GraSS: Graph Structure Summarization

Kristen LeFevre[*] Evimaria Terzi[†]

Abstract

Large graph databases are commonly collected and analyzed in numerous domains. For reasons related to either space efficiency or for privacy protection (e.g., in the case of social network graphs), it sometimes makes sense to replace the original graph with a *summary*, which removes certain details about the original graph topology. However, this summarization process leaves the database owner with the challenge of processing queries that are expressed in terms of the original graph, but are answered using the summary.

In this paper, we propose a formal semantics for answering queries on summaries of graph structures. At its core, our formulation is based on a random worlds model. We show that important graph-structure queries (e.g., adjacency, degree, and eigenvector centrality) can be answered efficiently and in closed form using these semantics. Further, based on this approach to query answering, we formulate three novel graph partitioning/compression problems. We develop algorithms for finding a graph summary that least affects the accuracy of query results, and we evaluate our proposed algorithms using both real and synthetic data.

1 Introduction

Graph-structured data is commonly collected and analyzed in a variety of application domains, e.g., the Web, internet, local-area networks (LANs), social and biological networks, and many more. In this paper, we investigate the problem of probabilistic query answering using compressed graphs, or *graph summaries.*

Graph structures are commonly summarized, or replaced with coarser representations, in at least the following two important (and very different) scenarios:

Compression, Space Efficiency: Over the past several years, particularly with the growth of the World Wide Web, the number and magnitude of graph-structured databases have been rapidly increasing, making it increasingly difficult and expensive to store the data contained therein. One common solution to dealing with the size of these databases is by expanding the amount of computing and storage resources (i.e., hardware and disks). On the other hand, a complementary solution deliberately removes some of the detail from the graphs in order to reduce space consumption.

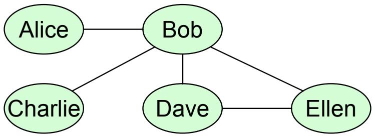


Figure 1: Social network graph

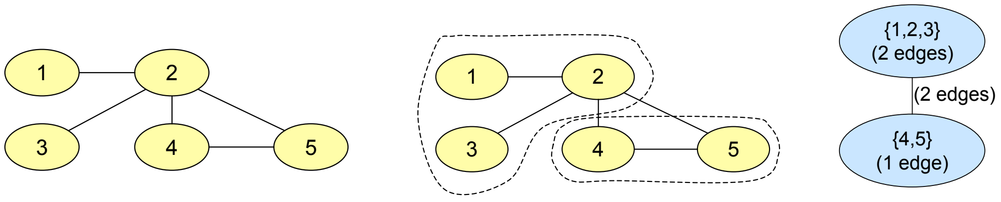
Privacy and Anonymity for Social Networks: Privacy and anonymity have emerged as important problems when publishing social network graphs (e.g., describing the "friend" relationships between users on Facebook or LinkedIn). Recent work has observed that removing known identifiers (e.g., Name, SSN) is often not enough to prevent re-identification [7, 20, 26]. As a simple example, consider the social network graph in Figure 1, and suppose that we replace it with the graph in Figure 2(a) (i.e., replace user names with meaningless integer pseudonyms). Now consider an attacker who knows that Bob is in the de-identified graph. If the attacker has some simple information about the graph topology surrounding Bob (e.g., Bob has 4 neighbors), then it is easy for the attacker to locate Bob.

In recent work, Hay et al. demonstrated that a graph summarization approach is sufficient to prevent this particular attack, even in the case of an adversary who has strong structural background knowledge (i.e., knows the entire network topology surrounding the target node) [20]. Essentially, by grouping each person (node) into a *super-node* containing at least $k-1$ other nodes, it is guaranteed that even a strong adversary will not be able to pinpoint a target individual beyond a group containing at least $k-1$ others.

While graph-structure summarization is important in multiple domains, there has not been much work on how to actually *use* these summaries for data analysis. The process of summarization, by definition, removes some information from the original graph. Intuitively, this introduces some uncertainty into any query or analysis that takes the summarized graph as input. To address this problem, we propose formal probabilistic semantics for evaluating structural queries on graph summaries.

[*] University of Michigan, Ann Arbor, MI. Email: klefevre@eecs.umich.edu

[†] Boston University, Boston, MA. Email: evimaria@cs.bu.edu



(a) Input graph (b) Partitioned graph (c) Graph summary

Figure 2: Graph summarization example

Summary of Contributions Our first main contribution is a formal semantics for query answering on graph summaries. We provide necessary definitions in Section 2, and then describe the query-answering semantics in Section 3. At its core, our approach is based on a random-worlds framework [6], and queries are answered using an expected value semantics. Interestingly, we show that several important graph structure queries (adjacency, degree, and eigenvector centrality) can be answered quickly and in closed form while still preserving our rigorous probabilistic semantics. More complex queries can be answered heuristically (with a high degree of accuracy) using a novel *expected adjacency matrix* structure.

Motivated by the above, we also study the problem of finding a "good" graph summary. In our problem setting, the quality of a summary is measured by the extent to which it alters the results of queries. In Section 4, we formulate three novel graph summarization problems. Two of these problems are motivated by compression and space efficiency; the third is motivated by privacy. For each of these problems, we describe algorithms for finding a good graph summary in Section 5. The graph summarization problems we study here give rise to three novel graph-partitioning problems that, to the best of our knowledge, have not been studied before. Finally, we provide an extensive experimental evaluation of our techniques in Section 6.

2 Graph Summarization Basics

Throughout this paper, we will consider an input graph $G(V, E)$ that is simple, undirected, and unweighted; V denotes the set of n nodes $V = \{v_1, \ldots, v_n\}$ and E denotes the set of edges among these nodes. Given an input graph $G(V, E)$ a summary $\mathbf{S}(G)$ consists of

1. A *partition* of the nodes of V into parts $\mathbf{V}(V) = \{V_1, \ldots, V_k\}$, such that $V_i \subseteq V$ and $V_i \cap V_j = \emptyset$, for $i, j \in \{1, \ldots, k\}$ and $i \neq j$. We refer to each group of nodes V_i as a *supernode* of the summary $\mathbf{S}$.

2. For every supernode $V_i \in \mathbf{V}$, summary $\mathbf{S}$ describes the *number of edges* within the nodes in the supernode. That is, it counts the number of edges in the input graph G that have both their endpoints in the nodes of V_i. For supernode V_i we denote this number by E_i. That is,
$$E_i = |\{e(u, v) \mid u, v \in V_i, e(u, v) \in E\}|\,.$$

3. For every pair of supernodes $V_i, V_j \in \mathbf{V}$, summary $\mathbf{S}$ also gives the *number of edges* across the two supernodes. That is, it counts the number of edges in the input graph G that have one of their endpoints in a node of V_i and their other endpoint in a node in V_j. For two supernodes V_i and V_j, we denote this number by E_{ij}.
$$E_{ij} = |\{e(u, v) \mid u \in V_i, v \in V_j, e(u, v) \in E\}|\,.$$

Example 1. *Consider the input graph in Figure 2(a). In this case, the nodes are uniquely named with integer values. Figure 2(b) shows a possible partition of the nodes of the input. Figure 2(c) shows the corresponding summary that consists of two supernodes.*

Note that for any input graph $G(V, E)$ there exist many possible summaries $\mathbf{S}$. In fact, there are as many as the possible partitions of the nodes in V. We use $\mathcal{S}(G)$ to denote the set of all (exponentially many) possible summaries that can be extracted from an input graph $G(V, E)$. In Section 4, we will discuss techniques for finding the *best* such partition / summary.

