u)$, is the number of unsatisfied edges incident on (or covered by) the vertex. The ratio of unsatisfied degree of $u$ is $\frac{ud(u)}{d(u)}$, where $d(u)$ is the degree of $u$.

Example 3.3. Assume that the threshold of graph confidence is 0.7. Then, edge $(u_1, u_4)$ in Figure 1b is unsatisfied, since $Pr((u_1, u_4)) = 0.5 > 1 - 0.7 = 0.3$. The unsatisfied degree of $u_4$ is 1, and the ratio of unsatisfied degree of $u_4$ is 0.5. Similarly, the ratio of unsatisfied degree of $u_1$ is 1 and of $u_2$ ($u_3$ or $u_5$) is 0.

4 The Merge of Vertex Classes

For efficient graph anonymization, we want to add/remove edges in groups. In term of a graph with an N-partition, a group of edge additions/removals corresponds to a merge of NECs or equivalently, a merge of vertexes in the N-map. In this section, we consider two specific types of NEC merge operations and present merge algorithms that modify the N-map to reflect changes on the underlying graph. Since the N-map is smaller than the underlying graph, these algorithms allow NEC merges to be performed efficiently.

4.1 Types of Merges

We consider a merge of a set of NECs on a graph as an operation that adds/removes edges so that vertexes in different NECs in the set prior to the merge end up in the same NEC after the merge.

Definition 4.1. (NEC Merge) Let $(H, \phi, \psi)$ be the N-map of a graph $G$ and $M$ be a set of two or more vertexes in $V(H)$. An NEC-merge on $G$ wrt $M$ (or NEC-merge in short) is a set of edge addition or removal operations that result in a graph $G'$, such that

1. $V(G) = V(G')$;
2. for each pair of vertexes \(v, v' \in V(G')\), such that \(v \not\in \psi(M)\) and \(v' \not\in \psi(M)\), \((v, v') \in E(G')\) iff \((v, v') \in E(G)\);

3. there exists a set of vertexes \(W \subseteq V(G')\) such that for each vertex \(v \in \psi(M)\), \(NS(v) = W - \{v\}\) in \(G'\).

Intuitively, a merge of two or more NECs causes all vertexes in those NECs to have the equivalent neighbor-sets, determined in term of the set \(W\). Depending on the choices of \(W\), we have the following two special types of NEC merges.

**Definition 4.2. (Merge-by-Union)** Let \((H, \phi, \psi)\) be the N-map of a graph \(G\) and \(M \subset V(H)\) be a non-empty set of vertexes. A merge-by-union (of \(G\) wrt \(M\)), denoted by \(mbu(G|M)\), is an NEC-merge that adds a set of edges to \(G\) so that in the resulting graph, also denoted by \(mbu(G|M)\),

\[
W = \begin{cases} 
\psi(\cup_{u \in M} NS(u)), & \text{if } \cup_{u \in M} NS(u) \cap M = \emptyset; \\
\psi(\cup_{u \in M} NS(u) \cup M), & \text{otherwise}.
\end{cases}
\]

Notice that to determine the new neighbor-set (roughly, \(W\)) of vertexes that are merged into the same NEC, we have to consider two cases depending on whether some vertex being merged is also in the new neighbor-set.

**Example 4.1.** Figures 1c and 1d show the graph and its N-map after \(mbu(G|\{u_1, u_2\})\) is performed on the input graph \(G\) in Figure 1a, where vertexes \(u_1\) and \(u_2\) are in Figure 1b. This merge-by-union will add two new edges to \(G\) and result in graph \(G'\) in Figure 1c. Notice that in \(G'\), vertexes \(v_1, v_2,\) and \(v_3\) have the equivalent neighbor-set, and this conforms to the intuitive meaning of “merging NEC \(\{v_1, v_2\}\) (corresponding to \(u_1\)) with NEC \(\{v_3\}\) (corresponding to \(u_2\))”. Also notice that the set \(W\) in this merge is \(W = \{v_5, v_6\}\). Figure 1d shows the graph \(H\) of the N-map of graph \(G'\). The new mapping \(\phi\) is \(\phi(v_1) = \phi(v_2) = \phi(v_3) = u_6, \phi(v_4) = u_3, \phi(v_5) = u_4, \phi(v_6) = u_5\). and the new mapping \(\psi\) is \(\psi(u_6) = \{v_1, v_2, v_3\}, \psi(u_3) = \{v_4\}, \psi(u_4) = \{v_5\}, \psi(u_5) = \{v_6\}\).

**Definition 4.3. (Merge-by-Intersection)** Let \((H, \phi, \psi)\) be the N-map of a graph \(G\) and \(M \subset V(H)\) be a non-empty set of vertexes. A merge-by-intersection, denoted by \(mbi(G|M)\), is an NEC-merge that removes a set of edges to \(G\) so that in the resulting graph, also denoted by \(mbi(G|M)\),

\[
W = \begin{cases} 
\psi(\cap_{u \in M} NS(u)), & \text{if } M \subseteq \cap_{u \in M} NS(u); \\
\psi(\cap_{u \in M} NS(u) - M), & \text{otherwise}.
\end{cases}
\]

This is similar to merge-by-union except that the new neighbor-set (roughly, \(W\)) only contains the neighbors shared by vertexes in \(\psi(M)\), thus is the intersection of their neighbor-sets. Again, we need to consider two cases.

It is worth noting that the merge-by-union has the largest new neighbor-set and will only add edges; and the merge-by-intersection has the smallest new neighbor-set and will only remove edges.
Input: N-map \((H, \phi, \psi)\) of graph \(G\), and a non-empty set \(M \subset V(H)\)

Output: \((H', \phi', \psi')\) that is the N-map of \(mbu(G|M)\)

1. compute \(\phi(W)\) of \(mbu(G|M)\);
2. for each \(u \in M\) and each \(u' \in \phi(W)\) s.t. \((u, u') \notin E(H)\) do
   if \((u \neq u'\) or \(|\psi(u)| > 1\))
      add \((u, u')\) to \(E(H)\) with label 0;
3. \(S_2 = (\phi(W) - \cap_{u \in M} NS(u)) \cup M;\)
4. while there is some vertex \(u\) in \(S_2\) do
5. \(U = \{u' | u' \in V(H) \land (\exists v \in \psi(u), v' \in \psi(u')(\psi(NS(u)) - \{v, v\}') = \psi(NS(u')) - \{v, v\})\};\)
6. if \(|U| > 1\) then
7. add a new vertex \(w\) into \(V(H)\) with label \(l_w = \sum_{u \in U} l_u;\)
8. if \(\cup_{s \in U} NS(s) \cap U \neq \phi\)
   add edge \((w, w)\) into \(E(H)\) with label
   \(l_{(w, w)} = \sum_{u, w \in U \land (u, w') \in E(H)} l_{(u, u')};\)
9. for each \(u' \in \cup_{s \in U} NS(s) - U\) do
   add edge \((w, u')\) into \(E(H)\) with label
   \(l_{(w, u')} = \sum_{u \in U \land (u, u') \in E(H)} l_{(u, u')};\)
10. set \(\psi(w) = \psi(U)\) and \(\phi(v) = w\) for each \(v \in \psi(w);\)
11. for each \(u \in U\) do
   remove \(\psi(u), u,\) and any edge incidents on \(u;\)
12. \(S_2 = S_2 - U;\)
13. return \((H, \phi, \psi);\)

Algorithm 1: Merge-by-union
4.2 Algorithms of Class Merges

As mentioned before, we not only need to merge NECs to anonymize a graph, but we also need to maintain the N-partition for the purpose of computing the graph confidence. Thus, we need to maintain the N-map and the associated mappings as the graph changes. One approach is to re-build the N-map from the underlying graph after each NEC merge. This will require us to maintain both the underlying graph and the N-map. Since an NEC merge does not remove vertex but may add/remove many edges, this approach is inefficient, especially for merge-by-union due to increasing size of the underlying graph.

An alternative approach is to work directly on the N-map. Basically, for each NEC merge, we only make necessary modifications to the N-map and its mappings. After all required NEC merges, the anonymized graph can be easily produced from the N-map. Since the N-map becomes smaller and smaller as more and more NEC merges are performed, this approach will be much more efficient.

Algorithm 1 performs a merge-by-union on the N-map. Before describing the algorithm, let us describe the changes expected to the N-map. Keep in mind that neighbors in the underlying graph are one-one correspondent to neighbors in the N-map. As mentioned before, the merge-by-union will only add new edges into the underlying graph. In the N-map, these new edges will be added between vertexes in $M$ (the merged vertexes) and vertexes in $\phi(W)$ (the new neighbors). Due to possible self-loops in the N-map, $M$ and $\phi(W)$ may share some vertexes, as expressed in Definition 4.2.

The set $\phi(W)$ may contain two subsets, $S_1$ and $S_2$, where each vertex in $S_1$ is a neighbor to all vertexes in $M$ before the merge and each vertex in $S_2$ is not a neighbor to some vertex in $M$ before the merge. Only the vertexes in $S_2$ will have new edges added by the NEC merge. Therefore, for each vertex in $S_2$, we must identify vertexes that share the same neighbor-set. Once these vertexes are found, they are replaced by a new vertex, and the affected mappings must be updated. The merge also add new edges to the vertexes in $M$. So we handle the vertexes in $M$ in the same way as those in $S_2$.

Now consider Algorithm 1. Step 1 identifies the new neighbors of merged vertexes, namely $\phi(W)$, in the N-map according to the definition of merge-by-union. Step 2 adds new edges to the N-map between vertexes in $M$ and vertexes in $\phi(W)$ so that vertexes in $M$ will all have the same neighbor-sets. Step 3 identifies vertexes in $S_2$ mentioned earlier, except it also contains vertexes in $M$. This is because the same merge operations will be performed on these vertexes. Step 5 takes a vertex in $S_2$ and identifies a set of vertexes, $U$, in the N-map that are candidates for merging with a vertex, $u$, from $S_2$. The set $U$ always contains the vertex $u$ from $S_2$. In particular, if the vertex $u$ is in $M$, then $U \supseteq M$. Step 6 determine if a merge needs to be performed on vertexes in $U$. To perform a merge on vertexes in $U$, Step 7 adds a new vertex $w$ into the N-map. Step 8 adds a self-loop edge to $w$ if vertexes in $U$ are neighbors to each other. Step 9 adds edges between $w$ and other neighbors of vertexes in $U$. Since each of these new edges will replace multiple existing edges, its label is the sum of labels of edges it will replace. Step 10 updates the $\phi$ and $\psi$ mappings for $w$. Step 11 cleans up the replaced mappings and removes the replaced vertexes and edges. Step 12 removes all vertexes in $U$ from $S_2$. Step 13 returns final N-map.

