Reconstructing randomized social networks

Niko Vuokko[*] Evimaria Terzi[†]

Abstract

In social networks, nodes correspond to entities and edges to links between them. In most of the cases, nodes are also associated with a set of features. Noise, missing values or efforts to preserve privacy in the network may transform the original network G and its feature vectors F. This transformation can be modeled as a randomization method. Here, we address the problem of reconstructing the original network and set of features given their randomized counterparts G' and F' and knowledge of the randomization model. We identify the cases in which the original network G and feature vectors F can be reconstructed in polynomial time. Finally, we illustrate the efficacy of our methods using both generated and real datasets.

Keywords: social-network analysis, privacy-preserving data mining

1 Introduction

Recent work on privacy-preserving graph mining [1, 8, 11, 15] focuses on identifying the threats to individual nodes' privacy and at the same time propose techniques to overcome these threats. The most simple such technique is *data randomization*: the original network is slightly randomized by the removal of some of its original edges and the addition of new ones. In this way, the network is expected to maintain its global properties, while individual nodes' privacy is protected.

In this paper, we give methods for reconstructing the original data given the randomized observations. In our setting each node in the social network is associated with a feature vector. We assume randomization methods that may randomize both the structure of the social network and the feature vectors of individual nodes. We show that the information encoded in the observed graph and feature vectors gives enough information to reconstruct the original version of the data. Of course, we assume that the data-randomization methods do not completely destroy the original dataset; had they done so, they would not serve their original goal to maintain the usefulness of the data.

The premise of our approach is captured in the famous quote from Plato: "Friends have all things in common". In other words, our basic assumption is that nodes that are connected in a social network have similar features and nodes that have similar features are highly probable to be connected in the network. This tendency is observed in real datasets: Figure 1 shows the total number of neighboring pairs of nodes that share more than $\phi = \{1, 2, \ldots, 10\}$ features[1] in some instance of the DBLP graph (solid line). The dashed line shows the corresponding number of neighboring pairs in a random graph that has the same nodes (and features) and the same number of edges as the original DBLP graph. Note, that the y-axis in the figure is in logarithmic scale. The number of edges in the original graph that connect nodes that share at least 1, 3, 5 and 7 features is 20670, 3416, 426 and 34 in the original graph. The corresponding numbers in the randomly generated graph are significantly smaller: 3708, 153, 10 and 0.

Although the motivation of our work comes primarily from privacy-preserving graph mining, our framework has also applications in reconstruction of noisy or incomplete datasets. For example, one can use our methodology to infer whether an individual has bought a specific product given some (noisy version) of the purchases of his friends in the social network.

Problem 1. *We solve the following reconstruction problem: given observed, randomized graph G' with feature vectors F' reconstruct the most probable true versions G and F; that is find G and F such that $Pr(G, F|G', F')$ is maximized.*

We assume that the graph is simple, i.e., the graph is undirected, unweighted, containing no self-loops or multiple edges. We study the following three variants of the above reconstruction problem.

1. $F' = F$ and $G' \neq G$: in this case only graph G has been randomized, while the features of individuals

[*] HIIT, Helsinki University of Technology, Finland. This work was mostly done when the author was at IBM Almaden Research Center.

[†] Boston University. This work was mostly done when the author was at IBM Almaden Research Center.

[1] Nodes correspond to authors; two authors are connected if they have written a paper together. Features encode each author's publishing venues.

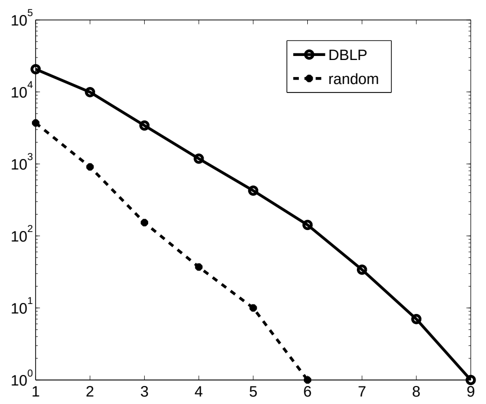

Figure 1: Number of edges in the DBLP graph that connect nodes that share at least $\phi = \{1, \ldots, 10\}$ features (solid line). The dashed line shows the same number for a random graph with the same number of edges as the original DBLP graph.

remain intact. We show that this problem can be solved optimally in polynomial time.

2. $F' \neq F$ and $G' = G$: in this case the features of individuals are randomized while the graph structure remains in its original form. We show that in this case the most probable original matrix F can be also reconstructed in polynomial time.

3. $F' \neq F$ and $G' \neq G$: this is the most general case where both the graph and the features are randomized. For this problem we distinguish some cases where optimal reconstruction can be achieved in polynomial time.

Remark 1.1. *Our framework can be combined with any data-randomization method as long as this method is known to the reconstruction algorithms. Here, we focus our attention on a randomization model similar to the one presented in [8]. However, our methodology is independent of the randomization procedure.*

1.1 Related work Our problem is primarily motivated by randomization techniques used for privacy-preserving graph analysis, like for example the ones presented in [1,8]. In [16] topological similarity of nodes is used to recover original links from a randomized graph. However, to the best of our knowledge we are the first to address the inverse problem of reconstructing a pair of original graph and feature vectors of individuals given their observed randomized versions.

In [14] the authors try to infer the links between individuals given their feature vectors. The main difference with our present work is that here we assume that a randomized version of the graph and the feature vectors is observed as part of the input – in [14] only feature vectors were given as part of the input. From the algorithmic point of view, the goal in [14] was to sample the space of possible graphs; here the goal is to find the most probable graphs and feature vectors given their randomized counterparts.

Related to ours is also the work on link-based classification – see [2, 9, 12, 13] and references therein. The goal there is to assign labels to nodes in a network not only by using the features appearing in the nodes, but also the link information associated with them. There are two main differences between our setting and collective classification: first, in collective classification, there are no randomized observations associated with every node. Therefore, the label of every node is decided based on the labels of its neighbors. In our case, the randomized version of the feature vectors F' can be used to decide whether a feature exists in a node or not. Even more importantly, link-based classification techniques assume that the network structure is not randomized but fixed. We, on the other hand, consider the case where the links between nodes are also randomized.

Link-based classification is a special case of the more general problem of classification with pairwise relationships among the objects to be classified. Such problems have been studied from many different view points (see [3] Ch. 8) and have been encountered in many applications domains (e.g., in [5, 7]). The difference between these problem formulations and ours is that in our case the pairwise relationships (i.e., links in the graph) between the objects to be classified are not fixed, but often randomized versions of the true relationships.