3 Probabilistic Queries on Graph Summaries

Replacing original graph G with summary $\mathbf{S}$ introduces uncertainty with respect to the structure of G. In this section, we formalize this intuition by defining the semantics of graph structure queries on summaries.

Our formalism is based on a random-worlds framework [6]. Intuitively, given a summary, there are many possible original graphs that could have produced the summary. Following the *principle of indifference* [6], in the absence of additional information, it is reasonable to assume that each such *reconstruction* is equally likely. Using this assumption, we define the *expected-value response* for any real-valued query $Q()$. Further, for several important kinds of graph structure queries – *adjacency*, *degree*, and *eigenvector centrality* – we show that the expected value response is easily computed from the summary in polynomial time.

3.1 Random Worlds and Reconstructions

Consider the case where we are given only a summary $\mathbf{S}$, but we want to find the answer to some structural query $Q()$ with respect to original graph G. In the absence of additional information, there are typically several graphs that could have produced summary $\mathbf{S}$. We refer to each such graph as a *reconstruction* of $\mathbf{S}$.[1]

Definition 1. (Graph Reconstruction) *Let* $\mathbf{S}$ *be a summary graph consisting of* k *supernodes* $\mathbf{V} = \{V_1, \ldots, V_k\}$ *and edge counts* E_i, E_{ij} *for* $i, j \in \{1, \ldots, k\}$. *The set of valid reconstructions of* $\mathbf{S}$, *denoted by* $\mathcal{R}(\mathbf{S})$, *is the set of graphs* $G(V, E)$, *such that*

- *For every* $i \in \{1, \ldots, k\}$,
$$|\{e(u,v) \mid u, v \in V_i, e(u,v) \in E\}| = N_i.$$
- *For every* $i, j \in \{1, \ldots, k\}$ *and* $i \neq j$,
$$|\{e(u,v) \mid u \in V_i, v \in V_j, e(u,v) \in E\}| = N_{ij}.$$

3.2 Expected-Value Semantics

To capture the uncertainty introduced by summarization, the semantics of query $Q()$ on $\mathbf{S}$ should be defined, conceptually, with respect to the set of *all* valid reconstructions. For related problems, two main approaches have been proposed: *expected value semantics* and *range semantics* (e.g., [5]).

Intuitively, using expected value semantics, the answer to a query $Q()$ is the expected result, given a distribution over possible reconstructions. In the absence of additional information, the *principle of indifference* suggests that it is reasonable to assume a uniform distribution.

Definition 2. (Expected Value Semantics) *Let* $\mathcal{R}(\mathbf{S})$ *denote the set of all valid reconstructions from summary* $\mathbf{S}$, *and let* $Q()$ *denote a query on* G *with a boolean or real-valued response.*[2] *Under expected value semantics, the answer to* $Q()$ *is defined to be the real number* e *such that*

$$e = \frac{\sum_{G \in \mathcal{R}(\mathbf{S})} Q(G)}{|\mathcal{R}(\mathbf{S})|}$$

Note that the above equation assumes uniform probability distribution over all graphs. Incorporating prior knowledge about the graph structure (e.g., giving preference to scale-free graphs) can be easily done by multiplying $Q(G)$ with the prior probability $\mathcal{P}(G)$ of the graph G.

	1	2	3	4	5
1	0	1	0	0	0
2	1	0	1	1	1
3	0	1	0	0	0
4	0	1	0	0	1
5	0	1	0	1	0

(a) Adjacency matrix for graph in Figure 2(a)

	1	2	3	4	5
1	0	2/3	2/3	1/3	1/3
2	2/3	0	2/3	1/3	1/3
3	2/3	2/3	0	1/3	1/3
4	1/3	1/3	1/3	0	1
5	1/3	1/3	1/3	1	0

(b) Expected adjacency matrix for summary in Figure 2(c)

Figure 3: Expected adjacency matrix example

3.3 Adjacency Queries

One of the simplest and most common graph-structure queries is the adjacency query, which simply asks: *Given graph* $G(V, E)$ *and two nodes* $u, v \in V$, *does there exist an edge* $(u, v) \in E$? Given a summary $\mathbf{S}$, the *expected adjacency matrix* captures the answers to all possible adjacency queries under expected value semantics.

Definition 3. (Expected Adjacency Matrix) *Let* $\mathbf{S}$ *be a summary graph. The expected adjacency matrix* $\overline{A}$ *for* $\mathbf{S}$ *is a* $|V| \times |V|$ *matrix, where all entries are real numbers in the range* $[0, 1]$ *defined as follows:*

$$\overline{A}(u,v) = \frac{\big|\{G(V,E) \mid G \in \mathcal{R}(\mathbf{S}), (u,v) \in E\}\big|}{\big|\mathcal{R}(\mathbf{S})\big|}$$

Given a graph summary, each of the entries in the expected adjacency matrix is easily computed in closed form.

Theorem 3.1. *Given summary* $\mathbf{S}$, *the entries of the expected adjacency matrix* $\overline{A}$ *given* $\mathbf{S}$ *can be computed as follows:*

1. *If* $u, v \in V$ *are distinct nodes in the same supernode* V_i, *then*
$$\overline{A}(u,v) = \frac{2E_i}{|V_i|(|V_i| - 1)} \tag{3.1}$$
2. *If* $u, v \in V$ *are distinct nodes in different supernodes,* V_i *and* V_j, *then*
$$\overline{A}(u,v) = \frac{E_{ij}}{|V_i| \times |V_j|} \tag{3.2}$$
3. *Otherwise (if* $u = v$*),*
$$\overline{A}(u,v) = 0 \tag{3.3}$$

[1] To use the terminology of [6], each reconstruction constitutes a *possible world*.

[2] For boolean queries true = 1 and false = 0.

The proof of the above theorem is based on basic probability manipulations and is omitted for space.

Example 2. *Figure 3(a) shows the adjacency matrix for the graph in Figure 2(a), and Figure 3(b) shows the expected adjacency matrix given the summary in Figure 2(c).*

3.4 Degree Queries

A *degree query* is another simple query of the form: *Given a node $v \in V$, how many edges in E touch v?* Under expected value semantics, we can define the answers to these queries using the expected adjacency matrix.