Example 4.2. Figure 1d shows the graph of the new N-map after using Algorithm 1 to perform $mbu(G\{u_1, u_2\})$ on the N-map in Figure 1b. The new mappings are exactly those given in Example 4.1. The new vertex $u_6$ is created to replace vertexes $u_1$ and $u_2$ in the resulted N-map $H'$. Notice
that \( u_6 \) is labeled with 3, the sum of the labels of \( u_1 \) and \( u_2 \). As a result, the linking probability on edge \((u_6, u_3)\) becomes \( Pr((u_6, u_4)) = \frac{1}{3} \).

The following result establishes the correctness of Algorithm 1. The proof is given in Appendix A.2.

**Theorem 4.1.** Algorithm 1 is correct.

**Remark.** The algorithm for merge-by-intersection is similar to Algorithm 1, with a small change to Steps 1 and 2. Namely, in Step 1, the \( W \) is changed to the one given in Definition 4.3, and replace the Step 2 with “for each \( u \in M \) and \( w \notin \phi(W) \), remove edge \((u, w)\) from \( E(H)\)

### 4.3 Utility Loss of a Class Merge

By altering the input graph, an NEC merge causes structural changes of the original graph, and therefore reduces the utility. To evaluate the quality of graph anonymization algorithms, we need to measure the utility of the anonymized graph relative to the utility of the original graph. On the other hand, in searching for a better anonymized graph, we need to choose NEC merges based on their estimated utility losses in the anonymization algorithms.

As mentioned in Section 2.2, we measure utility by the clustering coefficient and the EMD between degree distributions. Thus, we measure the loss of utility by the difference of clustering coefficients and the EMD between the original and the anonymized graphs.

For a merge-by-union, since it does not add/remove vertex, the number of triples in the graph does not change, that is, \( N_3(G) = N_3(G') = \frac{|V(G)|!}{6((|V(G)|-3)!} \), where \( G' \) is the result of \( mbu(G|M) \). In this case, the difference of clustering coefficients can be estimated by

\[
\Delta cc(G, G') = \frac{3|N_\Delta(G) - N_\Delta(G')|}{6(|V(G)|!)}. 
\]

The EMD is difficult to compute in general. However, as indicated by the following theorem, in some special cases, it can be computed very efficiently. Let \( H = \{(b_1, p_1), \ldots, (b_n, p_n)\} \) and \( H' = \{(c_1, q_1), \ldots, (c_m, q_m)\} \) be vertex degree histograms of the input and the output graphs, respectively, of an NEC merge. Recall that \( b_i \) (resp. \( c_j \)) is a degree (referred to as a bin) and \( p_i \) (resp. \( q_j \)) is the count of vertexes with degree \( b_i \) (resp. \( c_j \)).

**Theorem 4.2.** Let \( G \) and \( G' \) be two graphs such that \( V(G) = V(G') \) and \( E(G) \subseteq E(G') \). If the cost of changing the degree of a vertex from \( b_i \) to \( c_j \) is \( |b_i - c_j| \), then the EMD between the histograms of vertex degrees of \( G \) and \( G' \)

\[
EMD = \frac{2 \times |E(G') - E(G)|}{|V(G)|}. 
\]  

The proof of the theorem is given in Appendix A.3.

**Corollary 4.3.** If \( G' \) is the result of \( mbu(G, M) \) (or \( mbi(G, M) \)) and the cost of changing the degree of a vertex from \( b_i \) in \( G \) to \( c_j \) in \( G' \) is \( |b_i - c_j| \), then the EMD between degree histograms of \( G \) and \( G' \) is \( EMD = \frac{2 \times |E(G') - E(G)|}{|V(G)|} \).

**Proof.** If \( G' \) is the result of \( mbu(G, M) \), by Definition 4.2, \( V(G) = V(G') \) and \( E(G) \subseteq E(G') \). If \( G' \) is the result of \( mbi(G, M) \), by Definition 4.3, \( V(G) = V(G') \) and \( E(G) \supseteq E(G') \). In both cases, the corollary follows Theorem 4.2.
The importance of Corollary 4.3 is that if only merge-by-union (respectively, merge-by-intersection) is used to anonymize a graph, the algorithm can use the number of edges added (resp. removed) by the operation to compute the EMD and therefore to measure the loss of utility.

5 Graph Anonymization via Merge Plans

Recall that an unsatisfied edge in an N-map represents an EEC in the underlying graph whose linking probability does not satisfy a given graph confidence requirement. The purpose of an NEC merge is to reduce the number of unsatisfied edges. It does this by combining EECs or changing the composition of EECs. As a result, an NEC merge may improve privacy by lowering the linking probabilities of some unsatisfied edges. On the other hand, since an NEC changes the original graph, it may cause loss of utility. For a given graph, we want to find a sequence of NEC merges that can transform the graph to satisfy the given confidence requirement and yet best preserve the utility. In this section, an NEC merge is implicitly represented by its merge set.

Definition 5.1. (merge plan) Given the N-map \((H, \phi, \psi)\) of a graph \(G\), a merge plan is a sequence of vertex set \(M = (M_0, \ldots, M_{n-1})\) such that

- \(\forall i \in [0, n - 1], M_i \subseteq V(H)\) and \(|M_i| > 1\);
- \(\forall i, j \in [0, n - 1]\) and \(i \neq j\), \(M_i \cap M_j = \phi\);
- \(\bigcup_{i=0}^{n-1} M_i\) covers all unsatisfied edges in \(E(H)\).

Intuitively, a merge plan represents a sequence of NEC merges

\[
(m(G_0|M_0), \ldots, m(G_i|M_i), \ldots, m(G_{n-1}|M_{n-1}))
\]

where \(G_0 = G\) and for \(1 \leq i \leq n\), \(G_i\) is the result of NEC merge \(m(G_{i-1}|M_{i-1})\), which can be either a merge-by-union or a merge-by-intersection. It is worth noting that each merge set \(M_i\) contains only vertexes of the N-map of \(G_0\). This is because we want to obtain a static plan before executing any NEC merge.

Input: the graph \(G\), the confidence threshold \(\tau\), plan execution type \(t\)

Output: a \(\tau\)-confident graph

1. \((H, \phi, \psi) = \text{the N-map of graph } G\);
2. compute linking probabilities of edges in \((H, \phi, \psi)\);
3. while \((G\text{ is not } \tau\text{-confident})\) do
4. \(M = \text{MergePlanConstruction}((H, \phi, \psi), \tau)\);
5. if \((M=\bot)\) return null;
6. \((H, \phi, \psi) = \text{MergePlanExecution}(M, t, (H, \phi, \psi))\);
7. \(G' = \text{the graph corresponding to } (H, \phi, \psi)\);
8. return \(G'\);

Algorithm 2: Merge Plan Construction and eXecution (MPCoX)
Algorithm 2 creates an anonymized graph by repeatedly generating and executing merge plans. In Step 1, the N-partition of $G$ used to construct its N-map. In Step 2, the linking probabilities of edges of the N-map are computed. Step 3-6 iteratively build a merge plan using Algorithm 3, and execute the merge plan using Algorithm 4. The anonymized graph is constructed from the modified N-map in Step 7, and is returned in Step 8.

### 5.1 Merge Plan Construction

Given a large social graph, there can be a large number of possible merge plans. Ideally, we want to find a merge plan that can achieve a given graph confidence and has a minimum utility loss. But, this problem is conjectured to be $NP$-hard. Thus, we present a heuristic algorithm for constructing a good merge plan.

**Input:** the N-map $(H, \phi, \psi)$ of graph $G$, the graph confidence threshold $\tau$

**Output:** a merge plan $M$

1. $M = \bot$
2. $S = \{v | v \in V(H) \land ud(v) > 0\}$
3. while ($|S| \geq 1$ and at least one vertex are not in $S$ or $M$) do
4.   if ($|S| \geq 2$)
5.     $M =$ two vertexes from $S$ with the largest ratio of unsatisfied degree;
6.   else
7.     $M = S \cup \{a$ vertex not in $S$ or $M\}$;
8.     $S = S - M$;
9.     $M = concat(M, M)$;
10. if ($M \neq \bot$) then union $S$ with the last set of $M$;
11. return $M$;

**Algorithm 3: Merge Plan Construction**

Algorithm 3 constructs a merge plan using a greedy approach. Since it is very difficult, if not impossible, to predict if the execution of a given merge plan will result in a graph that satisfies a given $\tau$-confidence requirement, this algorithm only guarantees that the constructed merge plan covers all unsatisfied edges in the N-map.

At Step 1 of the algorithm, the merge plan $M$ is initialized. The Step 2 finds all vertexes of the N-map of which unsatisfied degree is greater than 0. Notice that for each unsatisfied edge $(u, u')$, both $u$ and $u'$ are included in $S$. The algorithm goes through Step 3 to Step 9 iteratively to build merge sets one at a time. With an exception of the last merge set, each merge set will contain precisely two vertexes that have the maximum ratio of unsatisfied degree. The last merge set may contain at most three vertexes. This heuristic aims to avoid the bias towards vertexes with large degree and as discussed in Section 4.3, to minimize the utility loss. The merge plan is completed when all unsatisfied edges are covered. Notice that the algorithm may return an empty merge plan if the input N-map has no unsatisfied edge. Also, the merge plan does not guarantee that the resulting graph will satisfy the given graph confidence.
5.2 Merge Plan Execution

We consider three strategies to execute a given merge plan, depending on the type of NEC merge to be perform for each merge set in the merge plan. Strategy U performs merge-by-union for every merge set, Strategy I performs merge-by-intersection, and Strategy H performs either a merge-by-union or a merge-by-intersection, depending on which of the two will changes less number of edges (according to an estimation). If both types of NEC merges change the same number of edges, an additional strategy is used to break the tie. Specifically, H-a breaks the tie by performing the merge-by-union, H-d by merge-by-intersection, and H-r by randomly choosing between the two types of NEC merges. Consequently, these result in five execution types: U, I, H-a, H-d, and H-r.

