Efficient Aggregation for Graph Summarization

Yuanyuan Tian
University of Michigan
Ann Arbor, MI, USA
ytian@eecs.umich.edu

Richard A. Hankins
Nokia Research Center
Palo Alto, CA, USA
rich.hankins@nokia.com

Jignesh M. Patel
University of Michigan
Ann Arbor, MI, USA
jignesh@eecs.umich.edu

ABSTRACT

Graphs are widely used to model real world objects and their relationships, and large graph datasets are common in many application domains. To understand the underlying characteristics of large graphs, graph summarization techniques are critical. However, existing graph summarization methods are mostly statistical (studying statistics such as degree distributions, hop-plots and clustering coefficients). These statistical methods are very useful, but the resolutions of the summaries are hard to control.

In this paper, we introduce two database-style operations to summarize graphs. Like the OLAP-style aggregation methods that allow users to drill-down or roll-up to control the resolution of summarization, our methods provide an analogous functionality for large graph datasets. The first operation, called *SNAP*, produces a summary graph by grouping nodes based on user-selected node attributes and relationships. The second operation, called *k-SNAP*, further allows users to control the resolutions of summaries and provides the "drill-down" and "roll-up" abilities to navigate through summaries with different resolutions. We propose an efficient algorithm to evaluate the *SNAP* operation. In addition, we prove that the *k-SNAP* computation is NP-complete. We propose two heuristic methods to approximate the *k-SNAP* results. Through extensive experiments on a variety of real and synthetic datasets, we demonstrate the effectiveness and efficiency of the proposed methods.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Query Processing; H.2.8 [**Database Applications**]: Data Mining

General Terms

Algorithms, Experimentation, Performance

Keywords

Graphs, Social Networks, Summarization, Aggregation

SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada.

1. INTRODUCTION

Graphs provide a powerful primitive for modeling data in a variety of applications. Nodes in graphs usually represent real world objects and edges indicate relationships between objects. Examples of data modeled as graphs include social networks, biological networks, and dynamic network traffic graphs. Often, nodes have attributes associated with them. For example, in Figure 1(a), a node representing a student may have attributes: gender and department. In addition, a graph may contain many different types of relationships, such as the friends and classmates relationships shown in Figure 1(a).

In many applications, graphs are very large, with thousands or even millions of nodes and edges. As a result, it is almost impossible to understand the information encoded in large graphs by mere visual inspection. Therefore, effective graph summarization methods are required to help users extract and understand the underlying information.

Most existing graph summarization methods use simple statistics to describe graph characteristics [6, 7, 13]; for example, researchers plot degree distributions to investigate the scale-free property of graphs, employ hop-plots to study the small world effect, and utilize clustering coefficients to measure the "clumpiness" of large graphs. While these methods are useful, the summaries contain limited information and can be difficult to interpret and manipulate. Methods that mine graphs for frequent patterns [11, 19, 20, 23] are also employed to understand the characteristics of large graphs. However, these algorithms often produce a large number of results that can easily overwhelm the user. Graph partitioning algorithms [14, 18, 22] have been used to detect community structures (dense subgraphs) in large networks. However, the community detection is based purely on nodes connectivities, and the attributes of nodes are largely ignored. Graph drawing techniques [3, 10] can help one better visualize graphs, but visualizing large graphs quickly becomes overwhelming.

What users need is a more controlled and intuitive method for summarizing graphs. The summarization method should allow users to freely choose the attributes and relationships that are of interest, and then make use of these features to produce small and informative summaries. Furthermore, users should be able to control the resolution of the resulting summaries and "drill-down" or "roll-up" the information, just like the OLAP-style aggregation methods in a traditional database systems.

In this paper, we propose two operations for graph summarization that fulfills these requirements. The first opera-

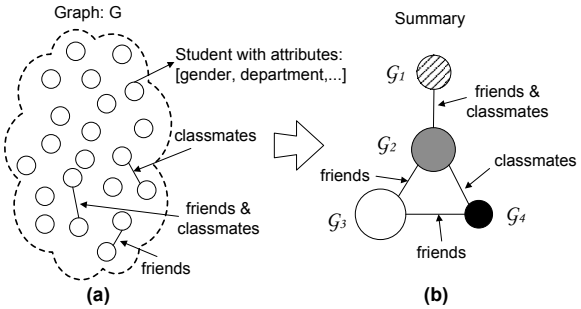

Figure 1: Graph Summarization by Aggregation

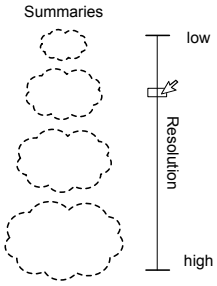

Figure 2: Illustration of Multi-resolution Summaries

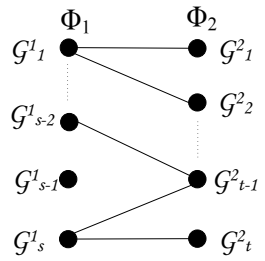

Figure 3: Construction of Φ_3 in the Proof of Theorem 2.4

tion, called *SNAP* (**S**ummarization by Grouping **N**odes on **A**ttributes and **P**airwise Relationships), produces a summary graph of the input graph by grouping nodes based on user-selected node attributes and relationships. Figure 1 illustrates the *SNAP* operation. Figure 1(a) is a graph about students (with attributes: gender, department and so on) and the relationships (classmates and friends) between them. Note that only few of the edges are shown in Figure 1(a). Based on user-selected gender and department attributes, and classmates and friends relationships, the *SNAP* operation produces a summary graph shown in Figure 1(b). This summary contains four groups of students and the relationships between these groups. Students in each group have the same gender and are in the same department, and they relate to students belonging to the same set of groups with friends and classmates relationships. For example, in Figure 1(b), each student in group $\mathcal{G}_1$ has at least a friend and a classmate in group $\mathcal{G}_2$. This compact summary reveals the underlying characteristics about the nodes and their relationships in the original graph.

The second operation, called *k-SNAP*, further allows users to control the resolutions of summaries. This operation is pictorially depicted in Figure 2. Here using the slider, a user can "drill-down" to a larger summary with more details or "roll-up" to a smaller summary with less details.

Our summarization methods have been applied to analyze real social networking applications. In one example, by summarizing the coauthorship graphs in database and AI communities, different coauthorship patterns across the two areas are displayed. In another application, interesting linking behaviors among liberal and conservative blogs are discovered by summarizing a large political blogs network.

The main contributions of this paper are:

(1) We introduce two database-style graph aggregation operations *SNAP* and *k-SNAP* for summarizing large graphs. We formally define the two operations, and prove that the *k-SNAP* computation is NP-complete.

(2) We propose an efficient algorithm to evaluate the *SNAP* operation, and also propose two heuristic methods (the top-down approach and the bottom-up approach) to approximately evaluate the *k-SNAP* operation.

(3) We apply our graph summarization methods to a variety of real and synthetic datasets. Through extensive experimental evaluation, we demonstrate that our methods produce meaningful summaries. We also show that the top-down approach is the ideal choice for *k-SNAP* evaluation in practice. In addition, the evaluation algorithms are very efficient even for very large graph datasets.

The remainder of this paper is organized as follows: Section 2 defines the *SNAP* and the *k-SNAP* operations. Section 3 introduces the evaluation algorithms for these operations. Experimental results are presented in Section 4. Section 5 describes related work, and Section 6 contains our concluding remarks.

2. GRAPH AGGREGATION OPERATIONS

In a graph, objects are represented by nodes, and relationships between objects are modeled as edges. In this paper, we support a general graph model, where objects (nodes) have associated attributes and different types of relationships (edges). Formally, we denote a graph G as (V, Υ) where V is the set of nodes, and $\Upsilon = \{E_1, E_2, ..., E_r\}$ is the set of edge types, with each $E_i \subseteq V \times V$ representing the set of edges of a particular type.

Nodes in a graph have a set of associated attributes, which is denoted as $\Lambda = \{a_1, a_2, ..., a_t\}$. Each node has a value for each attribute. These attributes are used to describe the features of the objects that the nodes represent. For example, in Figure 1(a), a node representing a student may have attributes that represent the student's gender and department. Different types of edges in a graph correspond to different types of relationships between nodes, such as friends and classmates relationships shown in Figure 1(a). Note that two nodes can be connected by different types of edges. For example, in Figure 1(a), two students can be classmates and friends at the same time.

For ease of presentation, we denote the set of nodes of graph G as $V(G)$, the set of attributes as $\Lambda(G)$, the actual value of attribute a_i for node v as $a_i(v)$, the set of edge types as $\Upsilon(G)$, and the set of edges of type E_i as $E_i(G)$. In addition, we denote the cardinality of a set S as $|S|$.

Our methods are applicable for both directed and undirected graphs. For ease of presentation, we only consider undirected graphs in this paper. Adaptations of our method for directed graphs are fairly straightforward, and omitted in the interest of space.

2.1 SNAP Operation

The *SNAP* operation produces a summary graph through a homogeneous grouping of the input graph's nodes, based on user-selected node attributes and relationships. We now formally define this operation.

To begin the formal definition of the *SNAP* operation, we first define the concept of node-grouping.

DEFINITION 2.1. ***(Node-Grouping of a Graph)*** *For a graph G, $\Phi = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_k\}$ is called a node-grouping of G,*

if and only if:
(1) $\forall \mathcal{G}_i \in \Phi, \mathcal{G}_i \subseteq V(G)$ *and* $\mathcal{G}_i \neq \emptyset$,
(2) $\bigcup_{\mathcal{G}_i \in \Phi} \mathcal{G}_i = V(G)$,
(3) for $\forall \mathcal{G}_i, \mathcal{G}_j \in \Phi$ *and* $(i \neq j)$, $\mathcal{G}_i \cap \mathcal{G}_j = \emptyset$.