Theorem 3.2. *The expected degree of a node $v \in V$ is computed by $\overline{d}(v) = \sum_{j=1}^{|V|} \overline{A}(v,j)$*

Proof. Suppose that there are m valid reconstructions, and the degree of node v in reconstruction i is given by $d_i(v)$. Then, the expected degree of v is by definition $\overline{d}(v) = \frac{\sum_{i=1}^m d_i(v)}{m}$. Let $A_i(u,v)$ denote the adjacency matrix entry for nodes u and v in the i^{th} reconstruction. Of course, the degree of a node v can be computed using a graph's adjacency matrix. Thus, $d_i(v) = \sum_{j=1}^{|V|} A_i(v,j)$ and $\overline{d}(v) = \frac{\sum_{i=1}^m \sum_{j=1}^{|V|} A_i(v,j)}{m} = \sum_{j=1}^{|V|} \overline{A}(v,j)$.

Example 3. *Using the expected adjacency matrix in Figure 3(b), which is constructed for the summary in Figure 2(c), we have $\overline{d}(1) = 2$, $\overline{d}(2) = 2$, $\overline{d}(3) = 2$, $\overline{d}(4) = 2$, and $\overline{d}(5) = 2$.*

3.5 Eigenvector-Centrality Queries

Eigenvector centrality is one common way of measuring the importance of a node in a network.

Definition 4. (Eigenvector centrality) *Consider $G(V,E)$. The* eigenvector centrality score *of node v_i, denoted $p(v_i)$, is the probability of being at node v_i after infinite steps of a random walk on G. Alternatively, consider A to be the adjacency matrix of G and D a diagonal matrix of the degrees $D = diag(d(v_1), \ldots, d(v_n))$ of the n nodes in G. Also, let matrix $M = D^{-1}A$. Then the eigenvector centrality $p(v_i)$ of v_i is the i-th element of the left eigenvector of the matrix M that corresponds to the largest eigenvalue.*

Note that for the eigenvector centrality to be defined, the Markov chain that corresponds to G needs to be connected and aperiodic. Therefore, the theoretical analysis that follows refers to only such graphs. In practice, we can guarantee aperiodicity, with small impact on the final centrality scores, by adding self-loops with very small weights to each input node.

Theorem 3.3. *The expected eigenvector centrality score of a node $v \in V$ is computed in closed form by $\overline{p}(v) = \frac{\overline{d}(v)}{2|E|}$*

Proof. Recall that in undirected graph $G(V,E)$, the eigenvector centrality score $p(v)$ of $v \in V$ can be computed based on the degree of v as $p(v) = \frac{d(v)}{2|E|}$, where $d(v)$ is the degree of v in G (for details see [25] Chapter 6). Now, suppose that there are m valid reconstructions given a summary $\mathbf{S}$. If the degree (centrality score) of node v in reconstruction i is denoted by $d_i(v)$ ($p_i(v)$), then we have $\overline{p}(v) = \frac{\sum_{i=1}^m p_i(v)}{m} = \frac{\sum_{i=1}^m \frac{d_i(v)}{2|E|}}{m} = \frac{\overline{d}(v)}{2|E|}$

Example 4. *For the graph summary in Figure 2(c), we have $\overline{p}(1) = 1/5$, $\overline{p}(2) = 1/5$, $\overline{p}(3) = 1/5$, $\overline{p}(4) = 1/5$, and $\overline{p}(5) = 1/5$.*

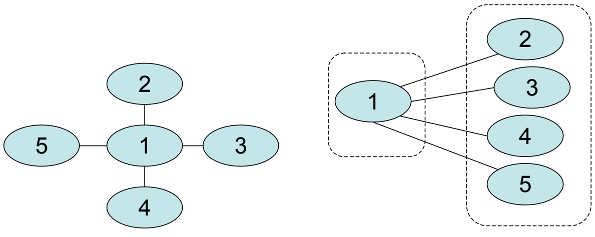
3.6 Complex Queries

In the previous three subsections, we have demonstrated that three simple kinds of queries (adjacency, degree, and eigenvector centrality) can be answered in closed form using expected value semantics. While strict adherence to the formal semantics is a worthy goal, and it is useful to continue to search for closed-form solutions to other queries, we observe that many queries can be answered heuristically, with reasonable accuracy, by viewing the expected adjacency matrix as a weighted graph. For example, in Section 6, we provide some experimental results for PageRank, indicating that this approach works fairly well.

4 Summarization Problems

In this section, we will turn our attention to the problem of finding a good summary, either when the goal is compression (or space efficiency), or when the goal is privacy protection. In both cases, we will measure the *quality* of a summary based on how well well the summary $\mathbf{S}(G)$ describes the input graph G. Given an input graph G, we want to find the summary $\mathbf{S}(G)$ such that G's adjacency matrix A and the expected adjacency matrix $\overline{A}$ of summary $\mathbf{S}(G)$ are as similar as possible. This intuition is captured by the *reconstruction error.* Intuitively, the reconstruction error measures the average absolute error (resulting from using summary $\mathbf{S}$ rather than G) across all possible adjacency queries.

Definition 5. (Reconstruction Error (Re)) *Let $G\ (V,E)$ be an input graph described by a ($|V| \times |V|$) adjacency matrix A. Let $\mathbf{S}$ be a summary of G, and let $\overline{A}$ be the expected adjacency matrix for $\mathbf{S}$. We define the (normalized) reconstruction error of $\mathbf{S}$ with respect*



(a) Input graph (b) Optimal partition ($k = 2$)

Figure 4: Novel partitioning problem

to G as follows:

$$\textsc{Re}\left(A \mid \overline{A}\right) = \frac{1}{|V|^2} \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} \left|\overline{A}(i,j) - A(i,j)\right|.$$

In the rest of this section we formally define the optimization problems that we study in the rest of the paper. Namely, we define three problems: k-Graph Summarization (k-Gs), Graph Summarization (Gs), and k-Capacitated Graph Summarization (k-CGs). k-Gs and Gs are inspired by the space efficiency problem; k-CGs is inspired by the privacy problem.

4.1 Space Efficiency Problems (k-Gs and Gs)

In the k-Gs problem, the maximum number of supernodes k in the output summary is specified as a constraint. Then, for a given input graph, the goal is to find the best summary graph with at most k supernodes. In contrast, in the Gs problem, we adopt a *Minimum Description Length* (MDL) formulation, rather than assuming that a fixed k is specified in advance. In that case, the goal is to find the best summary graph $\mathbf{S}$ so that exactly the right number of supernodes are included in the summary. The right number of supernodes is defined using information-theoretic arguments, and it depends on the complexity of the summary and the error that the summary introduces in the description of the data.

In the rest of the section we formally define each one of the above problems.