1.2 Roadmap The rest of the paper is organized as follows: in Section 2 we give the necessary notation and describe some basic assumptions about our model. The formal problem definitions are given in Section 3 and the algorithms are described in Section 4. Section 5 presents our experimental evaluation on both synthetic and real datasets, and we conclude the paper in Section 6.

2 Preliminaries

2.1 Basic notation Throughout we assume that there is an underlying *true* unweighted, undirected graph $G(V, E)$ with n nodes $V = \{1, \ldots, n\}$. Each node i is associated with 0–1 feature vector $\mathbf{f}_i \in \{0, 1\}^k$. That is, every node is associated with k binary features, such that $f_{i\ell} = 1$ if node i has feature ℓ. Otherwise, $f_{i\ell} = 0$. In the case of social networks, the binary feature vector of each individual may represent her hobbies, interests, diseases suffered, skills, products bought or conferences published in.

For ease of exposition, we represent graph $G(V, E)$ by its $n \times n$ adjacency matrix, which we also denote by G. We use g_{ij} to refer to the (i, j) cell of G, and we have that $g_{ij} = 1$ if there exists an edge between nodes i and j in the graph. Otherwise $g_{i,j} = 0$. Similarly we use F to denote the $n \times k$ 0–1 matrix that represents the features of the individual nodes. We use $\mathbf{f}_i$ to refer to the i-th row of matrix F, which in turn corresponds to the feature vector of node i, and $f_{i\ell}$ to refer to the ℓ feature of node i, which in turn is the (i, ℓ) cell of matrix F.

We assume that the existence of a certain edge in G is independent of the other edges of G and depends only on the feature vectors of endpoint nodes for the edge. As shown in [16], there are benefits in considering also the local topologies of nodes when recovering randomized networks. Incorporating this work into our approach and thus getting rid of the independence assumption is however left as future work.

2.2 Similarity functions As we have already discussed in the introduction, our basic assumption is that *nodes that are connected in G are likely to have similar feature vectors, and nodes that have similar feature vectors are likely to be connected in G.* In order to measure the similarity between two feature vectors, we need to define a *similarity function.* Our similarity functions, generally denoted by sim, map pairs of $\{0, 1\}^k$ vectors to positive reals. That is, $\text{sim} : \{0, 1\}^k \times \{0, 1\}^k \to \mathbb{R}$. We focus our attention to similarity functions that consider features independently and more specifically to similarity functions that are defined as the summation of the similarity of each individual attribute. That is, the similarity of vectors $\mathbf{f}_i$ and $\mathbf{f}_j$ is computed as

$$\text{sim}(\mathbf{f}_i, \mathbf{f}_j) = \sum_{\ell=1}^{k} \text{sim}(f_{i\ell}, f_{j,\ell}),$$

We focus our attention to two classes of similarity functions that we call **Dot Product (DP)** and **Hamming (H)**.

DP similarity functions : In these similarity functions the more 1-entries two vectors $\mathbf{f}_i$ and $\mathbf{f}_j$ share the more similar they are considered. The dot product similarity sim_{dp} between these vectors is an obvious instance of a **DP** similarity:

$$\text{sim}_{dp}(\mathbf{f}_i, \mathbf{f}_j) = \sum_{\ell=1}^{k} f_{i\ell} \cdot f_{j,\ell}.$$

H similarity functions : In these similarity functions, the more entries (either 1s or 0s) two vector share, the more similar they are considered. An example of such similarity function is the simple Hamming similarity function sim_H where

$$\text{sim}_H(\mathbf{f}_i, \mathbf{f}_j) = \sum_{\ell=1}^{k} \big(1 - |f_{i\ell} - f_{j,\ell}|\big).$$

We use similarity functions to capture the belief that the more similar the feature vectors $\mathbf{f}_i$ and $\mathbf{f}_j$ are, the more probable it is for the edge g_{ij} to exist. We model this by setting the probability of edge g_{ij} to be an exponential of $\text{sim}(\mathbf{f}_i, \mathbf{f}_j)$. That is,

$$\begin{aligned}\text{(2.1)}\quad & \Pr(g_{ij} = 1|\mathbf{f}_i, \mathbf{f}_j) = \\ &= \frac{1}{Z}\exp(\alpha\, \text{sim}(\mathbf{f}_i, \mathbf{f}_j)) \\ &= \frac{1}{Z}\exp\left(\alpha \sum_{\ell=1}^{k} \text{sim}(f_{i\ell}, f_{j\ell})\right),\end{aligned}$$

where Z and α are the appropriate normalization factors. Similarly,

$$\begin{aligned}\text{(2.2)}\quad & \Pr(g_{ij} = 0|\mathbf{f}_i, \mathbf{f}_j) = \\ & \frac{1}{Z}\exp\big(\alpha \sum_{\ell=1}^{k} (1 - \text{sim}(f_{i\ell}, f_{j\ell}))\big),\end{aligned}$$

and we require α and Z to be set so that $\Pr(g_{ij} = 1|\mathbf{f}_i, \mathbf{f}_j) + \Pr(g_{ij} = 0|\mathbf{f}_i, \mathbf{f}_j) = 1$. Since we assume that the existence of edges in G depends only on the feature vectors, we have

$$\text{(2.3)}\qquad \Pr(G|F) = \prod_{i<j} \Pr(g_{ij}|\mathbf{f}_i, \mathbf{f}_j).$$

2.3 Data randomization methods In our setting, we assume that the true values of G and F are not observed. Rather, only a randomized version of them is available. We denote these randomized versions by G' and F' respectively. Note that both G' and F' have the same number of nodes and features as the original G and F. The randomization method can only flip entries of the G and F tables: 0-entries can be transformed into 1s and vice versa.

To model the randomization, we assume no prior information about G or F and so we use uniform prior distributions for them. Several data-randomization methods can be adopted, each having its own advantages and disadvantages. For concreteness, we assume a specific randomization method that we call two-phase randomization. The method takes as input a 0–1 matrix X and integer value m and outputs a randomized version of this matrix $X' = \mathcal{R}(X, m)$. Matrix X' is created from X in two steps.

Step 1: Convert m, 1-entries from matrix X into 0s. In this way intermediate matrix X_1 is formed.

Step 2: Convert m 0-entries from matrix X_1 into 1s in order to form the final graph X'.