Input: a merge plan \( M \), execution type \( t \), an N-map \( (H, \phi, \psi) \)

Output: an N-map \( (H, \phi, \psi) \)

1. for each merged set \( M_i \) in \( M \) do
2. if \( M_i \subseteq V(H) \) and \( M_i \) covers at least one unsatisfied edge
3. if \( (t \text{ is } U) \)
4. \( (H, \phi, \psi) = \text{mbu}(G|M_i); \)
5. else if \( (t \text{ is } I) \)
6. \( (H, \phi, \psi) = \text{mbi}(G|M_i); \)
7. else /* \( t \text{ is } H-a, H-d, \text{ or } H-r */
8. \( n_u = \text{the number of edges to be added by } \text{mbu}(G|M_i); \)
9. \( n_i = \text{the number of edges to be deleted by } \text{mbi}(G|M_i); \)
10. if \( n_i < n_u \) then \( (H, \phi, \psi) = \text{mbi}(G|M_i); \)
11. else if \( n_i > n_u \) \( (H, \phi, \psi) = \text{mbu}(G|M_u); \)
12. else \( (H, \phi, \psi) = \text{mbi}(G|M_i) \text{ or } \text{mbu}(G|M_u) \) according to \( t \);
13. return \( (H, \phi, \psi); \)

Algorithm 4: Merge Plan Execution

Algorithm 4 executes a given merge plan according to a specific execution type. In Steps 1-12, it goes through the merge plan one merge set at a time. For each merge set, it verifies at Step 2 that the vertexes in the merge set are still in the (updated) N-map and they still cover at least one unsatisfied edge. If not, this merge set can be skipped. Otherwise, an appropriate merge operation is determined and performed in Steps 3-12. For type H-x (where x = a,d,or r) execution, the number of edges to be added/removed by each of the two types of merge operations is estimated at Steps 8 and 9. The justification of using these edge counts as a measure of utility loss is given in Section 4.3. The algorithm returns a new N-map at Step 13.

6 Experimental Results

We performed extensive experiments to study the performance of our graph anonymization algorithms. Due to the lack of common basis for comparison, we were unable to find existing method to compare with our algorithms. Therefore, we focus on properties and implementation alternatives of our algorithms. In particular, we empirically validate the following features of our algorithms.
1. Use NEC merge as a basic operation.
2. Perform operations on N-map rather than on the underlying graph.
3. Construct a merge plan according to heuristics.
4. Execute a merge plan with one of the five execution types.
5. Check the graph confidence after the execution of the entire merge plan.

To study these features in detail, we implemented the MPCoX algorithm (Algorithm 2). In this section, the five execution types of MPCoX are denoted as MPCoX-U, MPCoX-I, MPCoX-H-a, MPCoX-H-d, and MPCoX-H-r.

We used two data sets in the experiments. The EPINION data set was crawled from the Epinion.com in 2009. The graph contains 89,672 vertexes and 462,923 edges with vertexes representing members and edges representing trust of members on other members. For the purpose of our study, we treated the directed edges as undirected edges. The FB data set was crawled from the Facebook.com in 2010. The graph has 44,745 vertexes and 506,250 edges with vertexes representing members and edges representing friendships among members. The experiments were performed on a PC with 2.13GHz Pentium Core 2 processor and 2GB memory.

6.1 Using NEC Merge as Basic Operation

The MPCoX algorithm uses NEC merge as the basic operation. Notice that although our algorithm performs NEC merges on N-maps, an NEC merge can nevertheless be performed directly on the underlying graph. In this experiment, we investigate the benefit of using NEC merge as the basic operation on the underlying graph.

We compare using NEC merge as the basic operation with using single-edge addition/removal as the basic operation. To do this, we modified the GADED-Rand algorithm of [20] to measure the graph confidence based on an N-partition. This algorithm works directly on the underlying graph and performs edge deletion one edge at a time. After each edge deletion, it updates the graph partition and recompute the graph confidence.

We also implemented an algorithm MonG (Merge on Graph), which performs merge-by-intersection directly on the underlying graph and checks the graph confidence immediately after each NEC merge. Notice that this algorithm does not construct a merge plan, but it chooses the next merge set using the same heuristics used in Algorithm 3.

We run GADED-Rand and MonG on EPINION and FB data sets with various settings of graph confidence requirements and percentages of sensitive edges. Our results show that GADED-Rand runs much slower than MonG. For example, for graph confidence threshold $\tau = 0.7$ and with percentage of sensitive edges ranging from 10% to 90% in increment of 10%, GADED-Rand runs on an average of 1000 times slower than MonG. These results clearly show that using NEC merge as the basic operation is superior to using single-edge addition/deletion as the basic operation, even without N-map and merge plans.
6.2 Benefit of N-Map

The MPCoX algorithms performs NEC merges on the N-map rather than on the underlying graph. In this experiment, we investigate the performance gain of using the N-map.

For the comparison, we implemented a modified version of MPCoX, called MPCoX-g, which is identical to MPCoX except it performs NEC merges directly on the underlying graph. Therefore, it stores the underlying graph in the memory and updates the graph as the merge plan is executed.

We run both MPCoX-g and MPCoX with execution type U on the FB and EPINION graphs with the percentage of sensitive edges varying from 0% to 90% with increments of 10% and with various graph confidence thresholds.

As shown in Figure 2, for any given threshold of graph confidence and percentage of sensitive edges, MPCoX always returns an N-map that is smaller than the initial N-map of the input graph. In fact, during the execution of the MPCoX, the size of the N-map (in terms of the number of vertexes and the number of edges) decreased monotonically. On the other hand, although the number of vertexes in the graph maintained by MPCoX-g remains the same, the number of edges in the graph increased rapidly, causing the size of the graph to grow out of the available memory. As a result, MPCoX-g failed to complete in many of our test cases. For example, in all test cases shown in Figure 2, MPCoX-g crashed with the out-of-memory error. These results indicate that using the N-map in our algorithm effectively improved the ability for the algorithm to handle large graphs with limited memory resource.

For completeness, we also note that the difference of MPCoX-g and MPCoX will be much less with execution types I and H-d. The results are omitted here.

6.3 Effect of Our Merge Plan Construction Method

In this experiment, we study the quality of merge plans constructed by Algorithm 3, which is based on several heuristics.

For comparison, we implemented a new algorithm, called REA, which is identical to MPCoX except that it constructs the merge plan by randomly selecting two vertexes from the set $S$ to form each merge set.
We run both algorithms with execution type U on FB and EPINION data sets with various settings of the graph confidence threshold and percentage of sensitive edges. Figure 3 shows some typical results from our experiments. As shown by the results, MPCoX outperforms REA on the number of edges added, which indicates a better utility of the anonymized graph. REA runs faster than MPCoX on FB data set, but slower on EPINION data set. Hence, the time performance of the two algorithms seems to be data-dependent.

Similar results were also obtained on other execution types.

### 6.4 Comparison of Merge Plan Execution Strategies

In this experiment, we study the five merge plan execution strategies in terms of the utility loss. We used three measures of the utility: the actual EMD, the clustering coefficient and the vertex degree histogram. To measure the utility loss, we run MPCoX with the five execution types on FB and EPINION data sets with various settings of the graph confidence threshold and the percentage of sensitive edges.

Figure 4 shows some results based on EMD. As indicated by the results, MPCoX-U has the worst utility on both data sets. On EPINION data set, MPCoX-H-r performs the best on average and MPCoX-I is very close to MPCoX-H-r. On FB data set, MPCoX-I performs the best and MPCoX-H-r is the second best. MPCoX-H-a performs similar to MPCoX-U and MPC-H-d performs similar to MPCoX-I on both data sets. This is because most of the time, the two types of NEC merges change the same number of edges.

Figure 5 shows the actual vertex degree histograms of two data sets. For readability, we only plotted the interpolation of the degree histogram. In Figures 5a and 5b, we can see that the degree distribution produced by MPCoX-H-r is the closest to that of the original graphs, indicating the best utility in terms of the degree distribution. This is consistent with its performance on EMD. In Figure 5, the performances of MPCoX-U and MPCoX-H-a are similar to MPCoX-H-r for low threshold of graph confidence, but as the threshold of graph confidence increases, their performances become worse than MPCoX-H-r. On the other hand, MPCoX-I and MPCoX-H-d have the worst performance on degree histograms, which is somewhat surprising considering their performances on EMD. We think the reason is that since many edges are removed by MPCoX-I
(a) Result with EPINION. The x-axis is the percentage of sensitive edges. The threshold of graph confidence is $\tau = 0.7$.

(b) Result of EPINION. The x-axis is the threshold of graph confidence. The percentage of sensitive edges is $s = 50\%$.

(c) Result with FB. The x-axis is the percentage of sensitive edges. The threshold of graph confidence is $\tau = 0.7$.

(d) Result with FB. The x-axis is the threshold of graph confidence. The percentage of sensitive edges is $s = 50\%$.

Figure 4: The Actual Earth Move Distance. The y-axis of all four subfigures is the EMD between the degree distributions of the original and the anonymized graphs. In the legend, Union, Inter, Hybrid_a, Hybrid_d and Hybrid_r correspond to MPCoX-U algorithm, MPCoX-I algorithm, MPCoX-H-a algorithm, MPC-Hybrid-d algorithm and MPCoX-H-r algorithm, respectively. A greater EMD indicates a lower utility.
(a) Results with EPINION. The threshold of graph confidence is $\tau = 0.3$.

(b) Results with EPINION. The threshold of graph confidence is $\tau = 0.8$.

(c) Results with FB. The threshold of graph confidence is $\tau = 0.3$.

(d) Results with FB. The threshold of graph confidence is $\tau = 0.8$. Degree over 500 was cut off.

Figure 5: The Degree Distributions. The X-axis is the vertex degree. For readability, we ignored the degrees over 300. The Y-axis is the number of vertexes of a given vertex degree. The dark thickest solid line is the degree distribution of the original data set. The gray line is the histogram of the anonymized graph created by MPCoX-H-r. The lines of MPCoX-U and MPCoX-H-a are almost overlapped and the lines of MPCoX-I and MPCoX-H-d are very close.
and MPCoX-H-d, the degrees of most vertexes become zero in the anonymized graph, resulting in fast drops in the figures near the low degree range. Thus, although the EMDs between the degree histograms of the anonymized graphs and the original graph may not be very big, the changes to the degree histograms are nevertheless significant. Based on these results, for applications that are sensitive to vertex degrees, MPCoX-H-r will be a better choice.

Figure 6 shows the change of clustering coefficient, as defined in Section 4.3. The results show that for all five execution types, the relative change to the clustering coefficient is very small, indicating that this utility measure is not sensitive to the execution types of the merge plan.

### 6.5 Checking Graph Confidence after Executing a Merge Plan

In this experiment, we empirically study the advantages of withholding the check of graph confidence until after the execution of a merge plan. We compare MPCoX with a modified version, MPCoX-np, which checks the graph confidence immediately after each NEC merge. MPCoX-np uses the same heuristics as used by Algorithm 3 to determine the merge sets for each merge set, and same execution strategy as used by MPCoX to execute the NEC merges. However, this modified algorithm will execute the NEC merge immediately followed by checking the graph confidence.

We run MPCoX and MPCoX-np with different execution types on both FB and EPINION data set under various settings of the graph confidence threshold and the percentage of sensitive edges.