Problem 1. *Given input graph $G(V, E)$ and integer k, find a summary graph $\mathbf{S}$ for G with at most k supernodes $\mathbf{V}$ ($|\mathbf{V}| \leq k$), such that $\textsc{Re}(G \mid \mathbf{S})$ is minimized.*

Note that Problem 1 is a novel graph-partitioning problem. Although other variants of graph partitioning have been extensively studied (see Section 7), their focus has been on finding dense graph components, or (approximately) disconnected parts in the graph. Our problem has a different flavor; the nodes that belong to a partition (supernode) need not necessarily be densely connected; being disconnected is equally good. The special nature of the problem is illustrated by the following example.

Example 5. *Consider the star-shaped graph shown in Figure 4(a). The optimal partitioning solution to Problem 1 for $k = 2$ is shown in Figure 4(b). The summary graph resulting from this partitioning would have reconstruction error $\textsc{Re} = 0$. Such a partitioning is not optimal with respect to any other graph-partition problem.*

We devote the rest of this subsection to describing a variant of Problem 1 where the maximum number of supernodes k is not given as part of the input, but rather the objective function is such that it decides by itself the optimal number of supernodes. We achieve this balance by using the MDL principle to guide our problem formulation.

The Minimum Description Length (MDL) principle: The MDL principle states the following: assume two parties P and P' want to communicate with one another. More specifically, assume that P wants to send a graph G to P' using as few bits as possible. In order for P to minimize the communication cost, she has to select the model $\mathbf{S}$ from the class of models $\mathcal{S}$, and use $\mathbf{S}$ to describe her data. Then, she can send P' model $\mathbf{S}$, plus the additional information required to describe G, given the transmitted model. In our case, the class of models were are considering is the class of summary graphs; thus, P has to encode the summary graph, plus the data given the summary graph. The quality of the selected model (summary graph) can be evaluated based on the total number of bits required to encode the model and the data given the model.

MDL discourages complex models with small data cost and simple models with large data cost. Intuitively, it strikes a balance between these two extremes. Clearly, the MDL principle is more general than our graph summarization problem. In the past, it has been successfully applied in a variety of settings that include decision-tree classifiers [24], genetic-sequence modelling [23], patterns in sets of strings [22], and many more. We now describe our instantiation of the MDL principle.

Cost of the data given the model: Our input data is graph $G(V, E)$, while our model is the summary $\mathbf{S}$ of the graph. Let A and $\overline{A}$ be the adjacency matrix of G and the expected adjacency matrix given $\mathbf{S}$, respectively.

For now, assume that the permutations of the rows (and columns) of A and $\overline{A}$ are the same. That is, the i-th row (column) of A corresponds to the i-th row (column) of $\overline{A}$. Recall that the value of $\overline{A}(i, j)$ corresponds to the probability of the existence of an edge between nodes i and j of the original graph. Then, the probability of $A(i,j) = 1$ is

$$\Pr\big[A(i,j) = 1\big] = \overline{A}(i,j),$$

while the probability of $A(i,j) = 0$ is

$$\Pr\big[A(i,j) = 0\big] = 1 - \overline{A}(i,j).$$

Therefore, given model $\mathbf{S}$ described with the expected adjacency matrix $\overline{A}$, the probability of observing the original graph G, with adjacency matrix A is

$$\Pr\big[A \mid \overline{A}\big] = \prod_{i,j} \Big(\overline{A}(i,j)^{A(i,j)} \times \big(1-\overline{A}(i,j)\big)^{\big(1-A(i,j)\big)}\Big).$$

Subsequently, the number of bits required to describe A given $\overline{A}$ is $-\log\Pr\big[A \mid \overline{A}\big]$[3] and thus

$$\begin{aligned} & B\left(A \mid \overline{A}\right) \\ = \; & -\log\Pr\big[A \mid \overline{A}\big] \\ = \; & \sum_{i,j}\Big(-A(i,j)\log\overline{A}(i,j) - \\ & -\big(1 - A(i,j)\big)\log\big(1-\overline{A}(i,j)\big)\Big). \end{aligned}$$

Cost of describing the model: Our model is the summary $\mathbf{S}$ that is, in principle, described by the expected adjacency matrix $\overline{A}$. Here we will describe the cost (in bits) of describing the model $\overline{A}$. Assume that the summary graph $\mathbf{S}$ has k supernodes, then matrix $\overline{A}$ is of size $n \times n$ ($|V| = n$), but the n rows and columns can be rearranged so that all the nodes that belong to the same supernode in $\mathbf{S}$ are close to each other. Such a rearrangement will produce a matrix that consists of $k \times k$ blocks. Notice that the entries of matrix $\overline{A}$ within every such block are constant; that is, there is a single value that can describe the probability of an edge between nodes in the block. In order to encode this matrix we need to encode the following information:

1. We need to encode the permutation of the rows (and columns) used for the rearrangement in matrix $\overline{A}$. Since there are $n = |V|$ rows, $n\log n$ bits are needed to encode this permutation.
2. We need to specify the partitioning of the rows (and columns) of $\overline{A}$ into k groups. Given the permutation, it is enough to specify the indices of the rows where a new block starts. For k blocks, $k-1$ indices need to be specified which require a total of $(k-1)\log n$ bits. Note that we do not need to specify the indices of the columns since the permutation of the rows and the columns of matrix $\overline{A}$ are identical.
3. Each one of the $k \times k$ blocks of matrix $\overline{A}$ is characterized by a single parameter, namely the probability of an edge between any two nodes in the supernode(s) associated with the block. This parameter is a real number, and by standard information-theoretic arguments we need $\frac{1}{2}\log n^2 = \log n$ bits to encode it. Since there are k^2 such parameters, the cost of their description sums up to $k^2\log n$ bits.

Therefore, the total number of bits required to encode the model is

$$\begin{aligned} B\left(\overline{A}\right) &= n\log n + (k-1)\log n + k^2\log n \\ &= (n + k^2 + k - 1)\log n. \qquad (4.4) \end{aligned}$$

Putting it all together: Using the standard MDL methodology, we define our problem as that of finding the model $\mathbf{S}$ (or $\overline{A}$) such that the minimum total number of bits is used. This is formalized in the following problem definition.

Problem 2. *Given input graph $G(V,E)$ with adjacency matrix A, find a summary $\mathbf{S}$ of this graph with expected adjacency matrix $\overline{A}$ such that the total number of bits*

$$\textsc{Tb}\left(\overline{A}\right) = B\left(\overline{A}\right) + B\left(A|\overline{A}\right) \qquad (4.5)$$

is minimized.

4.2 Privacy Problem (k-CGs)

In the k-CGs problem we still want to find the graph summary that minimizes the reconstruction error. However, the difference between this and the k-Gs problem is that we restrict the graph summary so that all supernodes in the summary must contain at least k nodes from the original graph.[4]

The k-CGs problem is formalized as follows. This formalizes the k-anonymity requirement [30, 31] on graphs; every node in the original graph must be hidden in the anonymity of the supernode to which it belongs. At the same time, we seek to maximize the utility of the anonymized graph.

Problem 3. *Given input graph $G(V,E)$ and integer k, find a summary graph $\mathbf{S}$ for G with supernodes $\mathbf{V}$ such that* $\textsc{Re}(G \mid \mathbf{S})$ *is minimized and for* every $V' \in \mathbf{V}$ $|V'| \geq k$.