Assume that the original matrix X had a total of N entries (cells) and N_1 of those were 1s, then it is obvious that matrix X' will also have N_1 1s. For a fixed randomization method we can compute the probability of the original matrix X given the observed matrix X' by a simple application of the Bayes rule. That is,

$$\Pr(X|X') = \frac{\Pr(X'|X)\Pr(X)}{\Pr(X')}$$

Notice that since X' is the observed matrix, $\Pr(X')$ is fixed. Also, since we assume X to have a uniform prior distribution, $\Pr(X)$ is a constant. Therefore, the above equation becomes

$$\begin{aligned}\Pr(X|X') &\propto \Pr(X'|X) \\ &= \prod_{i,j} \Pr\left(x'_{ij}|x_{ij}\right).\end{aligned} \tag{2.4}$$

The last derivation assumes independence of the entries x_{ij} of matrix X. For the specific randomization method we discussed above, and for specific entries x and x' of matrices X and X' the probabilities $\Pr(x'|x)$ can be easily computed as follows:

$$\Pr(x'=0|x=0) = \frac{N-N_1}{N-N_1+m}, \tag{2.5}$$

$$\Pr(x'=1|x=0) = \frac{m}{N-N_1+m}, \tag{2.6}$$

$$\Pr(x'=0|x=1) = \frac{m}{N_1}\cdot\frac{N-N_1}{N-N_1+m}, \tag{2.7}$$

$$\Pr(x'=1|x=1) = \frac{N_1-m}{N_1} + \frac{m}{N_1}\frac{m}{N-N_1+m}. \tag{2.8}$$

Note, that Equations (2.5), (2.6), (2.7) and (2.8) unlike Equation (2.4) are specific to the randomization method we described above. We picked this randomization method because it is commonly used in privacy-preserving applications (see for example [8] and [1]).

In the above equations X can be instantiated by either G or F. When X is instantiated with G we have that $N = \binom{n}{2}$ and $N_1 = |E|$, while when it is instantiated with F we have that $N = nk$ and N_1 is the total number of 1-entries in feature matrix F. Matrices G and F are randomized independently from each other under this model.

3 Reconstruction problems

At a high-level, the problem we are trying to solve is the following: given observations G' and F' reconstruct the original forms of G and F. We call this problem the Reconstruction problem.

We adopt a maximum-likelihood approach where the goal is to find the *most probable* G and F given the observations G' and F'. That is, the objective is to find G and F such that $\Pr(G,F|G',F')$ is maximized. Using Bayes rule this probability can be easily decomposed as follows:

$$\begin{aligned}\Pr(G,F|G',F') &= \frac{\Pr(G',F'|G,F)\Pr(G,F)}{\Pr(G',F')} \\ &\propto \Pr(G',F'|G,F)\Pr(G,F) \\ &= \Pr(G'|G)\Pr(F'|F)\Pr(G,F).\end{aligned}$$

In the first step of the above decomposition we simply use Bayes rule. In the second step, we ignore the denominator, because $\Pr(G',F')$ is constant. Finally, the decomposition of $\Pr(G',F'|G,F)$ into the product $\Pr(G'|G)\Pr(F'|F)$ is due to the assumption that G and F are perturbed independently.

Probabilities $\Pr(F'|F)$ and $\Pr(G'|G)$ can be computed using Equation (2.4). The joint probability of G and F can be further decomposed as follows:

$$\begin{aligned}\Pr(G,F) &= \Pr(G|F)\Pr(F) \\ &\propto \Pr(G|F) \\ &= \prod_{i<j}\Pr(g_{ij}|\mathbf{f}_i,\mathbf{f}_j).\end{aligned}$$

Note that term $\Pr(F)$ was eliminated since F has a uniform prior distribution. Finally, factors $\Pr(g_{ij}|\mathbf{f}_i,\mathbf{f}_j)$ can be computed using Equations (2.1) and (2.2).

For conventional and practical reasons, instead of maximizing $\Pr(G,F|G',F')$ we minimize $-\log\Pr(G,F|G',F')$. We call this quantity the *energy* E of the reconstruction defined as,

$$\begin{aligned}E(G,F) &= -\log\Pr(G,F|G',F') \\ &= -\log\Pr(G'|G) - \log\Pr(F'|F) \\ &\quad -\log\Pr(G,F).\end{aligned} \tag{3.9}$$

Note that in the energy functions we use as arguments the variables whose value is unobserved. Therefore, G' and F' do not appear as arguments because they correspond to constants. The formal definition for reconstruction problems follows.

Problem 2. (Reconstruction problem) *Given a data-randomization method, a feature similarity function* sim, *observed graph G', and feature vectors F', find G and F such that $E(G,F)$ is minimized.*

Two special cases of the Reconstruction problem are of interest. In this section we define them formally and in the next section we give algorithms for solving them. Recall that the data-randomization method randomizes G and F independently. Therefore, it can be the case that only one of the two gets randomized and not necessarily both of them. The G-Reconstruction and the F-Reconstruction problems refer to exactly these cases.

We start with the G-Reconstruction problem where only the matrix G gets randomized to G', while the features maintain their original values. Since $F' = F$, we have that $\Pr(F'|F) = 1$ and the problem becomes that of finding the original graph G that minimizes energy function

$$\begin{aligned} E(G) &= -\log \Pr(G, F'|G', F') \\ &= -\log \Pr(G'|G) - \log \Pr(G, F') \\ &= \sum_{i<j} \Big(-\log \Pr\left(g'_{ij}|g_{ij}\right) - \log \Pr\left(g_{ij}|\mathbf{f}'_i, \mathbf{f}'_j\right) \Big). \end{aligned} \tag{3.10}$$

The elements in the summation can be evaluated using Equations (2.1), (2.2) and (2.4). This G-Reconstruction problem is defined formally below.

Problem 3. (G-Reconstruction problem) *Given a data-randomization method, a feature similarity function* sim*, randomized graph G', and feature vectors F', find G such that $E(G)$ (Equation (3.10)) is minimized.*

The F-Reconstruction problem is symmetric to Problem 3. The only difference is that now only the features F are randomized to F', while the graph maintains its original structure. Since $G' = G$, we have that $\Pr(G'|G) = 1$ and the the problem is to find the original vectors of features F that minimize energy function

$$\begin{aligned} E(F) &= -\log \Pr(G, F|G', F') \\ &= -\log \Pr(F'|F) - \log \Pr(G, F) \\ &= \sum_{i=1}^{n} \sum_{\ell=1}^{k} \left(-\log \Pr\left(f'_{i\ell}|f_{i\ell}\right) - \log \Pr\left(g'_{ij}|\mathbf{f}_i, \mathbf{f}_j\right)\right). \end{aligned} \tag{3.11}$$

As before the elements of the summation in Equation (3.11) can be computed from Equations (2.1), (2.2) and (2.4). Note now that the only unknown in the energy function are the feature vectors F. We call the problem of finding the most probable feature vectors the F-Reconstruction problem and we formally define it below.