Figure 7 shows results of our experiments using execution type U. As indicated by the result, MPCoX-np is several times slower than MPCoX. As the graph confidence increase, the MPCoX-
Figure 7: Performance of MPCoX-U-np vs MPCoX-U on EPINION and FB Data Sets. The X-axis is the threshold of graph confidence. The percentages of sensitive edges were 5% and 10%.

np needs to add more edges to obtain the satisfied graph. Hence, we can see that checking graph confidence after the execution of a whole merge plan is more efficient than checking after every single merge. Similar results were also obtained for other execution strategies.

7 Conclusion

In this paper, we present a new edge anonymization method, which partitions a graph using a new vertex equivalence relation called the neighbor-set equivalence. We show that whenever the resulting graph partition satisfies a graph confidence requirement, as the privacy measure, so does the graph partition based on automorphism. Our algorithm maintains the graph partition explicitly as a partition map and performs edge addition/removal in groups as merges of vertex classes on the partition map. To improve performance, our algorithm constructs a plan heuristically for a sequence of vertex class merges and executes the plan using a number of strategies. It checks for graph confidence only after the execution of the entire merge plan. We evaluate our method through extensive experiments using two real-world large social graphs and several graph utility measures. Our results show that the techniques used by our method effectively improve graph utility and reduce computational overhead.

References


A Appendix

A.1 Proof of Theorem 3.1

In the following, we prove that measuring graph confidence on N-partition is sufficient to achieve the graph with the same confidence on A-partition. The proof is based on the following Lemma.

Lemma A.1. The N-partition of any graph is stronger than its A-partition.

Proof. We show that each AEC in \( P(G, \sim_A) \) contains one or more NEC in \( P(G, \sim_N) \), that is, for any \( u, v \in V(G) \), if \( u \sim_N v \), then \( u \sim_A v \). Assume that \( u \sim_N v \), let us consider a specific mapping \( \sigma : V \rightarrow V \) such that \( u = \sigma(v) \), \( v = \sigma(u) \), and \( w = \sigma(w) \) for all \( w \in V(G) \) such that \( w \neq u \neq v \). We show that such a mapping is an automorphism. To prove that \( u \sim_A v \), we need to show that for any edge \((x, y)\) in \( E(G) \), the edge \((\sigma(x), \sigma(y))\) is also in \( E(G) \).

Let \((x, y)\) be an edge in \( E(G) \). If none of \( u \) and \( v \) is in \([x, y]\), we simply have \((\sigma(x), \sigma(y)) = (x, y)\). Now assume without loss of generality that \( x = u \) (therefore \( y \neq u \)). If \( y = v \), then \((\sigma(x), \sigma(y)) = (\sigma(u), \sigma(v)) = (v, u)\). Because of \((x, y) = (u, v) \in E(G)\) and \( G \) is undirected, \((v, u) \in E(G)\). If \( y = w \neq v \), then \( w \) is a neighbor of \( u \) and \((\sigma(x), \sigma(y)) = (\sigma(u), \sigma(w)) = (w, v)\). Since \( u \sim_N v \), \( w \) is also a neighbor of \( v \). That is, \((v, w) \in E(G)\).

Assume the graph confidence based on N-partition is not less than \( \tau \), which according to Definition 2.4 means that for any \( N_i, N_j \in P(G, \sim_N) \), \( Pr[N_i, N_j] = P_{N_{ij}} \leq 1 - \tau \) or equivalently \( \tau \leq 1 - \max\{P_{Nhk}[N_h, N_k \in P(G, \sim_N)]\} \). By Lemma A.1, each AEC \( A_i \in P(G, \sim_A) \) contains one or more NECs. We will show that for each \( A_i, A_j \in P(G, \sim_A) \), \( Pr[A_i, A_j] = P_{A_{ij}} \leq \max\{P_{Nhk}[N_h \subseteq A_i, N_k \subseteq A_j]\} \), therefore,

\[
\tau \leq 1 - \max\{P_{Nhk}[N_h, N_k \in P(G, \sim_N)]\} \leq 1 - \max\{P_{A_{ij}}[A_i, A_j \in P(G, \sim_A)]\}.
\]

We need to consider two cases depending on if \( A_i = A_j \).

Case 1: \( A_i \neq A_j \).

By Eq. 1 and Eq. 3, \( P_{A_{ij}} = \frac{\alpha_{ij}}{|A_i| \cdot |A_j|} \), where \( \alpha_{ij} \) is the number of s-edges between \( A_i \) and \( A_j \). Assume that \( A_i \) contains \( s \) NECs and \( A_j \) contains \( t \). Since \( A_i \cap A_j = \emptyset \), all NECs contained in \( A_i \) and \( A_j \) are distinct, thus \(|A_i| \cdot |A_j| = \sum_{N_h \subseteq A_i, N_k \subseteq A_j} |N_h| \cdot |N_k|\). The \( \alpha_{ij} \) s-edges can be partitioned among the \( s \cdot t \) pairs of NECs between \( A_i \) and \( A_j \), thus \( \alpha_{ij} = \sum_{N_h \subseteq A_i, N_k \subseteq A_j} \alpha_{hk} \), where \( \alpha_{hk} \) is the number of s-edges between NECs \( N_h \) and \( N_k \). We can rewrite \( P_{A_{ij}} \) as follows.

\[
P_{A_{ij}} = \frac{\sum_{N_h \subseteq A_i, N_k \subseteq A_j} \alpha_{hk} |N_h| \cdot |N_k|}{\sum_{N_h \subseteq A_i, N_k \subseteq A_j} |N_h| \cdot |N_k|} = \frac{\sum_{m=1}^{s \cdot t} a_m}{\sum_{m=1}^{s \cdot t} b_m} \leq \max\{\frac{a_m}{b_m} \mid 1 \leq m \leq s \cdot t\}
\]

where each \( m \) corresponding to a unique pair \( N_h \) and \( N_k \), and \( \frac{a_m}{b_m} = P_{Nhk} \). Thus, \( \max\{\frac{a_m}{b_m} \mid 1 \leq m \leq s \cdot t\} = \max\{P_{Nhk}[N_h \in A_i, N_k \in A_j]\} \). The inequality holds because if \( \frac{a_1}{b_1} = \max\{\frac{a_m}{b_m} \mid 1 \leq m \leq s \cdot t\} \), then \( \frac{a_1}{b_1} \geq \frac{a_m}{b_m} \) or \( a_1 b_m \geq b_1 a_m \) holds for every \( 1 \leq m \leq s \cdot t \), and therefore,

\[
\frac{a_1}{b_1} - \frac{\sum_{m=1}^{s \cdot t} a_m}{\sum_{m=1}^{s \cdot t} b_m} = \frac{a_1 \cdot \sum_{m=1}^{s \cdot t} b_m - b_1 \cdot \sum_{m=1}^{s \cdot t} a_m}{b_1 \cdot \sum_{m=1}^{s \cdot t} b_m} = \frac{\sum_{m=1}^{s \cdot t} a_1 b_m - a_1 b_m}{b_1 \cdot \sum_{m=1}^{s \cdot t} b_m} \geq 0.
\]
Thus for the \( N_h \) and \( N_k \) that correspond to \( \frac{\alpha_{ii}}{b_{ii}} \), we have \( P_{A_{ij}} \leq P_{N_{hh}} \leq 1 - \tau \). Since this is true for every \( P_{A_{ij}} \in P(G, \sim_A) \), we conclude that

\[
\max \{ P_{A_{ij}} | A_i, A_j \in P(G, \sim_A) \} \leq \max \{ P_{N_{hh}} | N_h, N_k \in P(G, \sim_N) \} \leq 1 - \tau
\]

and the theorem holds.

Case 2, \( A_i = A_j \).

By Eq. 1 and Eq. 3, \( P_{ij} = P_{A_{ii}} = \frac{\alpha_{ii}}{(|A_i| - (|A_i| - 1))^2} \), where \( \alpha_{ii} \) is the number of s-edges among vertexes in \( A_{ii} \). Again, assume that \( A_i \) contains s NECs. Then,

\[
\frac{|A_i| \cdot (|A_i| - 1)}{2} = \frac{1}{2} \left( \sum_{N_h \subseteq A_i} |N_h| \cdot \left( \sum_{N_k \subseteq A_i} |N_k| \right) - 1 \right)
\]

\[
= \frac{1}{2} \left( \sum_{N_h \subseteq A_i} |N_h| \cdot \left( \sum_{N_k \subseteq A_i} |N_k| - 1 \right) \right)
\]

\[
= \frac{1}{2} \left( \sum_{N_h \subseteq A_i} \left( \sum_{N_k \subseteq A_i} |N_h| \cdot |N_k| - |N_h| \right) \right)
\]

\[
= \frac{1}{2} \left( \sum_{N_h \subseteq A_i} \sum_{N_k \subseteq A_i, h \neq k} |N_h| \cdot |N_k| + \sum_{N_h \subseteq A_i} |N_h| \cdot (|N_h| - 1) \right)
\]

\[
= \sum_{N_h, N_k \subseteq A_i, h \neq k} \frac{|N_h| \cdot |N_k|}{2} + \sum_{N_h \subseteq A_i} \frac{|N_h| \cdot (|N_h| - 1)}{2}
\]

We can partition the \( \alpha_{ii} \) s-edges among pairs of NECs in \( A_i \), so that, \( \alpha_{ii} = \sum_{N_h, N_k \subseteq A_i, h \neq k} \alpha_{hh} + \sum_{N_h \subseteq A_i} \alpha_{hh} \). We can rewrite \( P_{A_{ij}} \) as follows.

\[
P_{A_{ij}} = \frac{\sum_{N_h, N_k \subseteq A_i, h \neq k} \alpha_{hh} + \sum_{N_h \subseteq A_i} \alpha_{hh}}{\sum_{N_h, N_k \subseteq A_i, h \neq k} \frac{|N_h| \cdot |N_k|}{2} + \sum_{N_h \subseteq A_i} \frac{|N_h| \cdot (|N_h| - 1)}{2}}
\]

\[
= \frac{\sum_{s=1}^{s} a_m}{\sum_{s=1}^{s} b_m}
\]

\[
\leq \max \left\{ \frac{a_m}{b_m} | 1 \leq m \leq s \cdot s \right\}
\]

where each \( m \) corresponds to a unique pair of NECs \( N_h \) and \( N_k \), and if \( h = k \), \( \frac{a_m}{b_m} = P_{N_{hh}} \). The inequality holds for the same reason as in case 1. Again, assume that \( \frac{a_1}{b_1} = \max \left\{ \frac{a_m}{b_m} | 1 \leq m \leq s \cdot s \right\} \). If \( m = 1 \) corresponds to \( N_h \) and \( N_k \) where \( h = k \), we have \( P_{A_{ij}} \leq P_{N_{hh}} \leq 1 - \tau \); otherwise (i.e., \( h \neq k \)), we have \( P_{A_{ij}} \leq \frac{1}{2} P_{N_{hh}} \leq P_{N_{hh}} \leq 1 - \tau \), both imply that the theorem holds.
Algorithm 1. The concept of a property holding is defined below.