Intuitively, a node-grouping partitions the nodes in a graph into non-overlapping subsets. Each subset $\mathcal{G}_i$ is called a group. When there is no ambiguity, we simply call a node-grouping a grouping. For a given grouping Φ of G, the group that node v belongs to is denoted as $\Phi(v)$. We further define the size of a grouping as the number of groups it contains.

Now, we define a partial order relation $\preccurlyeq$ on the set of all groupings of a graph.

DEFINITION 2.2. ***(Dominance Relation)*** *For a graph G, the grouping Φ dominates the grouping Φ', denoted as $\Phi' \preccurlyeq \Phi$, if and only if $\forall \mathcal{G}'_i \in \Phi'$, $\exists \mathcal{G}_j \in \Phi$ s.t. $\mathcal{G}'_i \subseteq \mathcal{G}_j$.*

It is easy to see that the dominance relation $\preccurlyeq$ is reflexive, anti-symmetric and transitive, hence it is a partial order relation. Next we define a special kind of grouping based on a set of user-selected attributes.

DEFINITION 2.3. ***(Attributes Compatible Grouping)*** *For a set of attributes $A \subseteq \Lambda(G)$, a grouping Φ is compatible with attributes A or simply A-compatible, if it satisfies the following: $\forall u, v \in V$, if $\Phi(u) = \Phi(v)$, then $\forall a_i \in A, a_i(u) = a_i(v)$.*

If a grouping Φ is compatible with A, we simply denote it as Φ_A. In each group of a A-compatible grouping, every node has exactly the same values for the set of attributes A. Note that there could be more than one grouping compatible with A. In fact a trivial grouping in which each node is a group is always compatible with any set of attributes.

Next, we prove that amongst all the A-compatible groupings of a graph, there is a global maximum grouping with respect to the dominance relation $\preccurlyeq$.

THEOREM 2.4. *In the set of all the A-compatible groupings of a graph G, denoted as S_A, $\exists \Phi_A \in S_A$, s.t. $\forall \Phi'_A \in S_A$, $\Phi'_A \preccurlyeq \Phi_A$.*

PROOF. We prove by contradiction. Assume that there is no global maximum A-compatible grouping, but more than one maximal grouping. Then, for every two of such maximal groupings Φ_1 and Φ_2, we will construct a new A-compatible grouping Φ_3 such that $\Phi_1 \preccurlyeq \Phi_3$ and $\Phi_2 \preccurlyeq \Phi_3$, which contradicts the assumption that Φ_1 and Φ_2 are maximal A-compatible groupings.

Assume that $\Phi_1 = \{\mathcal{G}_1^1, \mathcal{G}_2^1, ..., \mathcal{G}_s^1\}$ and $\Phi_2 = \{\mathcal{G}_1^2, \mathcal{G}_2^2, ..., \mathcal{G}_t^2\}$. We construct a bipartite graph on $\Phi_1 \cup \Phi_2$ as shown in Figure 3. The nodes in the bipartite graph are the groups from Φ_1 and Φ_2. And there is an edge between $\mathcal{G}_i^1 \in \Phi_1$ and $\mathcal{G}_j^2 \in \Phi_2$ if and only if $\mathcal{G}_i^1 \cap \mathcal{G}_j^2 \neq \emptyset$. After constructing the bipartite graph, we decompose this graph into connected components C_1, C_2, ..., C_m. For each connected component C_k, we union the groups inside this component and get a group $\cup(C_k)$. Now, we can construct a new grouping $\Phi_3 = \{\cup(C_1), \cup(C_2), ..., \cup(C_m)\}$. It is easy to see that $\Phi_1 \preccurlyeq \Phi_3$ and $\Phi_2 \preccurlyeq \Phi_3$. Now we prove that Φ_3 is compatible with A. From the definition of A-compatible groupings, if $\mathcal{G}_i^1 \cap \mathcal{G}_j^2 \neq \emptyset$, nodes in $\mathcal{G}_i^1 \cup \mathcal{G}_j^2$ all have the same attributes values. Therefore, every node in $\cup(C_k)$ has the same attributes values. Now, we have constructed a new A-compatible grouping Φ_3 such that $\Phi_1 \preccurlyeq \Phi_3$ and $\Phi_2 \preccurlyeq \Phi_3$. This contradicts our assumption that Φ_1 and Φ_2 are two different maximal A-compatible groupings. Therefore, there is a global maximum A-compatible grouping. □

We denote this global maximum A-compatible grouping as Φ_A^{max}. Φ_A^{max} is also the A-compatible grouping with the minimum cardinality. In fact, if we consider each node in a graph as a data record, then Φ_A^{max} is very much like the result of a group-by operation for these data records on the attributes A in the relational database systems.

The A-compatible groupings only account for the node attributes. However, nodes do not just have attributes, but also participate in pairwise relationships represented by the edges. Next, we consider relationships when grouping nodes.

For a grouping Φ, we denote the neighbor-groups of node v in E_i as $NeighborGroups_{\Phi,E_i}(v) = \{\Phi(u) | (u, v) \in E_i\}$.

Now we define groupings compatible with both node attributes and relationships.

DEFINITION 2.5. ***(Attributes and Relationships Compatible Grouping)*** *For a set of attributes $A \subseteq \Lambda(G)$ and a set of relationship types $R \subseteq \Upsilon(G)$, a grouping Φ is compatible with attributes A and relationship types R or simply (A, R)-compatible, if it satisfies the following:*
(1) Φ is A-compatible,
(2) $\forall u, v \in V(G)$, if $\Phi(u) = \Phi(v)$, then $\forall E_i \in R$, $NeighborGroups_{\Phi,E_i}(u) = NeighborGroups_{\Phi,E_i}(v)$.

If a grouping Φ is compatible with A and R, we also denote it as $\Phi_{(A,R)}$. In each group of an (A, R)-compatible grouping, all the nodes are homogeneous in terms of both attributes A and relationships in R. In other words, every node inside a group has exactly the same values for attributes A, and is adjacent to nodes in the same set of groups for all the relationships in R.

As an example, assume that the summary in Figure 1(b) is a grouping compatible with gender and department attributes, and classmates and friends relationships. Then, for example, every student (node) in group $\mathcal{G}_2$, has the same gender and department attributes values, and is a friend of some student(s) in $\mathcal{G}_3$, a classmate of some student(s) in $\mathcal{G}_4$, and a friend to some student(s) as well as a classmate to some student(s) in $\mathcal{G}_1$.

Given a grouping $\Phi_{(A,R)}$, we can infer relationships between groups from the relationships between nodes in R. For each edge type $E_i \in R$, we define the corresponding group relationships as $E_i(G, \Phi_{(A,R)}) = \{(\mathcal{G}_i, \mathcal{G}_j) |\ \mathcal{G}_i, \mathcal{G}_i \in \Phi_{(A,R)}$ and $\exists u \in \mathcal{G}_i, v \in \mathcal{G}_j$ s.t. $(u, v) \in E_i\}$. In fact, by the definition of (A, R)-compatible groupings, if there is one node in a group adjacent to some node(s) in the other group, then every node in the first group is adjacent to some node(s) in the second.

Similarly to attributes compatible groupings, there could be more than one grouping compatible with the given attributes and relationships. The grouping in which each node forms a group is always compatible with any given attributes and relationships.

Next we prove that among all the (A, R)-compatible groupings there is a global maximum grouping with respect to the dominance relation $\preccurlyeq$.

THEOREM 2.6. *In the set of all the (A, R)-compatible groupings of a graph G, denoted as $S_{(A,R)}$, $\exists \Phi_{(A,R)} \in S_{(A,R)}$, s.t. $\forall \Phi'_{(A,R)} \in S_{(A,R)}$, $\Phi'_{(A,R)} \preccurlyeq \Phi_{(A,R)}$.*

PROOF. Again we prove by contradiction. Assume that there is no global maximum (A, R)-compatible grouping, but more than one maximal grouping. Then, for every two of such maximal groupings Φ_1 and Φ_2, we use the same con-

struction method to construct Φ_3 as in the proof of Theorem 2.4. We already know that Φ_3 is A-compatible, $\Phi_1 \preccurlyeq \Phi_3$ and $\Phi_2 \preccurlyeq \Phi_3$. Using similar arguments as in Theorem 2.4, we can also prove that Φ_3 is compatible with R. This contradicts our assumption that Φ_1 and Φ_2 are two different maximal (A, R)-compatible groupings.

From the construction of Φ_3, we know that if $\mathcal{G}_i^1 \cap \mathcal{G}_j^2 \neq \emptyset$, then the nodes in $\mathcal{G}_i^1 \cup \mathcal{G}_j^2$ belong to the same group in Φ_3. Next, we prove that every node in $\mathcal{G}_i^1 \cup \mathcal{G}_j^2$ is also adjacent to nodes in the same set of groups in Φ_3.

Again we prove by contradiction. Assume that there are two nodes $u, v \in \mathcal{G}_i^1 \cup \mathcal{G}_j^2$, u is adjacent to $\cup(C_k)$ in Φ_3 but v is not. First, if both $u, v \in \mathcal{G}_i^1$ or both $u, v \in \mathcal{G}_j^2$, then as both Φ_1 and Φ_2 are (A, R)-compatible groupings, and the construction of Φ_3 does not decompose any groups in Φ_1 or Φ_2, u, v should always be adjacent to the same set of groups in Φ_3. This contradicts our assumption. Second, the two nodes can come from different groupings. For simplicity, assume $u \in \mathcal{G}_i^1$ and $v \in \mathcal{G}_j^2$. As $\mathcal{G}_i^1 \cap \mathcal{G}_j^2 \neq \emptyset$, a node $w \in \mathcal{G}_i^1 \cap \mathcal{G}_j^2$ is adjacent to the same set of groups as u in Φ_1 and adjacent to the same set of groups as v in Φ_2. As a result, every group that u is adjacent to in Φ_1 should intersect with some group that v is adjacent to in Φ_2. Since u is adjacent to $\cup(C_k)$, then u must be adjacent to at least one group in Φ_1 that is later merged to $\cup(C_k)$. This group should also intersect with a group $\mathcal{G}_l^2$ in Φ_2 that v is adjacent to. Then, by the construction algorithm of Φ_3, $\mathcal{G}_l^2$ should belong to the connected component C_k, thus should be later merged in $\cup(C_k)$. As a result, v is also adjacent to $\cup(C_k)$ in Φ_3, which contradicts our assumption.