Problem 4. (F-Reconstruction problem) *Given a data-randomization method, a feature similarity function* sim*, randomized feature vectors F', and graph G', find the feature vectors F such that $E(F)$ (Equation (3.11)) is minimized.*

4 Algorithms

We start this section by showing that the G-Reconstruction and F-Reconstruction problems can be solved optimally in polynomial time (Sections 4.1 and 4.2 respectively). The solution to the Reconstruction problem is slightly more involved and we discuss it in Section 4.3.

4.1 Algorithms for G-Reconstruction

The objective in the G-Reconstruction problem is to minimize energy $E(G)$ given in Equation (3.10). If we use

$$\beta(g_{ij}) \equiv -\log \Pr\left(g'_{ij}|g_{ij}\right) - \log \Pr\left(g_{ij}|\mathbf{f}'_i, \mathbf{f}'_j\right),$$

then the energy to be minimized can be simply written as a summation of $\binom{n}{2}$ terms, each one of which depends on a single g_{ij} value. That is,

$$E(G) = \sum_{i<j} \beta(g_{ij}).$$

In order to minimize $E(G)$ it is enough to minimize each term of the summation separately. Therefore, an optimal algorithm for the G-Reconstruction problem simply decides whether an edge g_{ij} exists or not in the original graph. That is, the algorithm goes through all the edges g_{ij} and if $\beta(g_{ij} = 1) < \beta(g_{ij} = 0)$ it sets $g_{ij} = 1$. Otherwise, it sets $g_{ij} = 0$.

We call this algorithm `OptG`. `OptG` has to go through all the $\binom{n}{2}$ edges and for each edge (i, j) compute the similarity between feature vectors $\mathbf{f}_i$ and $\mathbf{f}_j$. If the time required for a similarity computation is $\mathcal{O}(T_s)$ the running time of the `OptG` algorithm is $\mathcal{O}\left(n^2 T_s\right)$. In our case, $T_s = \mathcal{O}(k)$ and thus the running time of the `OptG` algorithm is $\mathcal{O}\left(n^2 k\right)$.

4.2 Algorithms for F-Reconstruction

We start this section by giving a polynomial-time algorithm for solving the F-Reconstruction problem optimally. We call this algorithm `OptF`; the basic principles of the algorithm originate from work on image restoration in computer vision (see for example [7, 10]).

Recall that in the F-Reconstruction problem, the goal is to find F such that energy function $E(F)$ (Equation (3.11)) is minimized. In other words, the goal is to find what values from $\{0, 1\}$ to assign to the nk variables $f_{i\ell}$, with $1 \le i \le n$ and $1 \le \ell \le k$.

The high-level idea of the `OptF` algorithm is to construct a *flow graph* $\mathcal{H}_F$. The flow graph $\mathcal{H}_F$ has one node $v_{i\ell}$ for every variable $f_{i\ell}$. In addition to these nk nodes it also has terminal nodes s and t. Weighted

Algorithm 1 The `OptF` algorithm for the F-Reconstruction problem.

```
    Input: Observed feature matrix $F'$ and graph $G' = G$.
    Output: Original feature matrix $F$.
1:  Construct flow graph $\mathcal{H}_F$
2:  $(S,T) \leftarrow$ Min-Cut($\mathcal{H}_F$)
3:  for all $v_{i\ell} \in S$ do
4:      $f_{i\ell} = 0$
5:  for all $v_{i\ell} \in T$ do
6:      $f_{i\ell} = 1$
```

directed edges connect these $nk+2$ nodes. The details of the edge addition process will be described shortly.

Assume a cut (S,T) of $\mathcal{H}_F$ such that nodes $s \in S$ and $t \in T$. Any such cut can be described by nk binary variables $f_{i\ell}$ with $1 \leq i \leq n$ and $1 \leq \ell \leq k$, such that $f_{i\ell} = 0$ if $v_{i\ell} \in S$ and $f_{i\ell} = 1$ if $v_{i\ell} \in T$. Therefore, there is a relationship between cuts in $\mathcal{H}_F$ and solutions to the F-Reconstruction problem. We formalize this relationship in the following definition.

Definition 1. (Min-Cut property of $E(F)$) *Energy function $E(F)$ has the* Min-Cut *property if there exists a flow graph $\mathcal{H}_F$ such that the minimum weight cut (S,T) of $\mathcal{H}_F$ that separates terminals s and t corresponds to the minimum-energy configuration of variables $f_{i\ell}$.*

The following lemma is a consequence of [10].

Lemma 4.1. *Energy function $E(F)$ given in Equation (3.11) has the* Min-Cut *property both for* **DP** *and* **Hamming** *similarity functions.*

Since energy function $E(F)$ has the Min-Cut property, the `OptF` algorithm simply constructs the flow graph $\mathcal{H}_F$ and then solves the Min-Cut problem on $\mathcal{H}_F$. This high-level idea of the `OptF` algorithm is depicted in the pseudocode shown in Algorithm 1. The following theorem is a direct consequence of Lemma 4.1.

Theorem 4.1. *For* **DP** *and* **Hamming** *similarity functions the* F-Reconstruction *problem can be solved optimally in polynomial time using the `OptF` algorithm.*

The running time of the `OptF` algorithm is the time required for constructing the flow graph ($\mathcal{O}(|E|k)$) and the time required for solving the Min-Cut problem on the flow graph. Note that in practical cases $|E| = \mathcal{O}(n)$ although it can be as high as $\binom{n}{2}$. If the time of the Min-Cut computation is $\mathcal{O}(T_C)$, then the total running time of the `OptF` algorithm is $\mathcal{O}(|E|k + T_C)$. The running time of the Min-Cut algorithm depends on the sparsity of the graph. Also, the flow graph $\mathcal{H}_F$ is very shallow; all paths from s to t are of length 2. Due to this special structure of the $\mathcal{H}_F$ graph we use the specialized Min-Cut algorithm introduced in [4]. A proper review of Min-Cut algorithms can be found in [6].

Here we give the details of the construction of the flow graph $\mathcal{H}_F$. Recall that the minimization function is $E(F)$, given in Equation (3.11). For simplicity of exposition consider the following transformation of Equation (3.11). Define

$$\begin{aligned}\gamma(f_{i\ell}) &\equiv -\log \Pr(f'_{i\ell}|f_{i\ell}),\\ \delta(f_{i\ell}, f_{j\ell}) &\equiv -\log \Pr(g'_{ij}|\mathbf{f}_i, \mathbf{f}_j).\end{aligned}$$

Then we can express $E(F)$ simply by

$$E(F) = \sum_{i=1}^{n}\sum_{\ell=1}^{k}\big(\gamma(f_{i\ell}) + \delta(f_{i\ell}, f_{i\ell})\big).$$

The flow graph $\mathcal{H}_F$ has $nk+2$ vertices $\{s, t, v_{11}, \ldots, v_{nk}\}$. Each non-terminal vertex $v_{i\ell}$ encodes a binary variable $f_{i\ell}$ whose value we want to determine.