[3] Recall that the number of bits needed to describe event e that occurs with probability p_e is $-\log p_e$

[4] The parameter k is given as input to the problem; note that the semantics is different from the semantics of parameter k in the k-Gs problem!

5 Algorithms

In this section we give algorithms for Problems 1, 2 and 3. Note that for Problems 1 and 2 we are going to use the same `Greedy` algorithm. On the other hand, our algorithmic solution to Problem 3 is different.

5.1 Solving the k-Gs and Gs problems We start by describing a natural greedy algorithm for Problems 1 and 2. The `Greedy` algorithm starts with a summary graph in which each node is placed in a separate supernode. In each step, it merges the two supernodes that cause the largest reduction in the objective function. The algorithm repeats until a stopping condition is met. For the k-Gs problem the stopping condition is that the summary graph contains k super-nodes, while for the Gs version of the problem the stopping condition states that there is no merging of super-nodes that can improve the objective function Tb (Equation 4.5).

We give a detailed description of `Greedy` for when the goal is to minimize the Re objective function. The algorithm is identical when used to optimize objective function Tb. As we mentioned before, the algorithm proceeds in iterations. Originally, the summary graph $\mathbf{S}^0$ is identical to the input graph. At the t-th step, t super-nodes have been merged and summary graph $\mathbf{S}^t$ with $n-t+1$ nodes has been created. At each step t the algorithm needs to perform two tasks: (a) find the pair of super-nodes in $\mathbf{S}^{t-1}$ such that summary graph $\mathbf{S}^t$ generated after their merge has the property that $\text{Re}\left(A|\overline{A}^{t-1}\right) - \text{Re}\left(A|\overline{A}^{t}\right)$ is maximum and (b) actually perform the merge of the right pair of nodes and update summary $\mathbf{S}^{t-1}$ to summary $\mathbf{S}^t$.

Step (a) requires going through all pairs of nodes in $\mathbf{S}^{t-1}$ and computing from scratch the reduction in Re that their merge would cause.[5] This can be naively done in $O(n^3)$ time, since one has to go through all pairs of super-nodes ($O(n^2)$) and for each such pair compute the total reduction in cost function Re. This latter step can be done, with the right bookkeeping, in $O(n)$ time. Step (b) requires updating $\mathbf{S}^{t-1}$ to $\mathbf{S}^t$ by simply removing the two individual super-nodes that have been merged and adding the new super-node that results form the merging. Thus, the time required for step (b) is just $O(n)$ since both the old super-nodes that are removed and the new super-node that is added can have at most $O(n)$ neighbors. Therefore, the total running time of the above implementation of steps (a) and (b) is $O(n^3+n) = O(n^3)$. For the k-Gs problem, steps (a) and (b) need to be repeated $n-k+1$ times, so the total running time of the algorithm is $O(n^4)$. Apparently, this running time is prohibitive for large datasets. The algorithms described in the next two subsections aim at reducing this running time.

5.1.1 The `SamplePairs` algorithm As for the `Greedy` algorithm, we describe the `SamplePairs` algorithm for the k-Gs problem and the Re optimization function.

In fact, the `SamplePairs` algorithm is an instance of the `Greedy` algorithm. Thus, `SamplePairs` also proceeds in rounds. At every round t a summary $\mathbf{S}^t$ is constructed by merging two supernodes of the $\mathbf{S}^{t-1}$ summary. The pair of supernodes to be merged are the ones that after merging, they cause the smallest increase (or the largest reduction) in the objective function. The only difference here is that *not all pairs of supernodes* are checked for the impact that their merge causes in the optimization function. Instead, in every iteration t, a sample $P(t)$ of the supernode pairs are selected uniformly at random. The pair in $P(t)$ that increases the least (or decreases the most) the objective function is picked. The next iteration is executed analogously by picking a new set of candidate supernodes for merging.

By following the same line of thought as before in the running-time analysis we have that the running time of the `SamplePairs` algorithm is $O\left(|P(t)|\, n^2\right)$. Therefore, if we pick a constant number of pairs in every round K, the running time of the `SamplePairs` algorithm is quadratic to the number of nodes in the input graph.

Alternatively, at every iteration t we may sample pairs $P(t)$ with cardinality proportional to the number of supernodes $n(t)$ in the graph summary at iteration t: $|P(t)| = c \cdot (n(t))$. In this case, the running time is $O\left(n^3\right)$, which is still an improvement to the `Greedy` algorithm.

5.1.2 The `LinearCheck` algorithm The `LinearCheck` algorithm is similar in spirit to the `SamplePairs` algorithm. Recall that in `SamplePairs`, at every iteration t, $P(t)$ pairs of supernodes were sampled as candidates for merging. In the `LinearCheck` algorithm, at every iteration t, a single supernode V_t is picked uniformly at random among all the supernodes in summary $\mathbf{S}^{t-1}$. Then, the gain in the objective function obtained by merging V_t with any of the supernodes in $\mathbf{S}^{t-1}$ is evaluated. The merge with the best gain is picked and the summary $\mathbf{S}^t$ is constructed.

By following the analysis of the previous two algorithms it is easy to see that the worst-case running time of the `LinearCheck` algorithm is $O\left(n^3\right)$. However, note that in this case, as well as in the `Greedy` and `SamplePairs` algorithms these are all worst-case run-

[5] The reduction in the value of Re can be both positive or negative.

ning times. In practice, depending on the number of neighbors of each node in the graph the running time of the algorithms is much less than the corresponding worst-case bounds.

5.2 Solving the k-CGs problem The algorithm we use for solving the k-CGs problem is again an iterative algorithm; we will call the algorithm `Condense` due to its similarity to the condensation algorithm proposed by Aggarwal and Yu for (non-graphical) tabular data [3].

Let $\mathbf{S}^i$ be the summary graph when iteration i starts. Also let V^i be the set of original nodes that have not yet been assigned to a supernode in $\mathbf{S}^i$. In iteration i, the `Condense` algorithm randomly selects a node v from V^i, and forms a "ball" around v. This ball, denoted by $B(v)$, consists of $k-1$ other nodes from V^i, that are chosen so that when combined with v in a supernode they cause the least increase in the current value of the reconstruction error. On each iteration, nodes $B(v) \cup \{u\}$ are added to the summary graph $\mathbf{S}^i$ as a new supernode.

At every step of the `Condense` algorithm the supernodes that are added to the summary have size *exactly* k. If in the last iteration fewer than k nodes remain without being assigned to a supernode, we assign them to already existing supernodes. Tha assignment is done sequantially (in random order) so that the representation error is minimized at each step.

Finally, on each iteration, we must find the ball $B(v)$ around node v that minimizes the increase in reconstruction error. This may not be a computationally easy task. Thus, we use a simple greedy process for forming balls around the nodes in the original graph.