Now we know if $\mathcal{G}_i \cap \mathcal{G}_j \neq \emptyset$, nodes in $\mathcal{G}_i \cup \mathcal{G}_j$ are all adjacent to the same set of groups in Φ_3. In each C_k, $\forall \mathcal{G}_i \in C_k$, $\exists \mathcal{G}_j \in C_k$ such that $\mathcal{G}_i \cap \mathcal{G}_j \neq \emptyset$. As a result, every node in $\cup(C_k)$ is adjacent to the same set of groups in Φ_3.

We have constructed a new (A, R)-compatible grouping Φ_3 such that $\Phi_1 \preccurlyeq \Phi_3$ and $\Phi_2 \preccurlyeq \Phi_3$. This contradicts the fact that Φ_1 and Φ_2 are two different maximal (A, R)-compatible groupings. Therefore, there is a global maximum (A, R)-compatible grouping. □

We denote the global maximum (A, R)-compatible grouping as $\Phi_{(A,R)}^{max}$. $\Phi_{(A,R)}^{max}$ is also the (A, R)-compatible grouping with the minimum cardinality. Due to its compactness, this maximum grouping is more useful than other (A, R)-compatible groupings.

Now, we define our first operation for graph summarization, namely *SNAP*.

DEFINITION 2.7. ***(SNAP Operation)*** *The SNAP operation takes as input a graph G, a set of attributes $A \subseteq \Lambda(G)$, and a set of edge types $R \subseteq \Upsilon(G)$, and produces a summary graph G_{snap}, where $V(G_{snap}) = \Phi_{(A,R)}^{max}$, and $\Upsilon(G_{snap}) = \{E_i(G, \Phi_{(A,R)}^{max}) | E_i \in R\}$.*

Intuitively, the *SNAP* operation produces a summary graph of the input graph based on user-selected attributes and relationships. The nodes of this summary graph correspond to the groups in the maximum (A, R)-compatible grouping. And the edges of this summary graph are the group relationships inferred from the node relationships in R.

2.2 k-SNAP Operation

The *SNAP* operation produces a grouping in which nodes of each group are homogeneous with respect to user-selected attributes and relationships. Unfortunately, homogeneity is often too restrictive in practice, as most real life graph data is subject to noise and uncertainty; for example, some edges may be missing because of the failure in the detection process, and some edges may be spurious because of errors. Applying the *SNAP* operation on noisy data can result in a large number of small groups, and, in the worst case, each node may end up an individual group. Such a large summary is not very useful in practice. A better alternative is to let users control the sizes of the results to get summaries with the resolutions that they can manage (as shown in Figure 2). Therefore, we introduce a second operation, called *k-SNAP*, which relaxes the homogeneity requirement for the relationships and allows users to control the sizes of the summaries.

The relaxation of the homogeneity requirement for the relationships is based on the following observation. For each pair of groups in the result of the *SNAP* operation, if there is a group relationship between the two, then every node in both groups participates in this group relationship. In other words, every node in one group relates to some node(s) in the other group. On the other hand, if there is no group relationship between two groups, then absolutely no relationship connects any nodes across the two groups. However, in reality, if most (not all) nodes in the two groups participate in the group relationship, it is often a good indication of a strong relationship between the two groups. Likewise, it is intuitive to mark two groups as being weakly related if only a tiny fraction of nodes are connected between these groups.

Based on these observations, we relax the homogeneity requirement for the relationships by not requiring that every node participates in a group relationship. But we still maintain the homogeneity requirement for the attributes, i.e. all the groupings should be compatible with the given attributes. Users control how many groups are present in the summary by specifying the required number of groups, denoted as k. There are many different groupings of size k compatible with the attributes, thus we need to measure the qualities of the different groupings. We propose the Δ-measure to assess the quality of an A-compatible grouping by examining how different it is to a hypothetical (A, R)-compatible grouping.

We first define the set of nodes in group $\mathcal{G}_i$ that participate in a group relationship $(\mathcal{G}_i, \mathcal{G}_j)$ of type E_t as $P_{E_t,\mathcal{G}_j}(\mathcal{G}_i) = \{u | u \in \mathcal{G}_i \text{ and } \exists v \in \mathcal{G}_j \text{ s.t. } (u, v) \in E_t\}$. Then we define the participation ratio of the group relationship $(\mathcal{G}_i, \mathcal{G}_j)$ of type E_t as $p_{i,j}^t = \frac{|P_{E_t,\mathcal{G}_j}(\mathcal{G}_i)| + |P_{E_t,\mathcal{G}_i}(\mathcal{G}_j)|}{|\mathcal{G}_i| + |\mathcal{G}_j|}$. For a group relationship, if its participation ratio is greater than 50%, we call it a strong group relationship, otherwise, we call it a weak group relationship. Note that in an (A, R)-compatible grouping, the participation ratios are either 0% or 100%.

Given a graph G, a set of attributes A and a set of relationship types R, the Δ-measure of $\Phi_A = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_k\}$ is defined as follows:

$$\Delta(\Phi_A) = \sum_{\mathcal{G}_i, \mathcal{G}_j \in \Phi_A} \sum_{E_t \in R} (\delta_{E_t,\mathcal{G}_j}(\mathcal{G}_i) + \delta_{E_t,\mathcal{G}_i}(\mathcal{G}_j)) \quad (1)$$

$$\delta_{E_t,\mathcal{G}_j}(\mathcal{G}_i) = \begin{cases} |P_{E_t,\mathcal{G}_j}(\mathcal{G}_i)| & \text{if } p_{i,j}^t \leq 0.5 \\ |\mathcal{G}_i| - |P_{E_t,\mathcal{G}_j}(\mathcal{G}_i)| & \text{otherwise} \end{cases} \quad (2)$$

Intuitively, the Δ-measure counts the minimum number of differences in participations of group relationships between

the given A-compatible grouping and a hypothetical (A, R)-compatible grouping of the same size. The measure looks at each pairwise group relationship: If this group relationship is weak ($p^t_{i,k} \leq 0.5$), then it counts the participation differences between this weak relationship and a non-relationship ($p^t_{i,k} = 0$); on the other hand, if the group relationship is strong, it counts the differences between this strong relationship and a 100% participation-ratio group relationship. The δ function, defined in Equation 2, evaluates the part of the Δ value contributed by a group $\mathcal{G}_i$ with one of its neighbors $\mathcal{G}_j$ in a group relationship of type E_t.

It is easy to prove that $\Delta(\Phi_A) \geq 0$. The smaller $\Delta(\Phi_A)$ value is, the more closer Φ_A is to a hypothetical (A, R)-compatible grouping. $\Delta(\Phi_A) = 0$ if and only if Φ_A is (A, R)-compatible. We can also prove that $\Delta(\Phi_A)$ is bounded by $2|\Phi_A||V||R|$, as each $\delta_{E_t,\mathcal{G}_j}(\mathcal{G}_i) \leq |\mathcal{G}_i|$.

Now we will formally define the *k-SNAP* operation.

DEFINITION 2.8. ***(k-SNAP Operation)*** *The k-SNAP operation takes as input a graph G, a set of attributes $A \subseteq \Lambda(G)$, a set of edge types $R \subseteq \Upsilon(G)$ and the desired number of groups k, and produces a summary graph $G_{k\text{-}snap}$, where $V(G_{k\text{-}snap}) = \Phi_A$, s.t. $|\Phi_A| = k$ and $\Phi_A = arg\min_{\Phi'_A}\{\Delta(\Phi'_A)\}$, and $\Upsilon(G_{k\text{-}snap}) = \{E_i(G, \Phi_A) \mid E_i \in R\}$.*

Given the desired number of groups k, the *k-SNAP* operation produces an A-compatible grouping with the minimum Δ value. Unfortunately, as we prove below, this optimization problem is NP-complete. To prove this, we first formally define the decision problem associated with this optimization problem and then prove it to be NP-complete.

THEOREM 2.9. *Given a graph G, a set of attributes A, a set of relationship types R, a user-specified number of groups k ($|\Phi^{max}_A| \leq k \leq |V(G)|$), and a real number D ($0 \leq D < 2k|V||R|$), the problem of finding an A-compatible grouping Φ_A of size k with $\Delta(\Phi_A) \leq D$ is NP-complete.*

PROOF. We use proof by restriction to prove the NP-completeness of this problem.

(1) This problem is in NP, because a nondeterministic algorithm only needs to guess an A-compatible grouping Φ_A of size k and check in polynomial time that $\Delta(\Phi_A) \leq D$. And an A-compatible grouping Φ_A of size k can be generated by a polynomial time algorithm.

(2) This problem contains a known NP-complete problem 2-Role Assignability (2RA) [16] as a special case. By restricting $A = \emptyset$, $|R| = 1$, $k = 2$ and $D = 0$, this problem becomes 2RA (which decides whether the nodes in a graph can be assigned with 2 roles, each node with one of the roles, such that if two nodes are assigned with the same role, then the sets of roles assigned to their neighbors are the same.) As proved in [16], 2RA is NP-complete. □

Given the NP-completeness, it is infeasible to find the exact optimal answers for the *k-SNAP* operation. Therefore, we propose two heuristic algorithms to evaluate the *k-SNAP* operation approximately.