Consider now variable $f_{i\ell}$: if $\gamma(f_{i\ell}=1) > \gamma(f_{i\ell}=0)$, then $f_{i\ell}$ is inclined towards taking value 0 and thus an edge $s \rightarrow v_{i\ell}$ is created with weight $\gamma(f_{i\ell}=1) - \gamma(f_{i\ell}=0)$. Otherwise, if $\gamma(f_{i\ell}=1) < \gamma(f_{i\ell}=0)$, variable $f_{i\ell}$ is inclined towards value 1, and thus directed edge $v_{i\ell} \rightarrow t$ is added in $\mathcal{H}_F$ with weight $\gamma(f_{i\ell}=0) - \gamma(f_{i\ell}=1)$. Finally for pairs of variables $(f_{i\ell}, f_{j\ell})$ such that $g'_{ij} = 1$, nodes $v_{i\ell}$ and $v_{j\ell}$ are connected with an undirected edge with weight $(\delta(0,0) + \delta(1,1) - \delta(0,1) - \delta(1,0))/2$.

Although the `OptF` algorithm solves F-Reconstruction optimally in polynomial time, it is quite inefficient for large values of n or k. A faster, though suboptimal, alternative is the `NaiveF` shown in Algorithm 2. This is a local iterative algorithm that in every iteration goes through all the entries of matrix F one by one and sets them to 0 or 1, depending on which value minimizes the energy $E(F)$. The notation $(F_{-i\ell}, f_{i\ell}=1)$ refers to a matrix that has the same entries as F in all cells except for cell (i,ℓ) that is set to 1. The running time of the `NaiveF` algorithm is $\mathcal{O}(I_{\max} nk T_S)$, where T_S is again the time required for computing the similarity between two vectors of dimensionality k. In our case, $T_S = \mathcal{O}(k)$ and thus the overall running time of `NaiveF` is $\mathcal{O}(I_{\max} nk^2)$. `NaiveF` is much more efficient than the optimal `OptF` algorithm; its complexity can be further improved by substituting some of the loops with matrix operations. Despite the running-time advantage the `NaiveF` algorithms gives solutions that are only locally, but not necessarily globally optimal.

Algorithm 2 The `NaiveF` algorithm for the F-Reconstruction problem.

```
    Input: Observed feature matrix $F'$ and matrix $G' = G$, and maximum number of iterations $I_{\max}$.
    Output: Original feature matrix $F$.
 1: $F = F'$ $iter = 1$
 2: while $iter < I_{\max}$ do
 3:     for $i = 1$ to $n$ do
 4:         for $\ell = 1$ to $k$ do
 5:             if $E(F_{-i\ell}, f_{i\ell} = 1) <$
 6:                 $E(F_{-i\ell}, f_{i\ell} = 0)$ then
 7:                 $f_{i\ell} = 1$
 8:             else
 9:                 $f_{i\ell} = 0$
10:     $iter = iter + 1$
```

4.3 Algorithms for Reconstruction problem

We start this section by observing that an algorithm similar to the `OptF` algorithm can be used for solving some instances of the Reconstruction problem optimally in polynomial time. We call this algorithm the `OptBoth` algorithm. However, the construction of the flow graph in this case is slightly more complex than the construction of graph $\mathcal{H}_F$ that we described in Section 4.2. Most importantly, the `OptBoth` algorithm solves the Reconstruction problem only when the similarity function used for computing the probability $\Pr(G|F)$ in Equation (2.3) is the **DP** similarity function. For general similarity functions we give an efficient greedy but suboptimal algorithm that we call `NaiveBoth` and which is a simple extension of the `NaiveF` algorithm that we described in Section 4.2 (Algorithm 2).

Simple, but lengthy, manipulations (not presented here) can transform the energy function $E(G,F)$ of Equation (3.9) to a summation of three terms as follows:

$$\begin{aligned} E(G,F) &= \sum_{i<j} \sigma_g(g_{ij}) + \sum_{i=1}^{n}\sum_{\ell=1}^{k} \sigma_f(f_{i\ell}) \\ &\quad + \sum_{i<j}\sum_{\ell=1}^{k} \sigma(g_{ij}, f_{i\ell}, f_{j\ell}), \end{aligned}$$

where

$$\begin{aligned} \sigma_g(g_{ij}) &= \alpha k g_{ij} - \log \Pr(g'_{ij}|g_{ij}), \\ \sigma_f(f_{i\ell}) &= -\log \Pr(f'_{i\ell}|f_{i\ell}), \\ \sigma(g_{ij}, f_{i\ell}, f_{j\ell}) &= \alpha(1-2g_{ij})\,\mathrm{sim}(f_{i\ell}, f_{j\ell}). \end{aligned}$$

The high-level description of the `OptBoth` algorithm is the same as this of `OptF` (Algorithm 1): Initially a flow graph $\mathcal{H}$ with terminal nodes s and t is constructed and then Min-Cut problem is solved in $\mathcal{H}$ to separate terminals s and t. The details of the construction of $\mathcal{H}$ are different from those of $\mathcal{H}_F$. After all, the corresponding objective functions $E(G,F)$ and $E(F)$ are also different.

If we can define a mapping between the nodes in $\mathcal{H}$ and the variables g_{ij} and $f_{i\ell}$ ($1 \le i \le n$, $1 \le \ell \le k$), then any cut (S,T) of $\mathcal{H}$ will correspond to an assignment of values in $\{0,1\}$ to these variables. There are cases in which this mapping actually exists. We formalize this relationship in the following definition that is similar to Definition 1 in Section 4.2.

Definition 2. (Min-Cut property of $E(G,F)$)
Energy function $E(G,F)$ has the Min-Cut *property if there exists a flow graph $\mathcal{H}$ such that the minimum weight cut (S,T) of $\mathcal{H}$ that separates terminals s and t corresponds to the minimum-energy configuration of variables g_{ij} and $f_{i\ell}$ with $1 \le i \le n$ and $1 \le \ell \le k$.*

The following lemma is a consequence of [10].