We now establish the correctness of Algorithm 1 proved in Theorem 4.1, namely, we show the corresponding conditions between $H_i, \phi_i, \psi_i$. Each of the following properties is said to hold at Step $i$ of Algorithm 1, if the corresponding conditions between $(H_i, \phi_i, \psi_i)$ and $G'$ are satisfied.

1. The semantics of $\phi$. $\forall u \in V(H_i), \exists v \in V(G'), \phi_i(v) = u$ and $\forall v' \in V(G')$ s.t. $v' \neq v, \phi_i(v') = u$ iff $v \sim_N v'$.

2. The semantics of $\psi$. $\forall u \in V(H_i)$, (a) $\psi_i(u) \neq \emptyset$, (b) $\forall u' \in V(H_i)$ s.t. $u' \neq u, \psi_i(u) \cap \psi_i(u') = \emptyset$, (c) $\bigcup_{u \in V(H_i)} \psi_i(u) = V(G')$, and (d) $\forall v \in V(G'), v \in \psi_i(u)$ iff $\phi_i(v) = u$.

3. The edge correspondence. (a) $\forall (u, u') \in E(H_i), \forall v \in \psi_i(u), \forall v' \in \psi_i(u')$ s.t. $v \neq v', (v, v') \in E(G')$; and (b) $\forall (v, v') \in E(G'), (\phi_i(v), \phi_i(v')) \in E(H_i)$.

4. The label consistency. $\forall u \in V(H_i), l_u = |\psi_i(u)|$ and $\forall (u, u') \in E(H_i), l_{(u, u')} = |S_i(u, u')|$.

The meaning should also be obvious if we say that one of these properties holds on a vertex or an edge.

### A.2 Proof of Theorem 4.1

We now establish the correctness of Algorithm 1 proved in Theorem 4.1, namely, we show the algorithm indeed returns the N-map of the graph resulted from the merge-by-union. The basic idea is to show that some properties, similar to those in Definition 3.4, holds at various steps of Algorithm 1. The concept of a property holding is defined below.

We consider step-by-step transformation of N-map in Algorithm 1. Let $G' = mbu(G|M)$ and $(H, \phi, \psi)$ be the N-map of $G$. Although at many steps of the transformation, the triple $(H, \phi, \psi)$ may not be the N-map of either the graph $G$ or the graph $G'$, we still refer it as an N-map to keep the description simple. However, we use $(H_i, \phi_i, \psi_i)$ to denote the snapshot of the N-map at Step $i$ of Algorithm 1. Table 1 shows step-by-step changes to the N-map. Similarly, we use $S_i(u, u')$ to denote the set of sensitive edges between $\psi_i(u)$ and $\psi_i(u')$.

**Definition A.1.** Each of the following properties is said to hold at Step $i$ of Algorithm 1, if the corresponding conditions between $(H_i, \phi_i, \psi_i)$ and $G'$ are satisfied.

<table>
<thead>
<tr>
<th>Step $i$</th>
<th>$V(H_i)$</th>
<th>$E(H_i)$</th>
<th>$\phi_i$</th>
<th>$\psi_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$V(H)$</td>
<td>$E(H)$</td>
<td>$\phi$</td>
<td>$\psi$</td>
</tr>
<tr>
<td>2</td>
<td>—</td>
<td>add new edges</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3 - 6</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>7</td>
<td>add $w$</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>8</td>
<td>—</td>
<td>may add $(w, w)$</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>9</td>
<td>—</td>
<td>may add $(w, u')s$</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>10</td>
<td>—</td>
<td>—</td>
<td>reset $\phi(v) = w$ for $v \in \psi_i(U)$</td>
<td>add $\psi(w) = \psi_i(U)$</td>
</tr>
<tr>
<td>11</td>
<td>remove $U$</td>
<td>remove edges</td>
<td>—</td>
<td>remove $\psi_i(u)$ for $u \in U$</td>
</tr>
<tr>
<td>12-13</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 1: Differences of N-map at various Steps: “—” at any step denotes no change at that step.
Lemma A.2. The semantics of \( \psi \), the edge correspondence, and the label consistency hold at Step 3 of Algorithm 1.

**Proof.** We have the following observations at Step 3: A) the input \((H, \phi, \psi)\) is the N-map of \( G \), which satisfies Definition 3.4; B) by Definitions 4.1 and 4.2, \( V(G) = V(G') \) and \( E(G) \subset E(G') \); and C) according to Table 1, \( V(H_3) = V(H) \), \( E(H) \subseteq E(H_3) \), \( \phi_3 = \phi \) and \( \psi_3 = \psi \).

(1) Consider the semantics of \( \psi \). Consider any \( u \in V(H_3) \). Because of the aforementioned observations, (a) \( \psi(u) \neq \emptyset \) implies \( \psi_3(u) \neq \emptyset \); (b) \( \forall u' \in V(H_3) \) s.t. \( u' \neq u \), \( \psi(u) \cap \psi(u') = \emptyset \) implies \( \psi_3(u) \cap \psi_3(u') = \emptyset \); and (c) \( \forall v \in V(G') \), \( v \in \psi(u) \) iff \( \phi(v) = u \) implies \( v \in \psi_3(u) \) iff \( \phi_3(v) = u \).

(2) Consider edge correspondence. Consider any \( (u, u') \in E(H_3) \). Either \( (u, u') \in E(H) \) or \( (u, u') \not\in E(H) \).

If \( (u, u') \in E(H) \), by edge correspondence of Definition 3.4, \( \forall v \in \psi(u) \) and \( \forall v' \in \psi(u') \), \( (v, v') \in E(G) \), thus by aforementioned observations B and C, \( v \in \psi_3(u) \), \( v' \in \psi_3(u') \) and \( (v, v') \in E(G') \).

If \( (u, u') \not\in E(H) \), it must be added at Step 2. Without lose of generality, let \( u \in M \) and \( u' \not\in \phi(W) \) in \( H \). Since \( \phi_3 = \phi \), \( \psi_3 = \psi \) and \( V(G) = V(G') \), for any \( v \in \psi_3(u) \) and \( v' \in \psi_3(u') \) in \( G' \), we must have \( v \in \psi(u) \) and \( v' \in \psi(u') \) in \( G \). Because \( \psi(u) \subseteq \psi(M) \) and \( \psi(u') \subseteq W \), by Definition 4.1, \( (v, v') \) is in \( E(G) \). Since \( E(G) \subset E(G') \), \( (v, v') \) is in \( E(G') \), too.

Consider any edge \( (v, v') \in E(G') \), where \( v \in \psi_3(u) \), \( v' \in \psi_3(u') \) for some \( u, u' \in V(H_3) \). Because \( V(G) = V(G') \), \( v \) and \( v' \) are both in \( V(G) \). The edge \( (v, v') \) may or may not be in \( E(G) \). If \( (v, v') \in E(G) \), by Definition 3.4, \( \phi(v), \phi(v') \) \( \in \psi_3(u) \). By Table 1, \( (\phi_3(v), \phi_3(v')) \in E(H_3) \).

If \( (v, v') \not\in E(G) \), it must be added by \( mbu(G|M) \). By Definition 4.1 and without loss of generality, let \( v \in \psi(M) \) and \( v' \in W \). By Definition 3.4, \( \exists u \in M \) and \( \exists u' \in \phi(W) \), s.t., \( \phi(v) = u \) and \( \phi(v') = u' \). According to the algorithm, \( (u, u') \) is added at Step 2 implying \( (u, u') \in E(H_3) \). Because \( \phi_3 = \phi \), \( \phi_3(v), \phi_3(v') \in E(H_3) \).

(3) Consider label consistency. Consider any \( u \in V(H_3) \). By observation A, \( l_u = |\psi(u)| \). By observations B and C, \( u \in V(H) \) and \( \psi(u) = \psi_3(u) \). Since there is no change to any vertex label in Steps 1 through 3, \( l_u = |\psi_3(u)| \). For each edge \( (u, u') \in E(H) \), \( (u, u') \in E(H_3) \) by observation C. Since \( \psi_3 = \psi \), edge correspondence holds in Steps 1 through 3, and there is no change to edge labels in Steps 1 through 3, \( l_{(u, u')} = |S(u, u')| = |S_3(u, u')| \).

**Lemma A.3.** In Algorithm 1, if the semantics of \( \psi \) holds at Step 4, it also holds at Step 12.

**Proof.** We show that conditions 2.(a) to 2.(d) of Definition A.1 are satisfied at Step 12.

Consider condition 2.(a) and 2.(d). Consider any vertex \( u \in V(H_{12}) \). Either \( u \in V(H_4) \) or \( u \not\in V(H_4) \).

If \( u \in V(H_4) \), according to Table 1, \( u \neq w \) and \( u \not\in U \), where \( w \) is the new vertex added at Step 7 and \( U \) is a set of vertexes removed at Step 11. Therefore, \( \psi_4(u) = \psi_{12}(u) \) and \( \forall v \in \psi_4(u) \), \( \phi_4(v) = \phi_{12}(v) \). Since \( \psi_4(u) \neq \emptyset \) by assumption, thus \( \psi_{12}(u) \neq \emptyset \); also \( \forall v \in V(G') \), since \( \phi_4(v) = u \) iff \( v \in \psi_4(u) \) by assumption, thus \( \phi_{12}(v) = u \) iff \( v \in \psi_12(u) \).

If \( u \not\in V(H_4) \), it follows \( u = w \), as well as \( |U| > 1 \) at Step 6. By the assumption, \( \forall u' \in U \), \( \psi_4(u') \neq \emptyset \), therefore, \( \psi_4(U) \neq \emptyset \). Since \( \psi_{10}(w) = \psi_4(U) \), it follows \( \psi_{10}(w) \neq \emptyset \). Also, \( \forall v \in \psi_{10}(w) \), \( \phi_{10}(v) = w \). Because by Table 1, \( \psi_{12}(w) = \psi_{10}(w) \) and \( \phi_{12} = \phi_{10} \), as a result,
ψ_{12}(w) \neq \emptyset$ and $φ_{12}(v) = w$. Since $∀u ∈ U$, $ψ_{11}(u)$ is removed, as a result, $∀v ∈ ψ_{11}(U)$, $v$ only belongs to $ψ_{12}(w)$. So $φ_{12}(v) = w$ iff $v ∈ ψ_{12}(w)$.