3. EVALUATION ALGORITHMS

In this section, we introduce the evaluation algorithms for *SNAP* and *k-SNAP*. It is computationally feasible to exactly evaluate the *SNAP* operation, hence the proposed evaluation algorithm produces exact summaries. In contrast, *k-SNAP* computation was proved to be NP-complete and, therefore, we propose two *heuristic* algorithms for efficiently approximating the solution. Before discussing the details of the algorithms, we first introduce the evaluation architecture and the data structures used for the algorithms. Note that, for ease of presentation, all algorithms discussed in this section are assumed to work on one type of relationship; extending these algorithms for multiple relationship types is straightforward, hence is omitted in the interest of space.

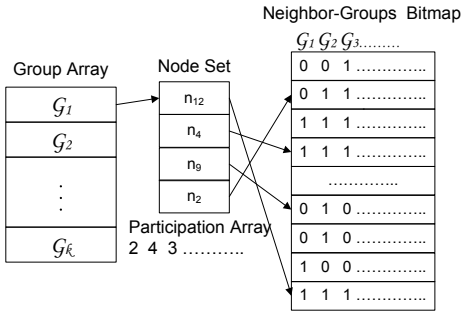

Figure 4: Data Structures Used in the Evaluation Algorithms

3.1 Architecture and Data Structures

All the evaluation algorithms employ an architecture as follows. The input graphs reside in the database on disk. A chunk of memory is allocated as a buffer pool for the database. It is used to buffer actively used content from disk to speed up the accesses. Every access of the evaluation algorithms to the nodes and edges of graphs goes through the buffer pool. If the content is buffered, then the evaluation algorithms simply retrieve the content from the buffer pool; otherwise, the content is read from disk into the buffer pool first. Another chunk of memory is allocated for the evaluation algorithms as the working memory, similar to the working memory space used by traditional database algorithms such as hash join. This working memory is used to hold the data structures used in the evaluation algorithms.

The evaluation algorithms share some common data structures as shown in Figure 4. The group-array data structure keeps track of the groups in the current grouping. Each entry in groups-array stores the id of a group and also points to a node-set, which contains the nodes in the corresponding group. Each node in the node-set points to one row of the neighbor-groups bitmap. This bitmap is the most memory consuming data structure in the evaluation algorithms. Each row of the bitmap corresponds to a node, and the bits in the row store the node's neighbor-groups. If bit position i is 1, then we know that this node has at least one neighbor belonging to group $\mathcal{G}_i$ with id i; otherwise, this node has no neighbor in group $\mathcal{G}_i$. For each group $\mathcal{G}_i$ in the current grouping, we also keep a participation-array which stores the participation counts $|P_{E,\mathcal{G}_j}(\mathcal{G}_i)|$ for each neighbor group. Note that the participation-array of a group can be inferred from the nodes' corresponding rows in the neighbor-groups bitmap. For example, in Figure 4, the participation-array of group $\mathcal{G}_1$ can be computed by counting the number of 1s in each column of the bitmap rows corresponding to n_{12}, n_4, n_9 and n_2. All the data structures shown in Figure 4 change dynamically during the evaluation algorithms. An increase in the number of groups leads to the growth of the group-array size, which also results in an increase of the width of

Algorithm 1 ***SNAP***$(G,\ A,\ R)$

Input: G: a graph; $A \subseteq \Lambda(G)$: a set of attributes; $R = \{E\} \subseteq \Upsilon(G)$: a set containing one relationship type E
Output: A summary graph.
1: Compute the maximum A-compatible grouping by sorting nodes in G based on values of attributes A
2: Initialize the data structures
3: **while** there is a group $\mathcal{G}_i$ with participation array containing values other than 0 or $|\mathcal{G}_i|$ **do**
4: Divide $\mathcal{G}_i$ into subgroups by sorting nodes in $\mathcal{G}_i$ based on their corresponding rows in the bitmap
5: Update the data structures
6: **end while**
7: Form the summary graph G_{snap}
8: **return** G_{snap}

Algorithm 2 ***k-SNAP-Top-Down***$(G,\ A,\ R,\ k)$

Input: G: a graph; $A \subseteq \Lambda(G)$: a set of attributes; $R = \{E\} \subseteq \Upsilon(G)$: a set containing one relationship type E; k: the required number of groups in the summary
Output: A summary graph.
1: Compute the maximum A-compatible grouping by sorting nodes in G based on values of attributes A
2: Initialize the data structures and let Φ_c denote the current grouping
3: ***SplitGroups***$(G,\ A,\ R,\ k,\ \Phi_c)$
4: Form the summary graph $G_{k\text{-}snap}$
5: **return** $G_{k\text{-}snap}$

the bitmap, as well as the sizes of the participation-arrays. The set of nodes for a group also change dynamically.

For most of this paper, we will assume that all the data structures needed by the evaluation algorithms can fit in the working memory. This is often a reasonable assumption in practice for a majority of graph datasets. However, we also consider the case when this memory assumption does not hold (see Section 4.4.3).

3.2 Evaluating SNAP Operation

In this section, we introduce the evaluation algorithm for the *SNAP* operation. This algorithm also serves as a foundation for the two *k-SNAP* evaluation algorithms.

The *SNAP* operation tries to find the maximum (A, R)-compatible grouping for a graph, a set of nodes attributes, and the specified relationship type. The evaluation algorithm starts from the maximum A-compatible grouping, and iteratively splits groups in the current grouping, until the grouping is also compatible with the relationships.

The algorithm for evaluating the *SNAP* operation is shown in Algorithm 1. In the first step, the algorithm groups the nodes based only on the attributes by a sorting on the attributes values. Then the data structures are initialized by this maximum A-compatible grouping. Note that if a grouping is compatible with the relationships, then all nodes inside a group should have the same set of neighbor-groups, which means that they have the same values in their rows of the bitmap. In addition, the participation array of each group should then only contain the values 0 or the size of the group. This criterion has been used as the terminating condition to check whether the current grouping is compatible with the relationships in line 3 of Algorithm 1. If there exists a group whose participation array contains values other than 0 or the size of this group, the nodes in this group are not homogeneous in terms of the relationships. We can split this group into subgroups, each of which contains nodes with the same set of neighbor-groups. This can be achieved by sorting the nodes based on their corresponding entries in the bitmap. (The radix sort is a perfect candidate for this task.) After this division, new groups are introduced. One of them continues to use the same group id of the split group, and the remaining groups are added to the end of the group-array. Accordingly, each row of the bitmap has to be widened. The nodes of this split group are distributed among the new groups. As the group memberships of these nodes are changed, the bitmap entries for them and their neighbor nodes have to be updated. Then the algorithm goes to the next iteration. This process continues until the condition in line 3 does not hold anymore.

It can be easily verified that the grouping produced by Algorithm 1 is the maximum (A, R)-compatible grouping. The algorithm starts from the maximum A-compatible grouping, and it only splits existing groups, so the grouping after each iteration is guaranteed to be A-compatible. In addition, each time we split a group, we always keep nodes with same neighbor-groups together. Therefore, when the algorithm stops, the grouping should be the maximum (A, R)-compatible grouping.

After we get the maximum (A, R)-compatible grouping, we can construct the summary graph. The nodes in the summary graph corresponds to the groups in the result grouping. The edges in the summary graph are the group relationships inferred from the node relationships in the original graph.

Now we will analyze the complexity of this evaluation algorithm. Sorting by the attributes values takes $O(|V| \log |V|)$ time, assuming the number of attributes is a small constant. The initialization of the data structures takes $O(|E|)$ time, where E is the only edge type in R (for simplicity, we only consider one edge type in our algorithms). At each iteration of the while loop, the radix sort takes $O(k_i|\mathcal{G}_i|)$ time, where k_i is the number of groups in the current grouping and $\mathcal{G}_i$ is the group to be split. Updating the data structures takes $|Edges(\mathcal{G}_i)|$, where $Edges(\mathcal{G}_i)$ is the set of edges adjacent to nodes in $\mathcal{G}_i$. Note that k_i is monotonically increasing, and that the number of iterations is less than the size of the final grouping, denoted as k. Therefore, the complexity for all the iterations is bounded by $O(k^2|V| + k|E|)$. Constructing the summary takes $O(k^2)$ time. To sum up, the upper-bound complexity of the *SNAP* algorithm is $O(|V| \log |V| + k^2|V| + k|E|)$.

As the evaluation algorithm takes inputs from the graph database on disk, we also need to analyze the number of disk accesses to the database. We assume all the accesses to the database are in the units of pages. For simplicity, we do not distinguish whether an access is a disk page access or a buffer pool page access. We assume that all the nodes information of the input graph takes $\|V\|$ pages in the database, and all the edges information takes $\|E\|$ pages. Then the *SNAP* operation incurs $\|V\|$ page accesses to read all the nodes with their attributes, $\|E\|$ page accesses to initialize the data structures, and at most $\|E\|$ page accesses each time it updates the data structures. So, the total number of page accesses is bounded by $\|V\| + (k+1)\|E\|$. Note that in

Algorithm 3 ***SplitGroups***(G, A, R, k, Φ_c)

Input: G: a graph; $A \subseteq \Lambda(G)$: a set of attributes; $R = \{E\} \subseteq \Upsilon(G)$: a set containing one relationship type E; k: the required number of groups in the summary; Φ_c: the current grouping.
Output: Splitting groups in Φ_c until $|\Phi_c| = k$.
1: Build a heap on the CT value of each group in Φ_c
2: **while** $|\Phi_c| < k$ **do**
3: Pop the group $\mathcal{G}_i$ with the maximum CT value from the heap
4: Split $\mathcal{G}_i$ into two based on the neighbor group $\mathcal{G}_t = arg\max_{\mathcal{G}_j}\{\delta_{E,\mathcal{G}_j}(\mathcal{G}_i)\}$
5: Update data structures (Φ_c is updated)
6: Update the heap
7: **end while**

practice, not every page access results in an actual disk IO. Especially for the updates of the data structures discussed in Section 3.1, most of the edges information will be cached in the buffer pool.