6 Experimental Evaluation

6.1 Datasets We evaluated our proposed algorithms using both real and synthetic graph data. The real data sets were as follows:

- **Authors:** The first data set is an undirected co-authorship graph. We studied the bibliographies for VLDB, SIGMOD, and PODS available at the Collection of Computer Science Bibliographies[6] and extracted the lists of authors from each paper. A selection of papers with at least 2 authors yielded 3109 papers with a total of 9538 authors (nodes in the graph). However, in some experiments, we reduced the size of the graph by sampling a smaller subset of the nodes.
- **Italian Wikipedia:** The second graph was extracted from the link structure of the Italian-language version of Wikipedia, and the full graph contains 15,000 nodes. While the original graph is directed, in all of our experiments, we view the graph as undirected.

In addition, we constructed a synthetic data generator, which generates undirected graphs according to two distributions:

- **Uniform**(n, d)**:** The edges in these graphs are chosen according to a uniform distribution based on a *density* parameter d. That is, suppose we want to generate a uniform graph with n nodes. Each of the $n(n-1)$ possible edges (no self-edges) is selected with probability d.
- **Barabasi-Albert**(n, m_0, m)**:** These graphs are generated according to the procedure proposed by Barabasi and Albert [8]. The graph begins with an initial network of m_0 nodes, and new nodes are added one at a time. Each node is connected to m of the existing nodes with probability proportional to the number of links the node already has.

6.2 Algorithms for k-Gs and Gs Our first set of experiments evaluates the algorithms that we proposed for the k-Gs and Gs problems. (These problems are the space-limited and MDL problem formulations, respectively.)

6.2.1 Comparing Algorithms Our first set of experiments compares the algorithms proposed for these problems in Section 5.1 – `SamplePairs`, and `LinearCheck` – in terms of resulting reconstruction error. Note that due to the high time complexity of the `Greedy` algorithm, we did not run experiments using the full enumeration of possible merges. However, we do observe that as we increase parameter c, the results for `SamplePairs` appear to converge. In addition, for the sake of comparison, we include two other algorithms: the first is a naive `Random` algorithm, which operates in a bottom-up manner, and at each iteration selects a random pair of nodes to merge. The second is a state-of-the-art community detection algorithm called `FastGreedy` [12]. This latter algorithm is designed to solve a different problem; the one of detecting dense, and relatively isolated components of the input graph.

Figures 5, 6, and 7 show our results for the Authors data set (samples of size 500 and 5,000), and Italy data set (sample of size 5000). Recall that the reconstruction error intuitively measures the average absolute error over all possible adjacency queries. As expected, the `LinearCheck` and `SamplePairs` algorithms significantly outperform `Random`, and within `SamplePairs`, as we increase the value of c, the overall quality of the results improve, appearing to near a point of convergence.

[6] `http://liinwww.ira.uka.de/bibliography/`

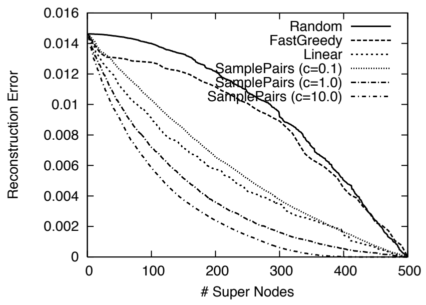


Figure 5: k-Gs, Authors(500)

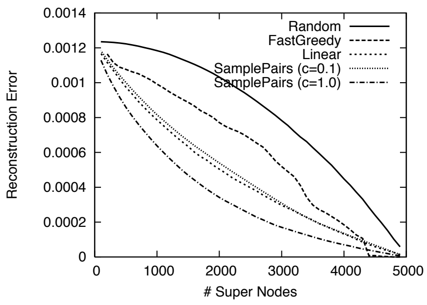


Figure 6: k-Gs, Authors(5000)

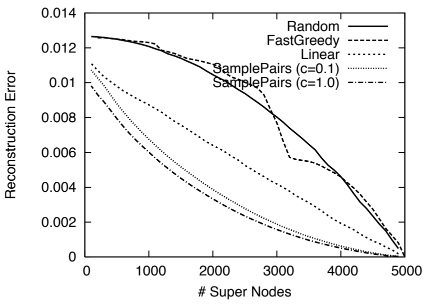


Figure 7: k-Gs, Italy(5000)

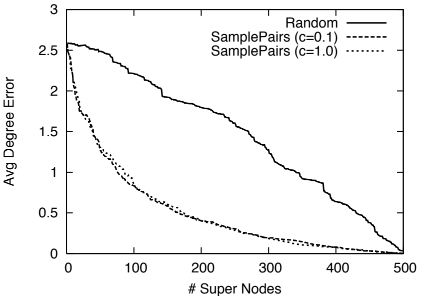


Figure 8: k-Gs, Authors(500)

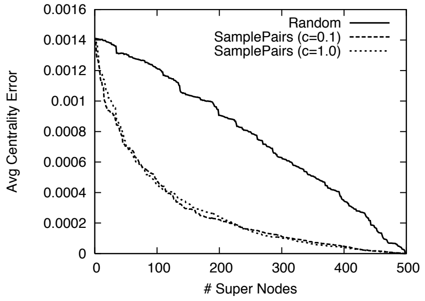


Figure 9: k-Gs, Authors(500)

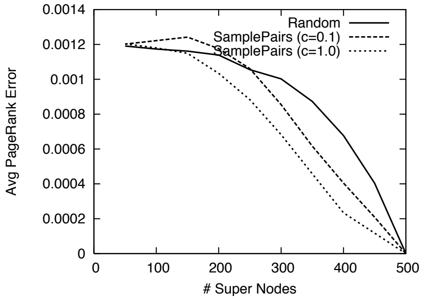


Figure 10: k-Gs, Authors(500)

It is also interesting to observe that LinearCheck and SamplePairs ($c = 1$) have the same time complexity. However, in all of our experiments, SamplePairs significantly outperformed LinearCheck. By not fixing either of the supernodes to participate in the merge, and instead selecting pairs at random, it appears that we achieve a better (though still incomplete) exploration of the space of possible merges during each iteration.

Finally, it is important to observe that our algorithms yield smaller reconstruction errors than FastGreedy, which indicates that it is important to consider the problem formulation based on reconstruction error in the case where the goal is accurate query answering.

6.2.2 Querying Graph Summaries

In addition to reconstruction error, we conducted additional experiments to understand the effects of summarization on other kinds of queries and analyses.

The first set of experiments measured the average absolute degree error across all nodes in the original graph $G(V, E)$. That is, if $d(v)$ and $\overline{d}(v)$ denote the original and expected responses, respectively, to the query requesting the degree of node v, then we measure average error as $\frac{1}{|V|}\sum_{v_i \in V} |d(v_i) - \overline{d}(v_i)|$. Figure 8 shows our results for Authors(500). Clearly, SamplePairs reduces the degree error substantially compared to Random. (We also obtained similar results for the Italy dataset, but they are omitted in the interest of space.)