Lemma 4.2. *The energy function $E(G,F)$ given in Equation (3.9) has the* Min-Cut *property only for* **DP** *similarity function.*

Therefore, the `OptBoth` algorithm can only be used for the **DP** similarity function. We give now some of the details of the construction of $\mathcal{H}$. Apart from the terminal nodes s and t, $\mathcal{H}$ contains the following sets of nodes: (a) one node a_{ij} for every variable g_{ij} with $1 \le i \le n$ and $i < j \le n$, (b) one node $b_{i\ell}$ for every variable $f_{i\ell}$ with $1 \le i \le n$ and $1 \le \ell \le k$, and (c) one node $u_{ij\ell}$ for every triplet of variables $(g_{ij}, f_{i\ell}, f_{j\ell})$. Therefore there are a total of $\left(\binom{n}{2} + nk + \binom{n}{2}k + 2\right)$ nodes in $\mathcal{H}$.

Connections between nodes a_{ij} and $b_{i\ell}$ and between the latter and terminals s and t are made using the same principles as in the construction of $\mathcal{H}_F$. The only difference here is that functions σ_g and σ_f are used to evaluate edge weights, instead of function γ used in $\mathcal{H}_F$. Additional edges $a_{ij} \to u_{ij\ell}, b_{i\ell} \to u_{ij\ell}, b_{j\ell} \to u_{ij\ell}, u_{ij\ell} \to t$, with weight α are also added in $\mathcal{H}$. When the minimum cut (S,T), with $s \in S$ and $t \in T$ is found then all variables g_{ij} (or $f_{i\ell}$) for which $a_{ij} \in S$ (or $b_{i\ell} \in S$) are assigned value 0. Otherwise they are assigned value 1.

As we have already mentioned, the main disadvantage of the `OptBoth` algorithm is the size of the $\mathcal{H}$ graph. Even though the specialized Min-Cut algorithm of [4] can solve such problem instances relatively efficiently, the memory requirements are still a bottleneck for large graphs.

For arbitrary similarity functions, including Hamming similarity, the energy function $E(G,F)$ does not

Algorithm 3 The `NaiveBoth` algorithm for the general Reconstruction problem.

```
Input: Observed graph $G'$ and feature matrix $F'$, and maximum number of iterations $I_{\max}$.
Output: Original graph $G$ and feature matrix $F$.
1:  $F = F'$, $G = G'$, $iter = 1$
2:  while $iter < I_{\max}$ do
3:      for $i = 1$ to $n$ do
4:          for $\ell = 1$ to $k$ do
5:              if $E(G, F_{-i\ell}, f_{i\ell} = 1) >$
6:                  $E(G, F_{-i\ell}, f_{i\ell} = 0)$ then
7:                  $f_{i\ell} = 1$
8:              else
9:                  $f_{i\ell} = 0$
10:         for $j = i + 1$ to $n$ do
11:             if $E(G_{-ij}, g_{ij} = 1, F) >$
12:                 $E(G_{-ij}, g_{ij} = 0, F)$ then
13:                 $g_{ij} = 1$
14:             else
15:                 $g_{ij} = 0$
16:     $iter = iter + 1$
```

have the Min-Cut property. In this case we can solve the Reconstruction problem using the `NaiveBoth` algorithm. The pseudocode of the algorithm is shown in Algorithm 3. The `NaiveBoth` algorithm is a simple extension of the `NaiveF` algorithm; it works iteratively and in every iteration it goes through variables g_{ij} and $f_{i\ell}$ and fixes their value to the one that locally minimizes the energy $E(G, F)$. We will refer to the `NaiveBoth` algorithms utilizing either **DP** or **H** similarity with terms `NaiveBoth`(**DP**) and `NaiveBoth`(**H**) respectively.

4.4 Computational speedups All the algorithms we described above and particularly the optimal `OptF` and `OptBoth` algorithms are rather inefficient when it comes to graphs with large number of nodes and feature vectors with many dimensions. In order to increase efficiency, we resort to the classical method of dividing the input problem instance into smaller instances, solving the reconstruction problem in each of these instances and then combine the results. Algorithm 4 shows how the `Split` routine works for the general Reconstruction problem.

Routine BFS-Split (G, i, d) creates for every node i subgraph G'_i; G'_i is the subgraph of G' that contains all the nodes that are in distance at most d_1 from the node i in the Breadth-First (BFS) tree rooted at i. If this subgraph has size less than $r(n)$, then also nodes at distance d_2 are included. For reasonable speedups we set $d_1 = 1, d_2 = 2$ and $r(n) = \sqrt[3]{n}$. The reconstruction of G'_i, F'_i may be done by invoking any of the algorithms

Algorithm 4 The `Split` routine for speeding up the algorithms for Reconstruction problems.

```
Input: Observed graph $G'$ and feature matrix $F'$.
Output: Original graph $G$ and feature matrix $F$.
1: for $i \in \{1, \ldots, n\}$ do
2:     $(G'_i, F'_i) =$BFS-Split$(G', i, d_1, d_2, r)$
3:     $(G_i, F_i) =$RecAlgo$(G'_i, F'_i)$
4: $G =$Combine$(G_1, \ldots, G_n)$
5: $F =$Combine$(F_1, \ldots, F_n)$
```

described in the previous sections. In the pseudocode shown in Algorithm 4 we use `RecAlgo` to refer to *any* of these algorithms. Observe that every node $i \in V$ may participate in more than one subgraphs, and therefore more than one reconstruction values (either 0s or 1s) might be assigned to it. The `Combine` function simply assigns to every node i the value (0 or 1) that the majority of its reconstructions have suggested.

For any method `M` we will denote with `S-M` (e.g. `S-OptBoth`) the method that uses `Split` speed-up combined with `M`.

5 Experiments

The goal of the experimental evaluation is (a) to demonstrate the usefulness of our framework in reconstructing the original data and (b) to explore the advantages and shortcomings of the different algorithms for the reconstruction problems. All the reported results are obtained by running our C++ implementation of the algorithms on 2.3 GHz 64-bit Linux machine.

5.1 Data sets

5.1.1 Synthetic data: A synthetic dataset consisting of n nodes and k features per feature vector is generated as follows: first we generate n feature vectors. Then, each one of these vectors is associated with a node (randomly). Nodes are connected with probability proportional to the similarity of their corresponding feature vectors. In order to maintain some structure in the constructed graph the n feature vectors are generated by first generating K centroid feature vectors and then constructing the rest $n - K$ vectors to be noisy versions of one of the K centroid vectors. In the experiments we use a dataset generated as above for $n = 200$ and $k = 20$. We controlled the probability of edges so that the graph G generated has 557 total edges.