3.3 Evaluating k-SNAP Operation

The *k-SNAP* operation allows a user to choose k, the number of groups that are shown in the summary. For a given graph, a set of nodes attributes A and the set of relationship types R, a meaningful k value should fall in the range between $|\Phi_A^{max}|$ and $|\Phi_{(A,R)}^{max}|$. However, if the user input is beyond the meaningful range, i.e. $k < |\Phi_A^{max}|$ or $k > |\Phi_{(A,R)}^{max}|$, then the evaluation algorithms will return the summary corresponding to Φ_A^{max} or $\Phi_{(A,R)}^{max}$, respectively. For simplicity, we will assume that the k values input to the algorithms are always meaningful. By varying the k values, users can produce multi-resolution summaries. A larger k value corresponds to a higher resolution summary. The finest summary corresponds to the grouping $\Phi_{(A,R)}^{max}$; and the coarsest summary corresponds to the grouping Φ_A^{max}.

As proved in Section 2.2, computing the exact answers for the *k-SNAP* operation is NP-complete. In this paper, we propose two heuristic algorithms to approximate the answers. The top-down approach starts from the maximum grouping only based on attributes, and iteratively splits groups until the number of groups reaches k. The other approach employs a bottom-up scheme. This method first computes the maximum grouping compatible with both attributes and relationships, and then iteratively merges groups until the result satisfies the user defined k value. In both approaches, we apply the same principle: nodes of a same group in the maximum (A, R)-compatible grouping should always remain in a same group even in coarser summaries. We call this principle KEAT (**K**eep the **E**quivalent **A**lways **T**ogether) principle. This principle guarantees that when $k = |\Phi_{(A,R)}^{max}|$, the result produced by the *k-SNAP* evaluation algorithms is the same as the result of the *SNAP* operation with the same inputs.

3.3.1 Top-Down Approach

Similar to the *SNAP* evaluation algorithm, the top-down approach (see Algorithm 2) also starts from the maximum grouping based only on attributes, and then iteratively splits existing groups until the number of groups reaches k. However, in contrast to the *SNAP* evaluation algorithm, which randomly chooses a splittable group and splits it into subgroups based on its bitmap entries, the top-down approach has to make the following decisions at each iterative step: (1) which group to split and (2) how to split it. Such decisions are critical as once a group is split, the next step will operate on the new grouping. At each step, we can only make the decision based on the current grouping. We want each step to make the smallest move possible, to avoid going too far away from the right direction. Therefore, we split one group into only two subgroups at each iterative step. There are different ways to split one group into two. One natural way is to divide the group based on whether nodes have relationships with nodes in a neighbor group. After the split, nodes in the two new groups either all or never participate in the group relationships with this neighbor group. This way of splitting groups also ensures that the resulting groups follow the KEAT principle.

Algorithm 4 ***k-SNAP-Bottom-Up***(G, A, R, k)

Input: G: a graph; $A \subseteq \Lambda(G)$: a set of attributes; $R = \{E\} \subseteq \Upsilon(G)$: a set containing one relationship type E; k: the required number of groups in the summary.
Output: A summary graph.
1: G_{snap}=***SNAP***(G, A, R)
2: Initialize the data structures using the grouping in G_{snap} and let Φ_c denote the current grouping
3: ***MergeGroups***(G, A, R, k, Φ_c)
4: Form the summary graph $G_{k\text{-}snap}$
5: **return** $G_{k\text{-}snap}$

Now, we introduce the heuristic for deciding which group to split and how to split at each iterative step. As defined in Section 2.2, the *k-SNAP* operation tries to find the grouping with a minimum Δ measure (see Equation 1) for a given k. The computation of the Δ measure can be broken down into each group with each of its neighbors (see the δ function in Equation 2). Therefore, our heuristic chooses the group that makes the most contribution to the Δ value with one of its neighbor groups. More formally, for each group $\mathcal{G}_i$, we define $CT(\mathcal{G}_i)$ as follows:

$$CT(\mathcal{G}_i) = \max_{\mathcal{G}_j}\{\delta_{E,\mathcal{G}_j}(\mathcal{G}_i)\} \quad (3)$$

Then, at each iterative step, we always choose the group with the maximum CT value to split and then split it based on whether nodes in this group $\mathcal{G}_i$ have relationships with nodes in its neighbor group $\mathcal{G}_t$, where

$$\mathcal{G}_t = arg\max_{\mathcal{G}_j}\{\delta_{E,\mathcal{G}_j}(\mathcal{G}_i)\}$$

As shown in Algorithm 3, to speed up the decision process, we build a heap on the CT values of groups. At each iteration, we pop the group with the maximum CT value to split. At the end of each iteration, we update the heap elements corresponding to the neighbors of the split group, and insert elements corresponding to the two new groups.

The time complexity of the top-down approach is similar to the *SNAP* algorithm, except that it takes $O(k_0^2 + k_0)$ time to compute the CT values and build the heap, and at most $O(k_i^2 + k_i \log k_i)$ time to update the heap at each iteration, where k_0 is the number of groups in the maximum A-compatible grouping, and k_i is the number of groups at each iteration. As $k < |V|$, the upper-bound complexity of the top-down approach is still $O(|V| \log |V| + k^2|V| + k|E|)$.

Algorithm 5 ***MergeGroups***$(G,\ A,\ R,\ k,\ \Phi_c)$

Input: G: a graph; $A \subseteq \Lambda(G)$: a set of attributes; $R = \{E\} \subseteq \Upsilon(G)$: a set containing one relationship type E; k: the required number of groups in the summary; Φ_c: the current grouping.
Output: Merging groups in Φ_c until $|\Phi_c| = k$.
1: Build a heap on ($MergeDist$, $Agree$, $MinSize$) for pairs of groups
2: **while** $|\Phi_c| > k$ **do**
3: Pop the pair of groups with the best ($MergeDist$, $Agree$, $MinSize$) value from the heap
4: Merge the two groups into one
5: Update data structures (Φ_c is updated)
6: Update the heap
7: **end while**

Following the same method of analyzing the page accesses for the $SNAP$ algorithm, the number of page accesses incurred by the top-down approach is bounded by $\|V\| + (k+1)\|E\|$.

3.3.2 Bottom-Up Approach

The bottom-up approach first computes the maximum (A, R)-compatible grouping using Algorithm 1, and then iteratively merges two groups until grouping size is k (see Algorithm 4). Choosing which two groups to merge in each iterative step is crucial for the bottom-up approach. First, the two groups are required to have the same attributes values. Second, the two groups must have similar group relationships with other groups. Now, we formally define this similarity between two groups.

The two groups to be merged should have similar neighbor groups with similar participation ratios. We define a measure called $MergeDist$ to assess the similarity between two groups in the merging process.

$$MergeDist(\mathcal{G}_i, \mathcal{G}_j) = \sum_{k \neq i,j} |p_{i,k} - p_{j,k}| \quad (4)$$

$MergeDist$ accumulates the differences in participation ratios between $\mathcal{G}_i$ and $\mathcal{G}_j$ with other groups. The smaller this value is, the more similar the two groups are.

If two pairs of groups have the same $MergeDist$, we need to further distinguish which pair is "more similar". We look at each common neighbor $\mathcal{G}_k$ of $\mathcal{G}_i$ and $\mathcal{G}_j$, and consider the group relationships $(\mathcal{G}_i, \mathcal{G}_k)$ and $(\mathcal{G}_j, \mathcal{G}_k)$. If both group relationships are strong ($p_{i,k} > 0.5$ and $p_{j,k} > 0.5$) or weak ($p_{i,k} \leq 0.5$ and $p_{j,k} \leq 0.5$), then we call it an *agreement* between $\mathcal{G}_i$ and $\mathcal{G}_j$. The total number of agreements between $\mathcal{G}_i$ and $\mathcal{G}_j$ is denoted as $Agree(\mathcal{G}_i, \mathcal{G}_j)$. Having the same $MergeDist$, the pair of groups with more agreements is a better candidate to merge.

If both of the above criteria are the same for two pairs of groups, we always prefer merging groups with smaller sizes (in the number of nodes). More formally, we choose the pair with smaller $MinSize(\mathcal{G}_i, \mathcal{G}_j) = min\{|\mathcal{G}_i|, |\mathcal{G}_j|\}$, where $\mathcal{G}_i$ and $\mathcal{G}_j$ are in this pair.

In Algorithm 5, we utilize a heap to store pairs of groups based on the values of the triple ($MergeDist$, $Agree$, $MinSize$). At each iteration, we pop the group pair with the best ($MergeDist$, $Agree$, $MinSize$) value from the heap, and then merge the pair into one group. At the end of each iteration, we remove the heap elements (pairs of groups) involving either of the two merged groups, update elements involving neighbors of the merged groups, and insert elements involving this new group.

The time cost of the bottom-up approach is the cost of the $SNAP$ algorithm plus the merging cost. The algorithm takes $O(k_{snap}^3 + k_{snap}^2)$ to initialize the heap, then at each iteration at most $O(k_i^3 + k_i^2 \log k_i)$ time to update the heap, where k_{snap} is the size of the grouping resulting from the $SNAP$ operation, and k_i is the size of the grouping at each iteration. Therefore, the time complexity of the bottom-up approach is bounded by $O(|V| \log |V| + k_{snap}^2 |V| + k_{snap}|E| + k_{snap}^4)$. Note that updating the in memory data structures in the bottom-up approach does not need to access the database (i.e. no IOs). All the necessary information for the updates can be found in the current data structures. Therefore, the upper bound of the number of page accesses for the bottom-up approach is $\|V\| + (k_{snap} + 1)\|E\|$.