The second set of experiments measured the average absolute error in centrality score, also across all nodes in $G(V, E)$. Figure 9 shows results for Authors(500); as expected, the results are proportional to those for degree error.

Finally, while we have a closed-form solution for computing the expected centrality score (Theorem 3.3), we also consider (empirically) computing the related PageRank score.[7] An intuitive heuristic for computing the expected PageRank score is to view the expected adjacency matrix as a weighted graph, and, conceptually, use this as input. Figure 10 shows that, despite the heuristic query evaluation semantics, use of clever summarization algorithms is effective in reducing the average PageRank error. (In Figure 10, we compute PageRank for $\beta = 0.85$.)

6.2.3 Graph Compressibility

Our final set of experiments is intended to understand the degree to which different kinds of graphs can be compressed using our summarization scheme (Section 2). For these experiments, we used synthetic data with varied generative models and parameters.

The first experiment used uniform graphs with varied density values. Intuitively, we expect that very sparse graphs and very dense graphs will be the most easily compressed. For example, consider a set of nodes

[7] PageRank [10] introduces an additional *damping factor* β. PageRank is equivalent to eigenvector centrality when $\beta = 1.0$.

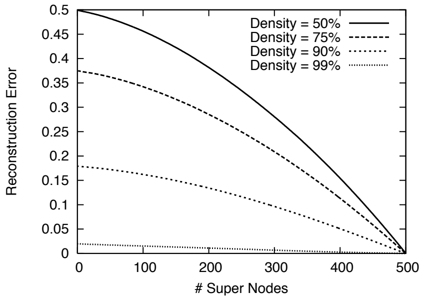


Figure 11: k-Gs, Uniform(500)

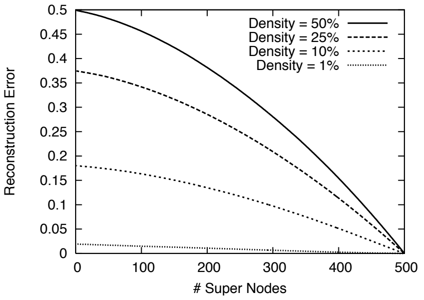


Figure 12: k-Gs, Uniform(500)

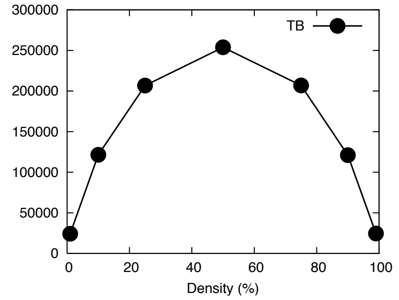


Figure 13: Gs, Uniform(500)

that constitute a connected component (or independent set). If these edges are grouped together in the same supernode, then there is no reconstruction error within this supernode. Similarly, consider the nodes in two supernodes s_1 and s_2. If every node in s_1 is connected to every node in s_2 (or every node in s_1 is not connected to any node in s_2), then there is no between-supernode reconstruction error.

We confirmed this intuition experimentally. For synthetic graphs consisting of 500 nodes each, Figures 11 and 12 show the reconstruction error for varied numbers of supernodes. Each of these lines was obtained using the SamplePairs algorithm ($c = 5.0$). Clearly, the very sparse (*density* $= 1\%$) and very dense (*density* $= 99\%$) graphs produce the least reconstruction error. (In fact, the symmetric pairs 1% and 10%; 25% and 75%; 10% and 90% were so similar that we had to display the results in two separate graphs.) In addition, Figure 13 shows, for varied density, the minimum value of TB (Equation 4.5), which measures the cost in bits of encoding the summary and the data given the summary. This also indicates that sparse and dense graphs are the most compressible.

Finally, Figure 14 shows results comparing Barabasi-Albert graphs and Uniform graphs. (Notice that Barabasi(5) and Uniform(1%), Barabasi(50) and Uniform(10%) each contain the same total number of edges.) Our empirical results indicate that the uniform graphs are somewhat more easily compressed.

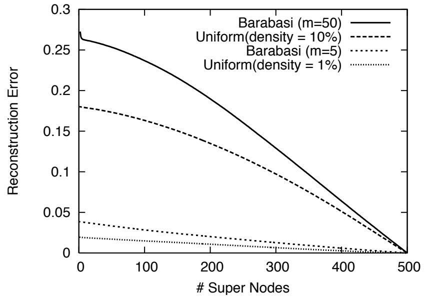


Figure 14: k-Gs, Barabasi-Albert(500)

6.3 Algorithms for k-CGs In this section, we evaluate our proposed algorithm for the k-CGs problem. This problem is inspired by the k-anonymity requirement for graphs. Our proposed algorithm – Condense – is described in Section 5.2.

To gage the effectiveness of our algorithm, we compared it to a strawman. In this case, the strawman iteratively chooses a random group of k nodes from G, and assigns these nodes to a single supernode. (At the end, the final supernode will contain between k and $2k - 1$ nodes from the original graph.) Figures 15 and 16 compare this strawman algorithm (Random) with the Condense algorithm using the Authors(500) and Italy(500) data sets, respectively. In each comparison, we use several different values of k, which is the required minimum occupancy for a supernode. Clearly, Condense outperforms Random.

7 Related work

Graph-partitioning and dense subgraph problems: Closely related to our work are problems associated with *graph partitioning*, where an input graph $G(V, E)$ must be partitioned into k disjoint parts, according to a certain optimization criterion. Several versions of this problem have been studied.[8] The use of eigenvectors for the purposes of partitioning was first introduced by Donath and Hoffman in [14] and has been studied extensively since. Although our problem seems similar to existing graph-partitioning problems, our optimization function (reconstruction error) is different from the standard optimization functions of previous work.

Given $G\,(V, E)$ the *densest subgraph* problem asks for a subset $V' \subseteq V$ of arbitrary size such that the vertex-induced subgraph has maximum average degree.

[8] See http://en.wikipedia.org/wiki/Graph_partitioning for a short overview of such problems and references

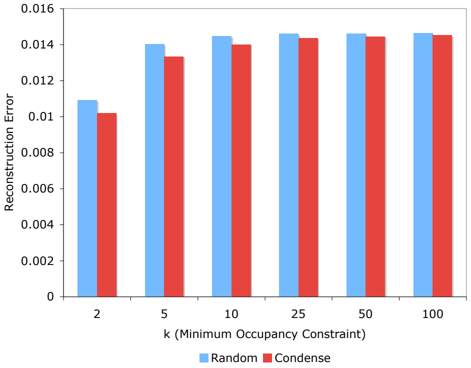


Figure 15: k-CGs, Authors(500)

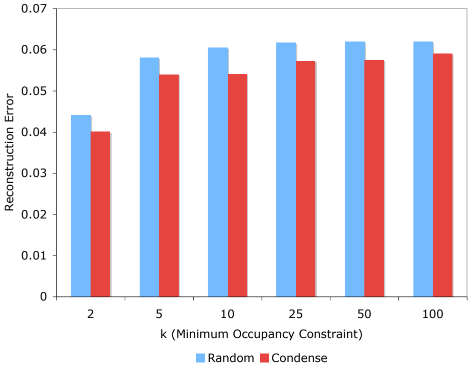


Figure 16: k-CGs, Italy(500)

This problem can be solved in polynomial time using flow techniques [17]. When the cardinality of the vertex set $|V'| \leq k$ is determined as part of the input, the problem is NP-hard [15]. The fundamental difference between this problem and our problem is that we want to *partition* the graph rather than find a single subgraph with maximum edge density.