5.1.2 Real-world data: The *Dblp* dataset contains data collected from the DBLP Bibliography database[2]. For the purposes of our experiments, we selected 19 conferences, viz. WWW, SIGMOD, VLDB, ICDE, KDD, SDM, PKDD, ICDM, EDBT, PODS, SODA, FOCS, STOC, STACS, ICML, ECML, COLT, UAI, and ICDT. These conferences act as the $k = 19$ dimensions of our feature vectors. An author has a feature if he has published in the conference corresponding to this feature. The corresponding graph has authors as nodes; two authors are connected by an edge if they have co-authored at least two papers. By ignoring least prolific authors we obtain a dataset consisting of 4981 nodes and 20670 edges.

The second real dataset is the *Terror* data[3]. The nodes of the graph in the *Terror* data correspond to terrorist attacks and the features describe several detailed characteristics of the attacks. Two attacks are connected in the graph if they have occurred in the same location. The *Terror* dataset contains 645 nodes and 94 features. The corresponding graph has 3172 edges.

5.2 Results In this section we evaluate the effectiveness of the different algorithms with respect to the reconstruction task. We use two evaluation metrics: (a) the *minus loglikelihood* and (b) the *error ratio* of the obtained reconstructions. The minus loglikelihood of a reconstruction G and F is in fact the energy score $E(G, F)$ defined in (Equation (3.9)) (or its specializations $E(G)$ – Equation (3.10) – and $E(F)$ – Equation (3.11)). The lower the values of the negative loglikelihoods the better the reconstructions. The error ratio of a reconstructed matrix X given its randomized version X' and its original version X_0 is defined as the ratio

$$\frac{\text{number of differing entries between } X \text{ and } X_0}{\text{number of differing entries between } X' \text{ and } X_0}.$$

The lower the values of the error ratio the more similar the reconstructed matrices to the original dataset.

In all cases, we show the minus loglikelihoods and error ratios of the reconstructions as a function of the randomization amount m imposed on the original dataset. Recall that the randomization of the data is done using the randomization method described in Section 2. The larger the value of the parameter m the more noisy the randomized version of the data. In all cases, except if it is explicitly mentioned otherwise, we use the **H** similarity function in our computations.

[2]http://www.informatik.uni-trier.de/ ley/db/

[3]The dataset is available at http://www.cs.umd.edu/projects/linqs/projects/lbc/index.html

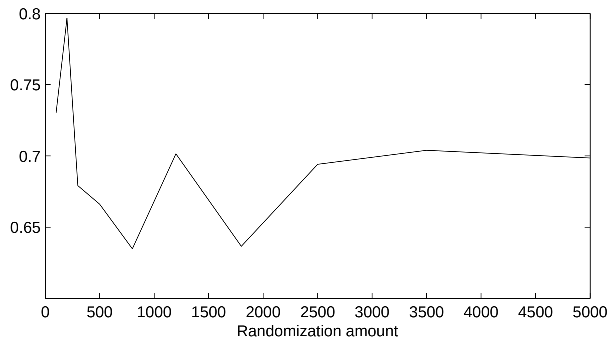

Figure 2: Error ratio of the reconstructions produced by `OptG` for the *Dblp* dataset.

5.2.1 Results for G-Reconstruction: Figure 2 shows the error ratios of the graph reconstructions obtained by the `OptG` algorithm. For the experiment we use the *Dblp* dataset and varying randomization amount $m \in \{ 100, 200, 300, 500, 800, 1200, 1800, 2500, 3500, 5000 \}$. For values of $m > 300$, the error ratio stabilizes to a value close to .675. Therefore, in the simple setting of G-Reconstruction, even on high levels of noise the `OptG` algorithm's performance doesn't suffer and it can still reconstruct many edges of the original network. Generally the reconstruction rate stays between .65 and .75 without significant variation.

A single run of `OptG` on the *Dblp* dataset does not exceed 5.1 seconds in running time.

5.2.2 Results for F-Reconstruction: In this experiment, we report the error ratio and the negative loglikelihoods of the reconstructions obtained by the `NaiveF` and the `OptF` algorithms. For this experiment we use the synthetic dataset *Gendata* and we test the performance of the algorithms for randomization amounts $m \in \{ 0, 15, 30, 45, 60, 75, 90 \}$. In order to be fair in the comparison of the two algorithms, for every experiment we first obtain the optimal reconstruction F by running `OptF`. Then, we run the same experiment with `NaiveF`; we fix the number of iterations of `NaiveF` so that the running times (in terms of clock time) of the two methods are approximately the same.

Figure 3 shows the error ratio of the reconstructions obtained by `OptF` and `NaiveF`. The results indicate that `NaiveF` performs almost as good as the optimal algorithm for small values of m; for higher values of m `OptF` exhibits noticeably better error ratio. Figure 4 shows that the loglikelihoods of the reconstructions obtained by the two algorithms are similar. That is, even for large values of m, the `NaiveF` heuristic gives reconstructions with energy scores surprisingly close to the energy scores of the optimal solution. Th comparative study of Figures 3 and 4 shows that the tw reported metrics (error ratio and loglikelihood) measu different qualities in the obtained reconstructions.

The running time of the `OptF` algorithm for th dataset is approximately 10 msecs.

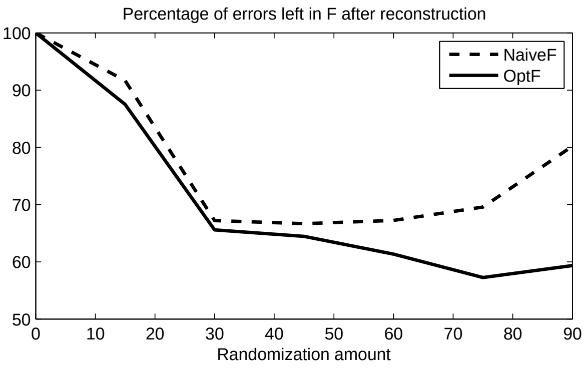

Figure 3: Error ratio of the reconstructions produced by `OptF` and `NaiveF` for the *Gendata* dataset.

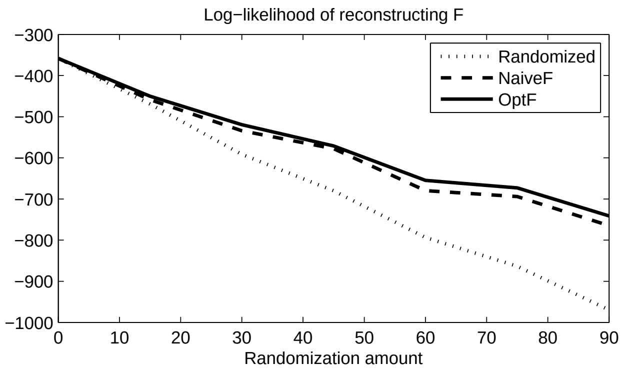

Figure 4: (Minus) Log-likelihoods (energy scores) of the reconstructions produced by `OptF` and `NaiveF` for the *Gendata* dataset.