3.3.3 Drill-Down and Roll-Up Abilities

The top-down and the bottom-up approaches introduced above both start from scratch to produce the summaries. However, it is easy to build an interactive querying scheme, where the users can drill-down and roll-up based on the current summaries. The users can first generate an initial summary using either the top-down approach or the bottom-up approach. However, as we will show in Section 4.3, the top-down approach has significant advantage in both efficiency and summary quality in most practical cases. We suggest using the top-down approach to generate the initial summary. The drill-down operation can be simply achieved by calling the *SplitGroups* function (Algorithm 3). To roll up to a coarser summary, the *MergeGroups* function (Algorithm 5) can be called. However, when the number of groups in the current summary is large, the *MergeGroups* function becomes expensive, as it needs to compare every pair of groups to calculate the $MergeDist$ (see Section 3.3.2). Therefore, using the top-down approach to generate a new summary with the decreased resolution is a better choice to roll-up when the current summary is large.

4. EXPERIMENTAL EVALUATION

In this section, we present experimental results evaluating the effectiveness and efficiency of the $SNAP$ and the k-$SNAP$ operations on a variety of real and synthetic datasets. All algorithms are implemented in C++ on top of PostgreSQL (`http://www.postgresql.org`) version 8.1.3. Graphs are stored in a node table and an edge table in the database, using the following schema: NodeTable(graphID, nodeID, attributeName, attributeType, attributeValue) and EdgeTable(graphID, node1ID, node2ID, edgeType). Nodes with multiple attributes have multiple entries in the node table, and edges with multiple types have multiple entries in the edge table. Accesses to nodes and edges of graphs are implemented by issuing SQL queries to the PostgreSQL database. All experiments were run on a 2.8GHz Pentium 4 machine running Fedora 2, and equipped with a 250GB SATA disk. For all experiments (except the one in Section 4.4.3), we set the buffer pool size to 512MB and working memory size to 256MB.

4.1 Experimental Datasets

In this section, we describe the datasets used in our empirical evaluation. We use two real datasets and one synthetic dataset to explore the effect of various graph characteristics.

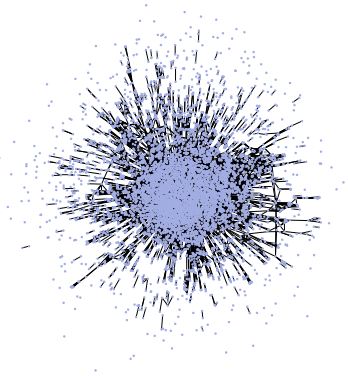

Figure 5: DBLP DB Coauthorship Graph

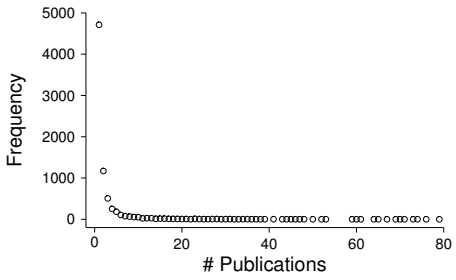

Figure 6: The Distribution of the Number of DB Publications (Avg: 2.6, Stdev: 5.1)

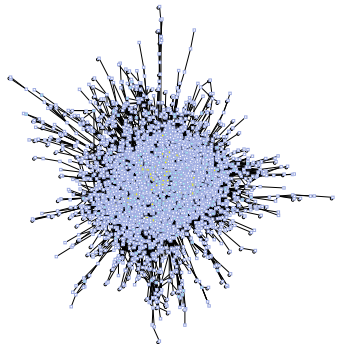

Figure 7: The *SNAP* Result for the DBLP DB Dataset

	Description	#Nodes	#Edges	Avg. Degree
D1	*DB*	7,445	19,971	5.4
D2	D1+*AL*	14,533	37,386	5.1
D3	D2+*OS*+*CC*	22,435	55,007	4.9
D4	D3+*AI*	30,664	70,669	4.6

Table 1: The DBLP Datasets for the Efficiency Experiments

DBLP Dataset This dataset contains the DBLP Bibliography data [12] downloaded on July 30, 2007. We use this data for both effectiveness and efficiency experiments. In order to compare the coauthorship behaviors across different research areas and construct datasets for the efficiency experiments, we partition the DBLP data into different research areas. We choose the following five areas: Database (DB), Algorithms (AL), Operating Systems (OS), Compiler Construction (CC) and Artificial Intelligence (AI). For each of the five areas, we collect the publications of a number of selected journals and conferences in this area[1]. These journals and conferences are selected to construct the four datasets with increasing sizes for the efficiency experiments (see Table 1). These four datasets are constructed as follows: D1 contains the selected DB publications. We add into D1 the selected AL publications to form D2. D3 is D2 plus the selected OS and CC publications. And finally, D4 contains the publications of all the five areas we are interested in. We construct a coauthorship graph for each dataset. The nodes in this graph correspond to authors and edges indicate coauthorships between authors. The statistics for these four datasets are shown in Table 1.

Political Blogs Dataset This dataset is a network of 1490 webblogs on US politics and 19090 hyperlinks between these webblogs [1] (downloaded from `http://www-personal.umich.edu/~mejn/netdata/`). Each blog in this dataset has an attribute describing its political leaning as either liberal or conservative.

[1]**DB**: VLDB J., TODS, KDD, PODS, VLDB, SIGMOD; **AL**: STOC, SODA, FOCS, Algorithmica, J. Algorithms, SIAM J. Comput., ISSAC, ESA, SWAT, WADS; **OS**: USENIX, OSDI, SOSP, ACM Trans. Comput. Syst., HotOS, OSR, ACM SIGOPS European Workshop; **CC**: PLDI, POPL, OOPSLA, ACM Trans. Program. Lang. Syst., CC, CGO, SIGPLAN Notices, ECOOP; **AI**: IJCAI, AAAI, AAAI/IAAI, Artif. Intell.

Synthetic Dataset Most real world graphs show power-law degree distributions and small-world characteristics [13]. Therefore, we use the R-MAT model [8] in the GTgraph suites [2] to generate graphs with power-law degree distributions and small-world characteristics. Based on the statistics in Table 1, we set the average node degree in each synthetic graph to 5. We used the default values for the other parameters in the R-MAT based generator. We also assign an attribute to each node in a generated graph. The domain of this attribute has 5 values. For each node we randomly assign one of the five values.

4.2 Effectiveness Evaluation

We first evaluate the effectiveness of our graph summarization methods. In this section, we use only the top-down approach to evaluate the *k-SNAP* operation, as we compare the top-down and the bottom-up approaches in Section 4.3.

4.2.1 DBLP Coauthorship Networks

In this experiment, we are interested in analyzing how researchers in the database area coauthor with each other. As input, we use the DBLP DB subset (see D1 of Table 1 and Figure 5). Each node in this graph has one attribute called *PubNum*, which is the number of publications belonging to the corresponding author. By plotting the distribution of the number of publications of this dataset in Figure 6, we assigned another attribute called *Prolific* to each author in the graph indicating whether that author is prolific: authors with ≤ 5 papers are tagged as low prolific (LP), authors with > 5 but ≤ 20 papers are prolific (P), and the authors with > 20 papers are tagged as highly prolific (HP).

We first issue a *SNAP* operation on the *Prolific* attribute and the coauthorships. The result is visualized in Figure 7. Groups with the HP attribute value are colored in yellow, groups with the P value are colored in light blue, and the remaining groups with the attribute value LP are in white. The *SNAP* operation results in a summary with 3569 groups and 11293 group relationships. This summary is too big to analyze. On the other hand, if we apply the *SNAP* operation on only the *Prolific* attribute (i.e. not considering any relationships in the *SNAP* operation), we will get a summary with only 3 groups as visualized in the top left figure in Table 2. The bold edges between two groups indicate strong group relationships (with more than 50% participation ra-

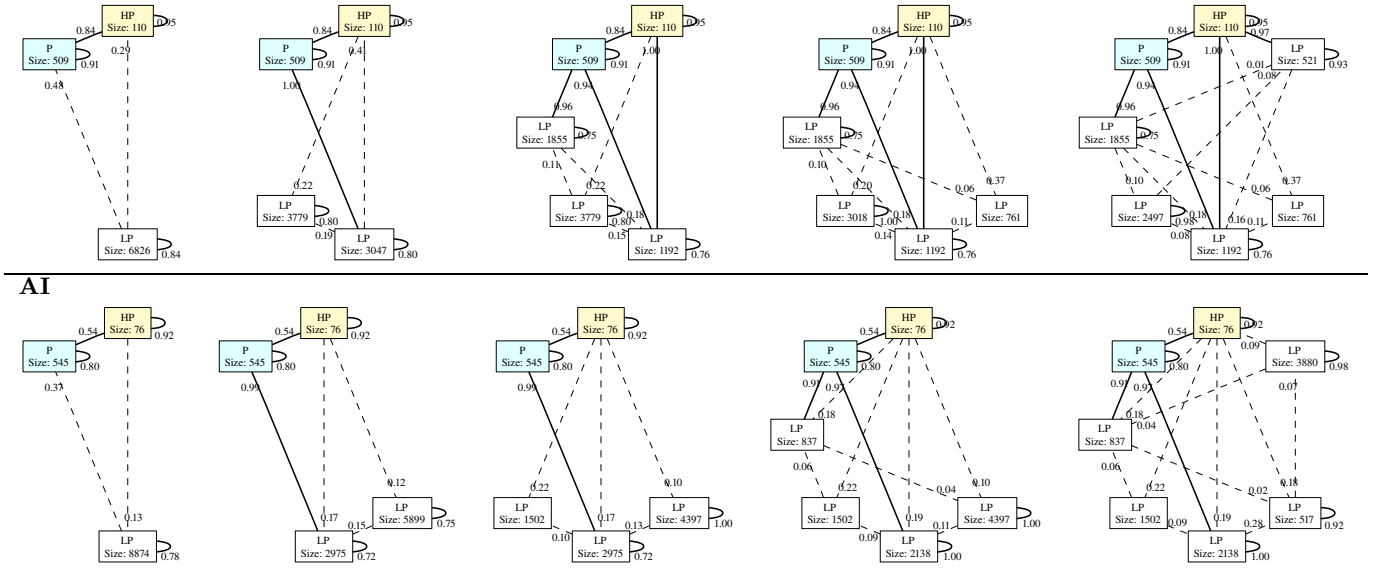

Table 2: The Aggregation Results for the DBLP DB and AI Subsets

tio), while dashed edges are weak group relationships. This summary shows that the HP researchers as a whole have very strong coauthorship with the P group of researchers. Researchers within both groups also tend to coauthor with people within their own groups. However, this summary does not provide a lot of information for the LP researchers: they tend to coauthor strongly within their group and they have some connection with the HP and P groups.