Discovering communities: In practice, several instances of graph-partitioning and dense-subgraph problems have arisen in the context of social networks and the Web. The goal there has been to identify online communities. For example, a practical algorithm for finding large, dense subgraphs is presented in [19]. The algorithm is based on shingles and has been successfully applied for studying the subgraph structure of the Web. Algorithms for identifying communities have also been proposed in [16, 18]. This line of research is rather orthogonal to ours since either the communities identified by these algorithms do not define a partition of the nodes or the optimization function different. A recent line of research is related to a novel graph clustering index called *modularity* that has been recently proposed in [28]. Algorithms for optimizing modularity of a partitioning have been proposed in [12, 29, 33, 35].

Graph compression: The problem of *graph summarization* has also been considered recently [27, 32]. Although the motivation of [27] is also to create condensed representations of a graph using the MDL principle, our setting is rather different. We deal with arbitrary graphs, while [27] only compresses bipartite graphs. The objective function in [27] is also different from ours. Tian et al. [32] proposed two database-style aggregation operations for specifying graph summaries, but did not consider the problem of answering queries using these summaries. Thus, the objective function in [32] is also different from ours.

The most extensive literature for graph compression comes from the studies related to the Web graph (see for example [2, 9]: the goal in this work focuses on finding compact Web-graph representations to be used to compute PageRank [10] or other measures. Most of this work, however, focuses on reducing the number of bits required to encode a link between two pages and not on constructing graph summaries that can be used for structural discovery instead of the original graph.

Querying uncertain, incomplete and imprecise data: Finally, significant classical and recent work has focused on problems related to querying data that is uncertain, incomplete, or imprecise [1, 4, 6, 11, 13, 21, 34]. A common approach to handling this ambiguity is to define the semantics of important queries in terms of possible worlds [1, 6]. For example, work in probabilistic databases extends the relational data model by associating an inclusion probability with each tuple t, which is interpreted to mean that t appears in this proportion of true database instances. A big challenge is evaluating queries expressed in terms of this and related data models [13, 34]. Other recent work aims to capture data uncertainty and imprecision in the OLAP data model and evaluating aggregate queries [11].

8 Conclusion

In this paper, we proposed probabilistic semantics for answering structural queries atop "coarsened" graph summaries. Using this as motivation, we proposed algorithms for finding high-quality, space-efficient and privacy-preserving summaries.

There are several interesting opportunities for future work. In particular, we are considering various approaches for scaling our summarization algorithms to large datasets and techniques for incrementally maintaining a high-quality summary when new nodes are inserted into underlying graph G. In addition, we are considering extensions to our query answering semantics that incorporate attribute values in addition to simple graph topologies.

References

[1] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. In *SIGMOD*, 1987.

[2] M. Adler and M. Mitzenmacher. Towards compressing web graphs. In *Proceedings of the Data Compression Conference*, 2001.

[3] C. Aggarwal and P.Yu. A condensation approach to privacy-preserving data mining. In *EDBT*, 2004.

[4] L. Antova, C. Koch, and D. Olteanu. $(10)\hat{(}10)\hat{6}$ worlds and beyond: Efficient representation and processing of incomplete information. In *ICDE*, 2007.

[5] M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar aggregation in inconsistent databases. *Theoretical Computer Science*, 296(3), 2003.

[6] F. Bacchus, A. Grove, J. Halpern, and D. Kohler. From statistical knowledge bases to degrees of belief. *A.I.*, 87(1-2), 1996.

[7] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x? anonymized social networks, hidden patterns, and structural steganography. In *WWW*, 2007.

[8] A. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[9] P. Boldi and S. Vigna. The webgraph framework i: compression techniques. In *WWW*, 2004.

[10] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *WWW*, 1998.

[11] D. Burdick, P. Deshpande, T. Jayram, and R. Ramakrishnan. Olap over uncertain and imprecise data. In *VLDB*, 2005.

[12] A. Clauset, M. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review*, 70, 2004.

[13] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Proceedings of the 30th International Conference on Very Large Databases*, 2004.

[14] W.E. Donath and A.J. Hoffman. Algorithms for partitioning of graphs and computer logic based on eigenvectors of connection matrices. *IBM Technical Disclosure Bulletin*, 15, 1972.

[15] U. Feige, G. Kortsarz, and D. Peleg. The dense k-subgraph problem. *Algorithmica*, 29:2001, 1997.

[16] G.W. Flake, S. Lawrence, and C. L. Giles. Efficient identification of web communities. In *KDD*, 2000.

[17] G. Gallo, M.D. Grigoriadis, and R.E. Tarjan. A fast parametric maximum flow algorithm and applications. *Siam Journal on Computing*, 18:30–55, 1989.

[18] D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *Hypertext*, 1998.

[19] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB*, 2005.

[20] M. Hay, G. Miklau, D. Jensen, and P. Weis. Resisting structural re-identification in anonymized social networks. In *VLDB*, 2008.

[21] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.

[22] P. Kilpeläinen, H. Mannila, and E. Ukkonen. Mdl learning of unions of simple pattern languages from positive examples. In *EuroCOLT*, 1995.

[23] M. Koivisto, M. Perola, T. Varilo, et al. An MDL method for finding haplotype blocks and for estimating the strength of haplotype block boundaries. In *Pacific Symposium on Biocomputing*, 2003.

[24] M. Mehta, J. Rissanen, and R. Agrawal. Mdl-based decision tree pruning. In *KDD*, 1995.

[25] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[26] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *IEEE Symposium on Security and Privacy*, 2009.

[27] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *SIGMOD*, 2008.

[28] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review*, 69, 2004.

[29] M. Newman and M. Girvan. Algorithms for detecting community structure in networks. *Physical Review*, 71, 2005.

[30] P. Samarati. Protecting respondents' identities in microdata release. *Transactions on Knowledge and Data Engineering*, 2001.

[31] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[32] Y. Tian, R. Hankins, and J. Patel. Efficient aggregation for graph summarization. In *SIGMOD*, 2008.

[33] S. White and P. Smyth. A spectral clustering approach to finding communities in graph. In *SDM*, 2005.

[34] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, 2005.

[35] E. Ziv, M. Middenforf, and C. Wiggins. An information-theoretic approach to network modularity. *Physical Review*, 71, 2005.