Consider condition 2.(b). Consider any pair of vertexes $u, u' ∈ V(H_{12})$. Since only one new vertex is added between Steps 4 and 12, at most one of $u$ and $u'$ can be the new vertex. If neither is the new vertex, they are in both $V(H_4)$ and $V(H_{12})$, which as argued previously means $ψ_4(u) = ψ_{12}(u)$ and $ψ_4(u') = ψ_{12}(u')$. Since by assumption, $ψ_4(u) ∩ ψ_4(u') = \emptyset$, it follows $ψ_{12}(u) ∩ ψ_{12}(u') = \emptyset$. If one vertex, say $u'$, is new vertex $w$, by Table 1, $ψ_{10}(u') = ψ_4(U)$ and $ψ_{10}(u) = ψ_4(u)$. By assumption, $ψ_4(u) ∩ ψ_4(U) = \emptyset$. So $ψ_{10}(u) ∩ ψ_{10}(w) = \emptyset$. Because $ψ_{12}(w) = ψ_10(w)$ and $ψ_{12}(u) = ψ_{10}(u)$, it follows $ψ_{12}(u) ∩ ψ_{12}(w) = \emptyset$.

Consider condition 2.(c). We show $∪_{u ∈ V(H_{12})}ψ_{12}(u) = V(G')$. Step 12 can be reached from Step 4 via Step 6 or Step 11. If it is via Step 6, $∪_{u ∈ V(H_{12})}ψ_{12}(u) = ∪_{u ∈ V(H_4)}ψ_4(u) = V(G')$ by the assumption. If it is via Step 11, $V(H_{12}) = V(H_4) − U + \{w\}$. By Table 1, $∀u ∈ V(H_{12})$, if $u ∈ V(H_4) − U$, $ψ_{12}(u) = ψ_{10}(u) = ψ_4(u)$; and if $u = w$, $ψ_{12}(u) = ψ_{10}(u) = ψ_4(U)$. So $∪_{u ∈ V(H_{12})}ψ_{12}(u) = ∪_{u ∈ V(H_4) − U + \{w\}}ψ_{12}(u) = ∪_{u ∈ V(H_4) − U}ψ_4(u) ∪ ψ_4(U) = ∪_{u ∈ V(H_4)}ψ_4(U) = V(G')$.

**Lemma A.4.** In Algorithm 1, if the semantics of $ψ$ and the edge correspondence hold at Step 4, then at Step 5, (a) $∀u,u' ∈ U, ∀v ∈ ψ_5(u), ∀v' ∈ ψ_5(u'), (v \sim_N v') in G'$; (b) $∀v,v' ∈ V(G')$, if $(v \sim_N v')$ then either both $φ_5(v)$ and $φ_5(v')$ are in $U$ or both not in $U$.

**Proof.** By assumption, the semantics of $ψ$ and the edge correspondence hold at Step 4. By Table 1, $ψ_4 = ψ_5$ and $E(H_4) = E(H_5)$. Thus the semantics of $ψ$ and the edge correspondence also hold at Step 5. Therefore, for each $v ∈ G'$, $∃u ∈ V(H_5)$ s.t. $v ∈ ψ_5(u)$ and $NS(v) = ψ_5(NS(u)) − \{v\}$. Consequently, $∀u ∈ V(H_5), ∀v,v' ∈ ψ_5(u), v \sim_N v'$ in $G'$.

For (a), consider any pair of $u,u' ∈ U$ at Step 5. By definition of $U$, $∃v ∈ ψ_5(u), \exists v' ∈ ψ_5(u')$ s.t., $ψ_5(NS(u)) − \{v,v'\} = ψ_5(NS(u')) − \{v,v'\}$, or equivalently, $v \sim_N v'$ in $G'$. This is because $NS(v) − \{v,v'\} = ψ_5(NS(u)) − \{v,v'\} = ψ_5(NS(u')) − \{v,v'\} = NS(v') − \{v,v'\}$. Since all vertexes in $ψ_5(u)$ are $N$-equivalent to $v$ and those in $ψ_5(u')$ to $v'$, $v \sim_N v'$ implies that every vertex in $ψ_5(u)$ is $N$-equivalent to every vertex in $ψ_5(u')$.

For (b), consider any pair of $v,v' ∈ V(G')$ s.t. $(v \sim_N v')$. Without loss of generality, let $φ_5(v) = u ∈ U \subseteq V(H_5)$. By assumption of the semantics of $ψ$, $v ∈ ψ_5(u)$. Let $φ_5(v') = u' ∈ V(H_5)$. By the semantics of $ψ$, $v' ∈ ψ_5(u')$. Since $v \sim_N v'$, by the semantics of $ψ$ and edge correspondence, $ψ_5(NS(u)) − \{v,v'\} = NS(v) − \{v,v'\} = NS(v') − \{v,v'\} = ψ_5(NS(u')) − \{v,v'\}$. So by definition of $U$, $φ_5(v') = u' ∈ U$.

**Lemma A.5.** In Algorithm 1, if the semantics of $ψ$ and edge correspondence hold at Step 4, and $∪_{u ∈ U}NS(u) ∩ U \neq \emptyset in H_8$, then $∀v ∈ ψ_8(U)$ in $G'$, $NS(v) \supseteq ψ_8(U) − \{v\}$.

**Proof.** In Algorithm 1, $U$ is computed at Step 5 and stays unchanged through Step 8. Thus $w \notin U$ and $∀v ∈ V(H_4)$, $ψ_4(u) = ψ_1(u)$ for $i = 5, 6, 7, 8$ and $E(H_4) = E(H_1)$ for $i = 5, 6, 7$. Also, $E(H_8) = E(H_4)$ or $E(H_8) = E(H_4) \cup \{(w,w)\}$. As a result, the semantics of $ψ$ holds on every vertex in $V(H_8)$ other than $w$ and the edge correspondence holds on each edge in $E(H_8)$ other than $(w,w)$.

Since $∪_{u ∈ U}NS(u) ∩ U \neq \emptyset in H_8$, there must be at least one edge $(u',u'') ∈ E(H_8)$ (i.e. $u' ∈ NS(u'')$ and $u'' ∈ NS(u'')$) for some (not necessarily distinct) vertexes $u',u'' ∈ U$. Since
w \not\in U$, it follows that $u' \neq w$ and $u'' \neq w$. Thus by edge correspondence, $\exists (s', s'') \in E(G')$ s.t. $s' \in \psi_8(u')$ and $s'' \in \psi_8(u'')$, therefore, $s \in NS(s')$ (and also $s' \in NS(s'')$).

Now, consider some vertex $v \in V(G')$ s.t. $v \in \psi_8(U)$. Since $\psi_8(U) = \psi_5(U) = \cup_{u \in U} \psi_5(u)$, $\exists u \in U$ s.t. $v \in \psi_5(u)$. Since $u, u', u'' \in U$, by Lemma A.4, $v, s'$ and $s''$ are pair-wise N-equivalent in $G'$. Since $s' \sim_N v$ and $s'' \in NS(s')$, if $v \neq s''$, it must follow that $s'' \in NS(v)$ and $v \in NS(s'')$. Thus, $\forall v \in \psi_8(U)$, if $v \neq s''$, $v \in NS(s'')$. In other words, $NS(s'') \supseteq \psi_8(U) - \{s''\}$. Since $s'' \sim_N v$, we also have $NS(v) = NS(s'') \cup \{s''\} - \{v\}$, thus $NS(v) \supseteq \psi_8(U) - \{v\}$. 

Lemma A.6. In Algorithm 1, if the semantics of $\psi$ and edge correspondence hold at Step 4, then at Step 9, $\forall u \in (\cup_{s \in U} NS(s) - U), \forall u' \in U, \forall v \in \psi_8(u), \forall v' \in \psi_8(u'), (v, v') \in E(G')$.

Proof. According to Table 1, the $\psi$ mapping does not change between Steps 4 and 9. Thus the semantics of $\psi$ will hold on vertexes in $V(H_4)$ at Steps 4 through 9. Also, since no edge in $H_4$ is removed between Steps 4 and 9, the edge correspondence will hold at Step 9 on any edge that does not incident on $w$.

Consider any $u \in (\cup_{s \in U} NS(s) - U), \exists u'' \in U$ s.t., $u \in NS(u'')$ (hence $(u, u'') \in E(H_9)$) but $u \not\in U$. By the semantics of $\psi$, $\psi_9(u) \neq \emptyset$ and $\psi_9(u'') \neq \emptyset$. By edge correspondence, $(u, u'') \in E(H_9)$ implies $\forall v \in \psi_9(u), \forall v'' \in \psi_9(u''), (v, v'') \in E(G')$, or equivalently, $\psi_9(u) \subseteq NS(v'')$ and $\psi_9(u'') \subseteq NS(v)$.

Consider any vertex $u' \in U$ at Step 9. Again by the semantics of $\psi$, $\psi_9(u') \neq \emptyset$ in $G'$. Consider any vertex $v' \in \psi_9(u')$. By Lemma A.4, since $u', u'' \in U, \forall v'' \in \psi_5(u''), v'' \sim_N v''$. Because $V(H_9) = V(H_5) \cup \{w\}$ and $w' \neq w, \forall v' \in \psi_9(u'), v' \sim_N v''$. Since $\psi_9(u) \subseteq NS(v'')$ and $v' \sim_N v''$, it follows $\psi_9(u) \subseteq NS(v')$. That is, $\forall v \in \psi_9(u'), (v, v') \in E(G')$.

Lemma A.7. In Algorithm 1, if the semantics of $\psi$ and the edge correspondence hold at Step 4, the edge correspondence will also hold at Step 12.

Proof. Consider any edge $(u, u') \in E(H_{12})$. Either $(u, u') \in E(H_4)$ or $(u, u') \not\in E(H_4)$.

If $(u, u') \in E(H_4)$, none of $u$ and $u'$ can be $w$ (the new vertex added at Step 7) or be removed at Step 12. Thus, $\psi_{12}(u) = \psi_4(u)$ and $\psi_{12}(u') = \psi_4(u')$; and by assumption, $\forall v \in \psi_{12}(u)$ and $v \neq w$. Since $\psi_9(u) \subseteq NS(v'')$ and $v' \sim_N v''$, it follows $\psi_9(u) \subseteq NS(v')$. That is, $\forall v \in \psi_9(u'), (v, v') \in E(G')$.

If $(u, u') \not\in E(H_4)$, the edge $(u, u')$ must be either $(w, w)$, the edge added at Step 8 or $(w, u')$ for some $u' \in \cup_{s \in U} NS(s) - U$, one of the edges added at Step 9. In both cases, $|U| > 1$ at Step 6.