Now we make use of the *k-SNAP* operation to produce summaries with multiple resolutions. The first row of figures in Table 2 shows the *k-SNAP* results for k =4, 5, 6 and 7. As k increases, more details are shown in the summaries.

When $k = 7$, the summary shows that there are 5 subgroups of LP researchers. One group of 1192 LP researchers strongly collaborates with both HP and P researchers. One group of 521 only strongly collaborates with HP researchers. One group of 1855 only strongly collaborates with P researchers. These three groups also strongly collaborate within their groups. There is another group of 2497 LP researchers that has very weak connections to other groups but strongly cooperates among themselves. The last group has 761 LP researchers, who neither coauthor with others within their own group nor collaborate strongly with researchers in other groups. They often write single author papers.

Now, in the *k-SNAP* result for $k = 7$, we are curious if the average number of publications for each subgroup of the LP researchers is affected by the coauthorships with other groups. The above question can be easily answered by applying the *avg* operation on the *PubNum* attribute for each group in the result of the *k-SNAP* operation.

With this analysis, we find that the group of LP researchers who collaborate with both P and HP researchers has a high average number of publications: 2.24. The group only collaborating with HP researchers has 1.66 publications on average. The group collaborating with only the P researchers has on average 1.55 publications. The group that tends to only cooperate among themselves has a low average number of publications: 1.26. Finally, the group of mostly single authors has on average only 1.23 publications. Not surprisingly, these results suggest that collaborating with HP and P researchers is very helpful for the low prolific (often beginning) researchers.

Next, we want to compare the database community with the AI community to see whether the coauthorship relationships are different across these two communities. We constructed the AI coauthorship graph with 9495 authors and 16070 coauthorships from the DBLP AI subset. The distribution of the number of publications of AI authors is similar to the DB authors, thus we use the same method to assign the *Prolific* attribute to these authors. The *SNAP* operation on the *Prolific* attribute and coauthorships results in a summary with 3359 groups and 7091 group relationships. The second row of figures in Table 2 shows the *SNAP* result based only on the *Prolific* attribute and the *k-SNAP* results for k =4, 5, 6 and 7. Comparing the summaries for the two communities for $k = 7$, we can see the differences across the two communities: The HP and P groups in the AI community have a weaker cooperation than the DB community; and there isn't a large group of LP researchers who strongly coauthor with both HP and P researchers in the AI area.

As this example shows, by changing the resolutions of summaries, users can better understand the characteristics of the original graph data and also explore the differences and similarities across different datasets.

4.2.2 Political Blogs Network

In this experiment, we evaluate the effectiveness of our graph summarization methods on the political blogs network (1490 nodes and 19090 edges). The *SNAP* operation based on the political leaning attribute and the links between blogs results in a summary with 1173 groups and 16657 group relationships. The *SNAP* result based only on the attribute and the *k-SNAP* result based on both the attribute and the links for $k = 7$ are shown in Table 3.

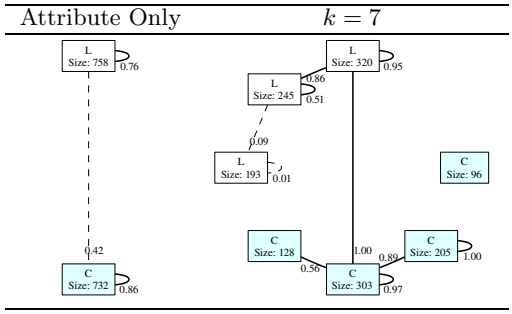

Table 3: The Aggregation Results for the Political Blogs Dataset

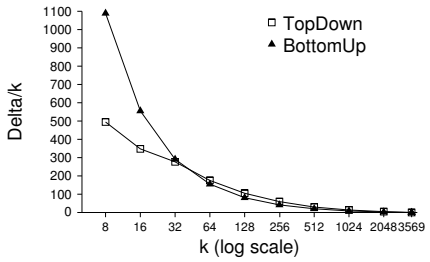

Figure 8: Quality of Summaries: Top-Down vs. Bottom-Up

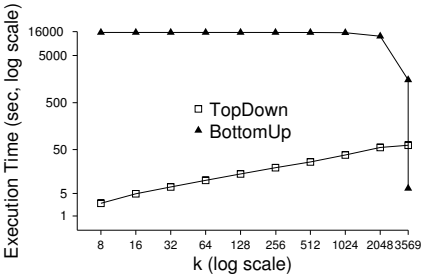

Figure 9: Efficiency: Top-Down vs. Bottom-Up

From the results, we see that there are a group of liberal blogs and a group of conservative blogs that interact strongly with each other (perhaps to refute each other). Other groups of blogs only connect to blogs in their own communities (liberal or conservative), if they do connect to other blogs. There is a relatively large group of 193 liberal blogs with almost no connections to any other blogs, while such isolated blogs compose a much smaller portion (96 blogs) of all the conservative blogs. Overall, conservative blogs show a slightly higher tendency to link with each other than liberal blogs, which is consistent with the conclusion from the analysis in [1]. Given that the blogs data was collected right after the US 2004 election, the authors in [1] speculated that the different linking behaviors in the two communities may be correlated with eventual outcome of the 2004 election.

4.3 k-SNAP: Top-Down vs. Bottom-Up

In this section, we compare the top-down and the bottom-up *k-SNAP* algorithms, both in terms of effectiveness and efficiency. We use the DBLP DB subset (D1 in Table 1) and apply both approaches for different k values.

For the effectiveness experiment, we use the Δ measure introduced in Section 2.2 to assess the qualities of summaries. Note that for a given k value, smaller Δ value means better quality summary, but for different k values, comparing Δ does not make sense, as a higher k value tends to result in a higher Δ value according to Equation 1. However, if we normalize Δ by k, we get the average contribution of each group to the Δ value, then we can compare $\frac{\Delta}{k}$ for different k values.

We acknowledge that $\frac{\Delta}{k}$ is not a perfect measure for "quantitatively" evaluating the quality of summaries. However, quality assessment is a tricky issue in general, and $\frac{\Delta}{k}$, though crude, is an intuitive measure for this study.

Figure 8 shows the comparison of the summary qualities between the top-down and the bottom-up approaches. Note that the x-axis is in log scale and the y-axis is $\frac{\Delta}{k}$. First, as k increases, both methods produce higher quality summaries. For small k values, top-down approach produces significantly higher quality summaries than the bottom-up approach. This is because, the bottom-up approach starts from the grouping produced by the *SNAP* operation. This initial grouping is usually very large, in this case, it contains 3569 groups. The bottom-up approach has to continuously merge groups until the number of groups decreases to a small k value. Each merge decision is only made based on the current grouping, and errors can easily accumulate. In contrast, the top-down approach starts from the maximum A-compatible grouping, and only needs a small number of splits to reach the result. Therefore, small amount of errors is accumulated. As k becomes larger, the bottom-up approach shows slight advantage over the top-down approach.

The execution times for the two approaches are shown in Figure 9. Note that both axes are in log scale. The top-down approach significantly outperform the bottom-up approach, except when k is equal to the size of the grouping resulting from the *SNAP* operation. Initializing the heap takes a lot of time for the bottom-up approach, as it has to compare every pair of groups. This situation becomes worse, if the size of the initial grouping is very large.

In practice, users are more likely to choose small k values to generate summaries. The top-down approach significantly outperforms the bottom-up approach in both effectiveness and efficiency for small k values. Therefore, the top-down approach is preferred for most practical uses. For all the remaining experiments, we only consider the top-down approach.

	# Groups	# Group Relationships	Time(sec)
D1	3569	11293	6.4
D2	7892	26031	16.1
D3	11379	35682	27.9
D4	15052	44318	44.0

Table 4: The *SNAP* Results for the DBLP Datasets

4.4 Efficiency Experiment

This section evaluates the efficiency the *SNAP* and the *k-SNAP* operations.

4.4.1 SNAP Efficiency

In this section, we apply the *SNAP* operation on the four DBLP datasets with increasing sizes (see Table 1). Table 4 shows the number of groups and group relationships in the summaries produced by the *SNAP* operation on the attribute *Prolific* (defined in the same way as in Section 4.2.1) and coauthorships, as well as the execution times. Even for the largest dataset with 30664 nodes and 70669 edges, the execution is completed in 44 seconds. However, all of the *SNAP* results are very large. The summary sizes are comparable to the input graphs. Such large summaries are often not very useful for analyzing the input graphs. This is an anecdotal evidence of why the *k-SNAP* operation is often more desired than the *SNAP* operation in practice.

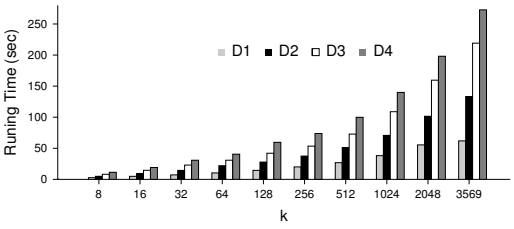

Figure 10: Efficiency of *k-SNAP* on the DBLP Datasets

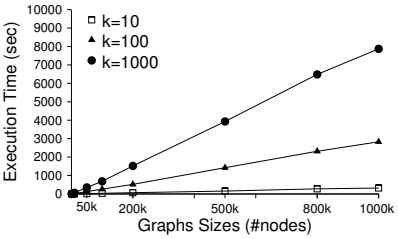

Figure 11: Efficiency of *k-SNAP* on the Synthetic Datasets

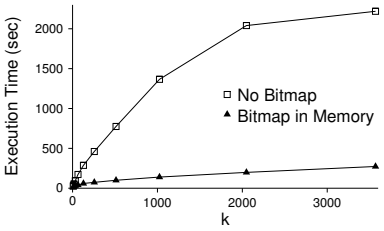

Figure 12: Bitmap in Memory vs. No Bitmap

4.4.2 k-SNAP Efficiency

This section evaluates the efficiency of the top-down *k-SNAP* algorithm on both the DBLP and the synthetic datasets.