5.2.3 Results for GF-Reconstruction**:** In this section we test the reconstructions obtained by the different algorithms for the the GF-Reconstruction problem. As before, we compare the methods by reporting the loglikelihoods and error ratios of their results. We only show here the results of the *Terror* dataset. The results for *Dblp* and *Gendata* are either similar or better but we omit their presentation due to space constraints.

Before presenting the actual results, several comments are in order: By Lemma 4.2, we know that `OptBoth` is only optimal when similarity function **DP** is used in the energy computations. Therefore, we need

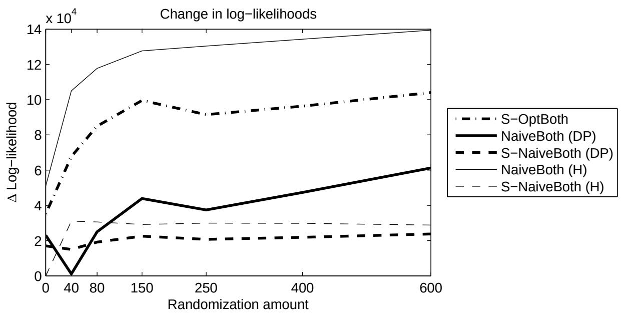

Figure 5: Improvement in the Loglikelihood (energy) scores of the reconstructions produced by `OptBoth` and `NaiveBoth` for the *Terror* dataset.

to distinguish the cases where we use **DP** or **H** similarity. Thus besides every algorithm we indicate in a parenthesis the similarity function it used for its calculations. Although `OptBoth` is used only for **DP** similarity, `NaiveBoth` can be used both with **DP** and **H**. The results of both `OptBoth` and `NaiveBoth` when combined with `Split` method are also reported. In fact, the results of the plain `OptBoth` method are omitted because the method failed to complete even for datasets of moderate size. For example, running `OptBoth` for the *Terror* data requires at least 12 GB of memory, and thus the task cannot be completed on a personal computer.

Figure 5 shows the improvement in the log-likelihood attained by each method for the *Terror* dataset; the improvement is measured with respect to the log-likelihood of the input (randomized) data. The change is computed using the optimization function every algorithm tries to optimize. For this experiment we fix the randomization parameter for feature vectors to 350 and vary the randomization parameter of the graph $m \in \{0, 40, 80, 150, 250, 400, 600\}$. From the plot, we can see that for **DP** similarity, `OptBoth` combined with `Split` speedup (i.e., `S-OptBoth`) achieves significantly larger improvements to the objective function than `NaiveBoth`(**DP**) and `S-NaiveBoth`(**DP**). For the **H** similarity function, the `NaiveBoth`(**H**) gives significantly better improvements that the version of `NaiveBoth`(**H**) that is combined with `Split` routine.

The running times of `S-OptBoth`, `NaiveBoth` (**DP**) and `NaiveBoth`(**H**) for the *Terror* dataset are on average 59, 20, 27 seconds respectively. A 10× speed-up is achieved through `Split` for `NaiveBoth` methods. The corresponding running times for *Gendata* are 2, 0.5 and 0.5 seconds and for *Dblp* 320, 15, and 35 seconds. The speedups achieved by `Split` are of the order of 3× and 20× for *Gendata* and *Dblp* data respectively. Therefore, using the `Split` speed-up leads to lower-quality results but it significantly improves the running time of the methods.

Our experience with the proposed methods suggests that the nature of the **DP** function lures the methods to fill up the matrices with too many ones. Every 1-entry makes other entries more probable to be set to one, which forms a vicious circle. This can be prevented either by careful parameter tuning (e.g., slightly modifying the correct values of α and Z in Equations (2.1) and (2.2)) or by using some adaptive penalties for adding 1-entries. The `NaiveBoth`($\mathbf{H}$) algorithm doesn't have the same problem, because any new 1-entries that are introduced may make other entries also less probably 1.

Note, that our objective function is such that there may exist plenty of reconstructions with high likelihood but low structural similarity to the original data. Taking structural constraints into account when finding the reconstructions is an issue that we plan to investigate further in the future.

6 Conclusions

Social network data, the network itself and (or) the feature vectors of nodes, may become randomized for various reasons. We studied the inverse problem: how to reconstruct the original network and the individuals' feature vectors given their randomized counterparts. We formally defined this reconstruction problem as a maximum-likelihood estimation problem and identified some interesting special cases. We also presented optimal and heuristic algorithms for solving the different variants of the problem. A set of preliminary experimental results in real and synthetic data illustrated the efficacy of our methods. In the future we plan to explore how our methods could be enhanced by considering structural constraints on the output reconstructions.

References

[1] Lars Backstrom, Cynthia Dwork, and Jon M. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW*, pages 181–190, 2007.

[2] Mustafa Bilgic and Lise Getoor. Effective label acquisition for collective classification. In *KDD*, pages 43–51, 2008.

[3] Christopher Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[4] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2002.

[5] P. Ferrari, A. Frigessi, and P. De Sá. Fast approximate maximum a posteriori restoration of multi-color images. *Journal of Royal Statistical Society B*, 1995.

[6] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998.

[7] D. M. Greig, B. T. Porteous, and A. H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of Royal Statistical Society*, 51:271–279, 1989.

[8] Michael Hay, Gerome Miklau, David Jensen, Don Towsley, and Philipp Weis. Resisting structural identification in anonymized social networks. In *VLDB*, 2008.

[9] David Jensen, Jennifer Neville, and Brian Gallagher. Why collective inference improves relational classification. In *KDD*, pages 593–598, 2004.

[10] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts? In *ECCV*, pages 65–81, 2002.

[11] Kun Liu and Evimaria Terzi. Towards identity anonymization on graphs. In *SIGMOD Conference*, pages 93–106, 2008.

[12] Qing Lu and Lise Getoor. Link-based classification. In *ICML*, pages 496–503, 2003.

[13] S. Macskassy and F. Provost. Classification in networked data: A toolkit and a univariate case study. *Journal of Machine Learning Research*, 8:935–983, 2007.

[14] Heikki Mannila and Evimaria Terzi. Finding links and initiators: a graph-reconstruction problem. In *SDM*, pages 1207–1217, 2009.

[15] Xiaowei Ying and Xintao Wu. Randomizing social networks: a spectrum preserving approach. In *SDM*, pages 739–750, 2008.

[16] Xiaowei Ying and Xintao Wu. On link privacy in randomizing social networks. In *PAKDD '09: Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 28–39, 2009.