If $(u, u') = (w, w)$, it follows $\cup_{s \in U} NS(s) \cup U \neq \emptyset$. By Lemma A.5, $\forall v \in \psi_8(U), NS(v) \supseteq \psi_8(U) - \{v\}$. Therefore, $\forall v, v' \in \psi_8(U)$ s.t. $v \neq v', (v, v') \in E(G')$. By Table 1, $\psi_8(U) = \psi_4(U) = \psi_{10}(w) = \psi_{12}(w)$ and $\psi_4(U)$ is removed at Step 11. Hence, $\forall v, v' \in \psi_{12}(w)$ s.t. $v \neq v', (v, v') \in E(G')$.

If $(u, u') = (w, u')$, by Lemma A.6, $\forall v \in \psi_9(U)$ and $\forall v' \in \psi_9(u'), (v, v') \in E(G')$. Similarly, $\psi_9(U) = \psi_4(U) = \psi_{10}(w) = \psi_{12}(w)$ and $\psi_4(U)$ is removed at Step 11. Because $u' \not\in U$, $\psi_{12}(u') = \psi_9(u')$. So $\forall v \in \psi_{12}(w), \forall v' \in \psi_{12}(u'), (v, v') \in E(G')$.

Consider any edge $(v, v') \in E(G')$. By Lemma A.3, we can assume that $v \in \psi_4(u)$ and $v' \in \psi_4(u')$ for some $u, u' \in V(H_4)$. By the edge correspondence and the semantics of $\psi$, $(u, u') \in E(H_4)$. Let us consider what happens to $(u, u')$ at Step 12.

If $(u, u') \in E(H_{12})$, then neither of $u$ and $u'$ is involved in Step 8, 9, or 11. By Table 1, $\psi_{12}(u) = \psi_{12}(u')$. Thus, $(v, v') \in E(H_4)$ and $(v, v') \in E(G')$ and by Lemma A.3, $\phi_{12}(v) = u$ and $\phi_{12}(v') = u'$. So $(u, u') = (\phi_{12}(v), \phi_{12}(v')) \in E(H_{12})$. 

33
If \((u, u') \notin E(H_{12})\), at least one of \(u\) and \(u'\) must be removed at Step 11. According to the algorithm, this happens only if at least one of \(u\) and \(u'\) is in \(U\) at Step 5 and \(|U| > 1\). Without loss of generality, assume \(u \in U\) and consider \(u'\). Either \(u' \in U\) or \(u' \notin U\).

If \(u' \in U\), then \((u, u') \in E(H_4)\) implies \(\cup_{v \in U} NS(v) \cap U \neq \phi\) at Step 8, therefore, \((w, w) \in E(H_8)\). Because \((w, w)\) is not removed at Step 11, it follows \((w, w) \in E(H_{12})\). Because \(\phi_{10}(v') = \phi_{10}(v) = w\) and \(\phi_{12} = \phi_{10}\), it follows \((\phi_{12}(v), \phi_{12}(v')) \in E(H_{12})\).

If \(u' \notin U\), then \(u' \in U \cup_{v \in V} NS(v) - U\) at Step 9, therefore, \((w, u') \in E(H_6)\). Since \((w, u')\) is not removed at Step 11, it follows \((w, u') \in E(H_{12})\). Because \(\psi_4(u) \subseteq \psi_4(U)\), \(\phi_{10}(v) = w\), and because \(\psi_4(u') \not\in \psi_4(U)\), \(\phi_{10}(v') = \phi_4(v') = u'\). Because \(\phi_{12} = \phi_{10}\), \((\phi_{12}(v), \phi_{12}(v')) \in E(H_{12})\).

**Lemma A.8.** In Algorithm 1, if at Step 4, the semantics of \(\psi\) and the edge correspondence hold and the semantics of \(\phi\) holds on a vertex \(u \in V(H_4)\), then at Step 12, \(u \in V(H_{12})\) and the semantics of \(\phi\) still holds on \(u\).

**Proof.** We prove by contradiction.

Assume that at Step 4, the semantics of \(\psi\) and the edge correspondence hold and the semantics of \(\phi\) holds on a vertex \(u \in V(H_4)\), but at Step 12, \(u \notin V(H_{12})\). Then, \(u\) must be removed at Step 11. So \(u \in U\) and \(|U| > 2\) at Step 6. Thus \(\exists u' \in U, u' \neq u\). By Lemma A.4, \(\forall v \in \psi_5(u)\) and \(\forall v' \in \psi_5(u')\), \(v \sim_N v'\). Because \(\psi_5 = \psi_4\) and the semantics of \(\psi\) holds at Step 4, \(u' \neq u\) implies \(\phi_4(v) \neq \phi_4(v')\). Thus, \(v \sim_N v'\) but \(\phi_4(v) \neq \phi_4(v')\), which is contrary to the assumption that the semantics of \(\phi\) on \(u\) holds at Step 4.

Thus, \(u \in V(H_4)\) and \(u \in V(H_{12})\). Now assume that the semantics of \(\phi\) on \(u\) does not hold at Step 12. That is, \(\forall v, v' \in V(G')\) s.t. \(v \neq v'\) and \(\phi_4(v) = u\), \(\phi_4(v') = u\) iff \(v \sim_N v'\) in \(G'\), but at Step 12, \(\exists v, v' \in V(G')\) s.t. \(v \neq v'\) and \(\phi_12(v) = u\), either (1) \(\phi_12(v') = u\) but \(v \sim_N v'\) or (2) \(\phi_12(v') \neq u\) but \(v \sim_N v'\). Notice that since \(G'\) is fixed, we only need to consider changes to the \(\phi\)-mapping.

In case (1), since \(v \sim_N v'\) in \(G'\), the change to the \(\phi\)-mapping is from \(\phi_4(v') \neq u\) to \(\phi_12(v') = u\), which can only occur at Step 10. Since \(\phi_{10}(v') = w\) and \(\phi_{12} = \phi_{10}\), it follows \(u = w\), where \(w\) is the new vertex added at Step 7. However, since \(w \notin V(H_4)\) but \(u \in V(H_4)\), it follows \(u \neq w\), a contradiction.

In case (2), since \(v \sim_N v'\) in \(G'\), the change to the \(\phi\)-mapping is from \(\phi_4(v') = u\) to \(\phi_12(v') \neq u\), which again can occur only at Step 10. Thus, \(\phi_12(v') = w\) and \(u \in U\). Since \(\phi_12(v) = u \neq w = \phi_12(v')\), it follows \(u \notin U\), a contradiction.

**Lemma A.9.** In Algorithm 1, if at Step 4, the semantics of \(\psi\) and the edge correspondence hold and if a new vertex \(w\) is added at Step 7, then the semantics of \(\phi\) will hold on \(w\) at Step 12.

**Proof.** We need to show \(\forall v, v' \in V(G')\) s.t. \(v \neq v'\) and \(\phi_{12}(v) = w\), (1) if \(\phi_{12}(v') = w, v \sim_N v'\) in \(G'\) and (b) if \(v \sim_N v'\) in \(G'\), \(\phi_{12}(v') = w\). Notice that since \(w\) is added to \(H\) at Step 7, \(U\) must contain at least two vertices.

(a) If \(\phi_{12}(v') = w\), then \(\phi_{10}(v) = \phi_{10}(v') = w\) by Table 1. So \(v, v' \in V_{10}(w) = \psi_4(U)\). Because \(\psi_4 = \psi_5\) and the semantics of \(\psi\) holds at Step 4, \(\exists u, u' \in U\) at Step 5, s.t. \(v \in \psi_5(u)\) and \(v' \in \psi_5(u')\). By Lemma A.4, \(v \sim_N v'\) in \(G'\).

(b) If \(v \sim_N v'\) in \(G'\), where \(v \neq v'\) and \(\phi_5(v) = w\), then \(v \in \psi_12(w)\) by Lemma A.3. By Table 1, \(\psi_{12}(w) = \psi_{10}(w) = \psi_4(U)\) and \(\phi_5 = \phi_4\). So \(\exists u \in \psi_4(U)\). And \(\phi_4(v) = u = \phi_5(v) \in U\).
by the assumption. Because \( v \sim_N v', \phi_5(v') \in U \) by Lemma A.4. Thus, \( \phi_4(v') \in U \). Therefore, \( v' \in \psi_4(U) = \psi_{12}(w) \). By Lemma A.3, \( \phi_{12}(v') = w \).

**Lemma A.10.** In Algorithm 1, if the label consistency holds at Step 4 for vertexes in \( H_4 \), it will still hold at Step 12 for vertexes in \( H_{12} \).

**Proof.** Consider any vertex in \( u \in H_{12} \). Either \( u \in V(H_4) \) or \( u \notin V(H_4) \). If \( u \in V(H_4) \), by assumption, \( l_u = |\psi_4(u)| \). But, \( \psi_4(u) = \psi_{12}(u) \), otherwise, it must be changed at Step 11, where \( u \) must be removed from \( H_4 \). Thus, \( l_u = |\psi_{12}(u)| \). If \( u \notin V(H_4) \), we must have \( u = w \) and \( l_u = l_w = \sum_{u \in U} \psi_4(u) \). By the semantics of \( \psi_4 \), \( \psi_4(u) \cap \psi_4(v') = \emptyset \) for any \( u \neq v' \). So \( \psi_4(U) = \bigcup_{u \in U} \psi_4(u) \). \( |\psi_4(U)| = \sum_{u \in U} |\psi_4(u)| \). Since \( \psi_{10}(w) = \psi_{10}(U) = \psi_4(U) \) by Table 1, \( |\psi_{10}(w)| = \sum_{u \in U} l_u \). Since \( \psi_{12}(w) = \psi_{10}(w) \), \( l_w = |\psi_{12}(w)| \).

**Lemma A.11.** In Algorithm 1, if at Step 4, the semantics of \( \psi \) holds and the label consistency holds for all edges in \( E(H_4) \), then at Step 12, the label consistency will also hold for all edges in \( E(H_{12}) \).

**Proof.** Consider any edge \( (u, u') \in E(H_{12}) \). Either \( (u, u') \in E(H_4) \) or \( (u, u') \notin E(H_4) \).

If \( (u, u') \in E(H_4) \), by assumption, \( l_{(u, u')} = |S_4(u, u')| \). Since \( (u, u') \in E(H_4) \) and \( (u, u') \in E(H_{12}) \), none of \( u \) and \( u' \) is \( w \) (because \( w \) is not in \( H_4 \)) or is in \( U \) at Step 11 (otherwise it must be removed already). By Table 1, \( \psi_{12}(u) = \psi_4(u) \) and \( \psi_{12}(u') = \psi_4(u') \). So \( S_{12}(u, u') = S_4(u, u') \). Thus, \( l_{(u, u')} = |S_{12}(u, u')| \).