DBLP Data In this experiment, we apply the top-down *k-SNAP* evaluation algorithm on the four DBLP datasets shown in Table 1 (the *k-SNAP* operation is based on *Prolific* attribute and coauthorships). The execution times with increasing graph sizes and increasing k values are shown in Figure 10. For these datasets, the performance behavior is close to linear, since the execution times are dominated by the database page accesses (as discussed in Section 3.3.1).

Synthetic Data We apply the *k-SNAP* operation on different sized synthetic graphs with three k values: 10, 100 and 1000. The execution times with increasing graph sizes are shown in Figure 11. When $k = 10$, even on the largest graph with 1 million nodes and 2.5 million edges, the evaluation algorithm finishes in about 5 minutes. For a given k value, the algorithm scales nicely with increasing graph sizes.

4.4.3 Evaluation with Very Large Graphs

So far, we have assumed that the amount of working memory is big enough to hold all the data structures (shown in Figure 4) used in the evaluation algorithms. This is often the case in practice, as large multi GB memory configurations are common and many graph datasets can fit in this space (especially if a subset of large graph is selected for analysis). However, our methods also work, when the graph datasets are extremely large and this in-memory assumption in not valid. In this section, we discuss the behaviors of our methods for this case. We only consider the most practically useful top-down *k-SNAP* algorithm for this experiment.

In the case when the most memory consuming data structure, namely the neighbor-groups bitmap (see Figure 4), cannot fit in memory, the top-down approach drops the bitmap data structure. Without the bitmap, each time the algorithm splits a group, it has to query the edges information in the database to infer the neighbor-groups. We have implemented a version of the top-down *k-SNAP* algorithm without the bitmap data structure, and compared it with the normal top-down algorithm.

To keep this experiment manageable, we scaled down the experiment settings. We used the DBLP D4 dataset in Table 1, and set the buffer pool size and working memory size to 16MB and 8MB, respectively. This "scaled-down" experiment exposes the behaviors of the two versions of the top-down algorithm, while keeping the running times reasonable. As shown in Figure 12, the version of the top-down approach without bitmap is much slower than the normal version. This is not surprising as the former incurs more disk IOs.

Given the graph size and the k value, our current implementation can decide in advance whether the bitmap can fit in the working memory, by estimating the upper bound of the bitmap size. It can then choose the appropriate version of the algorithm to use. In the future, we plan on designing a more sophisticated version of the top-down algorithm in which part of the bitmap can be kept in memory when the available memory is small.

5. RELATED WORK

Graph summarization has attracted a lot of interest from both the sociology and the database research communities. Most existing works on graph summarization use statistical methods to study graph characteristics, such as degree distributions, hop-plots and clustering coefficients. Comprehensive surveys on these methods are provided in [6] and [13]. A-plots [7] is a novel statistical method to summarize the adjacency matrix of graphs for outlier detection. Statistical summaries are useful but hard to control and navigate. Methods for mining frequent graph patterns [11, 19, 23] are also used to understand the characteristics of large graphs. Washio and Motoda [20] provide an elegant review on this topic. However, these mining algorithms often produces an overwhelmingly large number of frequent patterns. Various graph partitioning algorithms [14, 18, 22] are used to detect community structures (dense subgraphs) in large graphs. SuperGraph [17] employs hierarchical graph partitioning to visualize large graphs. However, graph partitioning techniques largely ignore the node attributes in the summarization. Studies on graph visualization are surveyed in [3, 10]. For very large graphs, these visualization methods are still not enough. Unlike these existing methods, we introduce two database-style operations to summarize large graphs. Our method allows users to easily control and navigate through summaries.

Previous research [4, 5, 15] have also studied the problem of compressing large graphs, especially Web graphs. However, these graph compression methods mainly focus on compact graph representation for easy storage and manipulation, whereas graph summarization methods aim at producing small and understandable summaries.

Regular equivalence is introduced in [21] to study social roles of nodes based on graphs structures in social networks. It shares resemblance with the *SNAP* operation. However, regular equivalence is defined only based on the relationships

between nodes. Node attributes are largely ignored. In addition, the *k-SNAP* operation relaxes the stringent equivalence requirement of relationships between node groups, and produces user controllable multi-resolution summaries.

The *SNAP* algorithm (Algorithm 1) shares similarity with the automorphism partitioning algorithm in [9]. However, the automorphism partitioning algorithm only partitions nodes based on node degrees and relationships, whereas *SNAP* can be evaluated based on arbitrary node attributes and relationships that a user selects.

6. CONCLUSIONS AND FUTURE WORK

This paper has introduced two aggregation operations *SNAP* and *k-SNAP* for controlled and intuitive database-style graph summarization. Our methods allow users to freely choose node attributes and relationships that are of interest, and produce summaries based on the selected features. Furthermore, the *k-SNAP* aggregation allows users to control the resolutions of summaries and provides "drill-down" and "roll-up" abilities to navigate through the summaries. We have formally defined the two operations and proved that evaluating the *k-SNAP* operation is NP-complete. We have also proposed an efficient algorithm to evaluate the *SNAP* operation and two heuristic algorithms to approximately evaluate the *k-SNAP* operation. Through extensive experiments on a variety of real and synthetic datasets, we show that of the two *k-SNAP* algorithms, the top-down approach is a better choice in practice. Our experiments also demonstrate the effectiveness and efficiency of our methods. As part of future work, we plan on designing a formal graph data model and query language that allows incorporation of *k-SNAP*, along with a number of other additional common and useful graph matching methods.

7. ACKNOWLEDGMENT

This research was supported by the National Institutes of Health under grant 1-U54-DA021519-01A1, the National Science Foundation under grant DBI-0543272, and an unrestricted research gift from Microsoft Corp. We thank Taneli Mielikainen and Yiming Ma for their valuable suggestions during the early stage of this research. We also thank the reviewers of this paper for their constructive comments on a previous version of this manuscript.

8. REPEATABILITY ASSESSMENT RESULT

All the results in this paper were verified by the SIGMOD repeatability committee.

9. REFERENCES

[1] L. A. Adamic and N. Glance. The political blogosphere and the 2004 US Election: Divided they blog. In *Proceedings of the 3rd International Workshop on Link Discovery*, pages 36–43, 2005.

[2] D. A. Bader and K. Madduri. GTgraph: A suite of synthetic graph generators. `http://www.cc.gatech.edu/~kamesh/GTgraph`.

[3] G. Battista, P. Eades, R. Tamassia, and I. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[4] D. K. Blandford, G. E. Blelloch, and I. A. Kash. Compact representations of separable graphs. In *Proceedings of SODA'03*, pages 679–688, 2003.

[5] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *Proceedings of WWW'04*, pages 595–602, 2004.

[6] D. Chakrabarti and C. Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Comput. Surv.*, 38(1), 2006.

[7] D. Chakrabarti, C. Faloutsos, and Y. Zhan. Visualization of large networks with min-cut plots, A-plots and R-MAT. *Int. J. Hum.-Comput. Stud.*, 65(5):434–445, 2007.

[8] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A recursive model for graph mining. In *Proceedings of 4th SIAM International Conference on Data Mining*, 2004.

[9] D. G. Corneil and C. C. Gotlieb. An efficient algorithm for graph isomorphism. *J. ACM*, 17(1):51–64, 1970.

[10] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Trans. Vis. Comput. Graph.*, 6(1):24–43, 2000.

[11] J. Huan, W. Wang, J. Prins, and J. Yang. SPIN: Mining maximal frequent subgraphs from graph databases. In *Proceedings of KDD'04*, pages 581–586, 2004.

[12] M. Ley. DBLP Bibliography. `http://www.informatik.uni-trier.de/~ley/db/`.

[13] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[14] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113, 2004.

[15] S. Raghavan and H. Garcia-Molina. Representing Web graphs. In *Proceedings of ICDE'03*, pages 405–416, 2003.

[16] F. S. Roberts and L. Sheng. How hard is it to determine if a graph has a 2-role assignment? *Networks*, 37(2):67–73, 2001.

[17] J. F. Rodrigues, A. J. M. Traina, C. Faloutsos, and C. Traina Jr. SuperGraph visualization. In *Proceedings of the 8th IEEE International Symposium on Multimedia*, pages 227–234, 2006.

[18] J. Sun, Y. Xie, H. Zhang, and C. Faloutsos. Less is more: Sparse graph mining with compact matrix decomposition. *Stat. Anal. Data Min.*, 1(1):6–22, 2008.

[19] W. Wang, C. Wang, Y. Zhu, B. Shi, J. Pei, X. Yan, and J. Han. GraphMiner: A structural pattern-mining system for large disk-based graph databases and its applications. In *Proceedings of SIGMOD'05*, pages 879–881, 2005.

[20] T. Washio and H. Motoda. State of the art of graph-based data mining. *SIGKDD Explor. Newsl.*, 5(1):59–68, 2003.

[21] D. R. White and K. P. Reitz. Graph and semigroup homomorphisms on semigroups of relations. *Social Networks*, 5(2):193–234, 1983.

[22] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger. SCAN: A structural clustering algorithm for networks. In *Proceedings of KDD'07*, pages 824–833, 2007.

[23] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of ICDM'02*, pages 721–724, 2002.