If \( (u, u') \notin E(H_4) \), it must be a new edge added to \( H_4 \) at Step 8 or Step 9.

If \( (u, u') \notin E(H_4) \), it is labeled by the value \( \sum_{u, u' \in U \setminus (u, u') \in E(H_4)} l_{(u, u')} = \sum_{u, u' \in U} |S_4(u, u')| \) because \( S_4 = S_8 \) according to Table 1. Let \( (v, v') \) be an \( s \)-edge in \( S_{12}(w, w) \). By Table 1, \( \psi_{12}(w) = \psi_{10}(w) = \psi_4(U) \). Thus, \( S_{12}(w, w) = S_4(U, U) \). By the assumption and the semantics of \( \psi \), for any two distinct edges \( (u, u') \neq (s, s') \) in \( H_4 \), \( |S_4(u, u')| + |S_4(s, s')| = |S_4(u, u') \cup S_4(s, s')| \). So there must be a unique \( (u, u') \in H_4 \) s.t. \( (v, v') \in S_4(u, u') \). Thus \( S_4(U, U) = \bigcup_{u, u' \in U} S_4(u, u') \), and \( |S_{12}(w, w)| = \sum_{u, u' \in U} |S_4(u, u')| \).

If \( (u, u') = (w, w) \), it is labeled by the value \( \sum_{u \in U} |S_4(u, u')| \) where \( u' \in U \setminus NS(s) \cup U \). Similarly, if the new edge is \( (w, u') \) for some \( u' \in U \setminus NS(s) \cup U \), each \( s \)-edge \( (v, v') \in S_{12}(w, u') \) will be in \( S_4(u, u') \) for some \( u \in U \). This implies \( S_{12}(w, u') = S_4(U, U) = \bigcup_{u \in U} S_4(u, u') \). Again, due to disjoint sets of \( s \)-edges \( |S_{12}(w, u')| = |\bigcup_{u \in U} S_4(u, u')| = \sum_{u \in U} |S_4(u, u')| \).

**Theorem A.12.** (Same as Theorem 4.1) Algorithm 1 is correct.

**Proof.** We need to show that all four properties of Definition A.1 hold at Step 13. At Step 1, \( (H, \phi, \psi) \) is the N-map of graph \( G \), therefore the properties of Definition 3.4 are satisfied.

By Lemma A.2, the semantics of \( \psi \), the edge correspondence, and the label consistency hold at Step 3, and therefore at Step 4 of iteration 1. By Lemma A.3, A.7, A.10 and A.11, these three properties hold at Step 12 of iteration 1 and at Step 4 of iteration 2. Likewise, these three properties hold at Steps 4 and 12 of each iteration. Since \( S_2 \) must be \( \emptyset \) eventually, the while loop must terminate. So three properties hold at Step 13. We only need to prove that the semantics of \( \phi \) holds as the algorithm terminates.

Assume the algorithm executes \( n \) iterations. Let \( U_k \) and \( w_k \) denote the set \( U \) and the vertex \( w \) in iteration \( k \) of the loop. Consider any \( u \in V(H_{13}) \). Vertex \( u \) is \( w_k \) for some \( k \) \( (1 \leq k \leq n) \) or \( u \in V(H) \).
If $u = w_k$, by Lemma A.9, the semantics of $\phi$ holds on $u$ at Step 12 of iteration $k$, and at Step 4 of iteration $k + 1$. By Lemma A.8, the semantics of $\phi$ holds on $u$ in the subsequent iterations. So it also holds at Step 13.

If $u \in V(H)$, then by Table 1, $u \in V(H_3)$. The semantics of $\phi$ may or may not hold on $u$ at Step 3.

If the semantics of $\phi$ holds on $u$ at Step 3, it must also hold at Step 4 of iteration 1. By Lemma A.8, it holds on $u$ at Step 4 and 12 in each subsequent iteration. So it also holds at Step 13.

If the semantics of $\phi$ does not hold on $u$ at Step 3, we prove that $u \in U_k$ and $|U_k| > 1$ for some iteration $k$. Therefore, it will be removed from the N-map at Step 11 of iteration $k$. Since vertexes removed from the N-map in an iteration are never added back, $u \notin V(H_{13})$. As a result, for any vertex that is in both $V(H)$ and $V(H_{13})$, the semantics of $\phi$ must hold on $u$.

Since the semantics of $\phi$ does not hold on $u$ at Step 3, therefore, $\exists v, v' \in V(G')$, s.t. $v \neq v'$ and $\phi_3(v) = u$, either (1) if $\phi_3(v') = u, v \sim v'$ in $G'$ or (2) if $v \sim_N v'$ in $G'$, $\phi_3(v') \neq u$. By Table 1, $\phi_3 = \phi$. If $\phi_3(v') = u$, then $\phi(v) = \phi(v') = u$. Because $(H, \phi, \psi)$ is an N-map of $G$, $v \sim_N v'$ in $G$. Since $\phi_3 = \psi$, vertexes $v, v' \in \psi_3(u)$. By Lemma A.2, the edge correspondence holds at Step 3. So $\forall v \in \psi_3(u), NS(v) = \psi_3(NS(u)) - \{v\}$ in $G'$. Hence, $v \sim_N v'$ in $G'$. Therefore, if the semantics of $\phi$ does not hold on $u$, only (2) is possible. By Lemma A.2 the semantics of $\psi$, we can assume $\phi_3(v') = u' \neq u$ and $v' \in \psi_3(u')$ for some $u' \in V(H_3)$. By Table 1, if $\phi_3(v) \neq \phi_3(v')$, then $\phi(v) \neq \phi(v')$ implies $v \sim_N v'$ in $G$. However, $v \sim_N v'$ in $G'$. Hence, at least $NS(v)$ or $NS(v')$ is changed by $mbu(G|M)$.

Suppose $NS(v)$ is changed by $mbu(G|M)$. Thus, $v \in (W - \psi(\cap_{u \in M} NS(u))) \cup \psi(M)$. By the semantics of $\phi$, $u \in S_2$ at Step 3 of the algorithm. Then, vertex $u$ must be in $U$ at Step 5 in some iteration $k$. Let $u \in U_k$. We must have $\phi_4(v) = \phi_3(v) = u$. Otherwise, $u$ must be removed before iteration $k$. Since $v \sim_N v'$, by Lemma A.4, $\phi_4(v') \in U_k$. Because $\phi_3(v') = u', \phi_4(v') = u'$. Otherwise, $\phi_4(v') = w$, a new vertexes added at Step 7 in iteration $l < k$. By Lemma A.9, the semantics of $\phi$ holds on $w$, contradict with $w \neq u$ but $v \sim_N v'$. Thus, $u, u' \in |U_k|$. So $|U_k| \geq 2$. $U_k$ will be removed at Step 11. Thus, $u \notin V(H_{13})$.

A.3 Proof of Theorem 4.2

Proof. Let $H = \{(b_1, p_1), ..., (b_n, p_n)\}$ and $H' = \{(c_1, q_1), ..., (c_m, q_m)\}$ be the vertex degree histograms of $G$ and $G'$, respectively. Then, $\sum_{i=1}^n p_i = \sum_{j=1}^m q_j = |V(G)|$. Thus, for $H$ and $H'$, the constraints (ii)-(iv) of Eq. (4) become (ii) $\sum_{j=1}^m f_{ij} = p_i$, (iii) $\sum_{i=1}^n f_{ij} = q_j$, and (iv) $\sum_{i=1}^n \sum_{j=1}^m f_{ij} = |V(G)|$. We show that under these constraints, $f_{ij} = n_{ij}$ is the optimal solution to Eq. (4) if $d_{ij} = |b_i - c_j|$ and $n_{ij}$ is the number of vertexes whose degree changes from $b_i$ to $c_j$, if $c_j \geq b_i$; and 0, if $c_j < b_i$.

First, it is easy to verify that $f_{ij} = n_{ij}$ satisfies all constraints. Obviously, $n_{ij} \geq 0$ for each $n_{ij}$. Since the NEC merge may only add edges to vertexes, the degree of each vertex is either increased on stays the same in $G'$, thus, $p_i = \sum_{j=1}^m n_{ij}, q_j = \sum_{i=1}^n n_{ij}$, and $\sum_{i=1}^n \sum_{j=1}^m n_{ij} = |V(G)|$.

We now show that $f_{ij} = n_{ij}$ is the optimal solution.

Notice that for any given set of flow $f_{ij}$, $1 \leq i \leq n$ and $1 \leq j \leq m$, we have $\sum_{i=1}^n \sum_{j=1}^m f_{ij} \cdot |b_i - c_j| \geq |\sum_{i=1}^n \sum_{j=1}^m f_{ij} \cdot (b_i - c_j)|$. By constraints (ii) and (iii), $|\sum_{i=1}^n \sum_{j=1}^m f_{ij} \cdot (b_i - c_j)| \geq$
\[ |\sum_{i=1}^{n} p_i \cdot (b_i - c_1) + \sum_{j=2}^{m} q_j \cdot (c_1 - c_j)|. \] However,

\[
\sum_{i=1}^{n} \sum_{j=1}^{m} n_{ij} \cdot |b_i - c_j| = \sum_{i=1}^{n} \sum_{j=1}^{m} n_{ij} \cdot |b_i - c_j| + \sum_{i=1}^{n} \sum_{j=1}^{m} n_{ij} \cdot |b_i - c_j|
\]

\[
= |\sum_{i=1}^{n} \sum_{j=1}^{m} n_{ij} \cdot (b_i - c_j)| + |\sum_{i=1}^{n} \sum_{j=1}^{m} n_{ij} \cdot (b_i - c_j)|
\]

\[
= |\sum_{i=1}^{n} \sum_{j=1}^{m} n_{ij} \cdot (b_i - c_j)| + |\sum_{i=1}^{n} \sum_{j=1}^{m} n_{ij} \cdot (b_i - c_j)|
\]

\[
= |\sum_{i=1}^{n} \sum_{j=1}^{m} n_{ij} \cdot (b_i - c_j)|
\]

Thus, \( \sum_{i=1}^{n} \sum_{j=1}^{m} n_{ij} \cdot |b_i - c_j| \) is the minimum of \( \sum_{i=1}^{n} \sum_{j=1}^{m} f_{ij} \cdot |(b_i - c_j)| \). Since \( \sum_{i=1}^{n} \sum_{j=1}^{m} n_{ij} = V(G) \), \( EMD = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} n_{ij} \cdot |b_i - c_j|}{V(G)} \). Because \( \sum_{i=1}^{n} \sum_{j=1}^{m} n_{ij} \cdot |b_i - c_j| \) is the total number of vertices degree change, it equals to \( 2 \times |E(G') - E(G)| \).