Graph Compression

Fang Zhou

Department of Computer Science and
Helsinki Institute for Information Technology HIIT,
PO Box 68, FI-00014 University of Helsinki, Finland
fang.zhou@cs.helsinki.fi

Abstract. Graphs form the foundation of many real-world datasets. At the same time, the size of graphs presents a big obstacle to understand the essential information they contain. In this report, I mainly review the framework in article [1] for compressing large graphs. It can be used to improve visualization, to understand the high-level structure of the graph, or as a pre-processing step for other data mining algorithms. The compression model consists of a graph summary and a set of edge corrections. This framework can produce either lossless or lossy compressed graph representations. Combined with Minimum Description Length (MDL), it can produce a compact summary. The performance of this framework is evaluated on multiple sets of real data graph.

1 Introduction

Nowadays, graphs are everywhere. Examples are social networks, communication networks, biological networks and World Wide Web. With the development of technology and the discovery of new knowledge, the size of these graphs become larger and larger. Some graphs contain millions or even billions of nodes and edges. Such a bulky graph presents new challenges to graph mining. One issue is that a huge graph may severely restrict the application of existing pattern mining technologies. Additionally, directly visualizing such a large graph is beyond our capability.

The large number of nodes and edges encumber our discovery. Imagine a compressed graph, conserving the characteristics of the original graph. We can easily visualize it. The goal of compressing a graph is to make the high-level structure of the graph easily understood. Therefore, informative graph compression techniques are required and have wide application domains. As an example, Chen [2] successfully applies a graph compression method to reduce the number of embeddings when searching frequent subgraphs in a large graph. Navlakha [3] reveals biological modules with the help of compressed graphs. Furthermore, Chen [4] incorporates the compressed graph notion with a generic topological OLAP framework to realize online graph analysis.

Part of the graph compression problem is addressed by clustering algorithms, as group members have similar characteristics and can be represented by (super)nodes. Similar work is done by graph partitioning algorithms (e.g. [5, 6]),

which can be used to detect hidden community structures. However, most of these algorithms are based on the distance matrix, and do not focus on the structure of the original graph.

Another feasible technique is to construct a small graph consisting of a set of the most important nodes and edges. Much literature has discussed how to rank the centrality of nodes and edges (e.g. [7, 8]). For this work, the user needs to specify a parameter which restricts the size of the output, while this parameter sometimes will implicitly omit some information.

Frequent subgraph mining (e.g. [9]) can be an alternative technique, as it already gives insight into the structure of subgraphs then we can only focus on the rest of the graph. However, finding frequent subgraphs in a huge graph is a complex computational task.

In this report, I focus on an information-theoretic technique [1] which can construct both lossless and lossy compressed graph representations. This representation R has two parts: one is a graph summary S, which captures the important communities and relationships in the original graph; the other is a set of edge corrections C, which helps to re-create the original graph from a compressed graph.

The remainder of this report is organized as follows. We first describe several compressed representation models in Section 2. In Section 3, we review one algorithm to construct a globally compact representation and discuss its performance in Section 4. Finally, we discuss future work in Section 5.

2 Compressed Representation Models

In this section, I first review some compressed graph representations proposed by Navlakha [1]. Then, I discribe a complementary representation model by Tian et al [10].

2.1 A general compressed graph representation

Assume an unweighted graph $G = (V_G, E_G)$ is given. It has a representation $R = (S, C)$, which consists of a graph summary $S = (V_S, E_S)$ and a set of edge corrections C. Every node v in V_G belongs to a (super)node V in V_S which represents a set of nodes of G. A (super)edge $E = (V_i, V_j)$ in E_S represents the set of all edges connecting all pairs of nodes in V_i and V_j. I.e., we simply collapse one bi-partite subgraph into two (super)nodes V_i and V_j, and replace all the edges by a (super)edge between the (super)nodes. The edge corrections in C have two forms: $+e$ means adding e when re-creating the original graph, and $-e$ means deleting the edge when performing the re-creation. Now we have the definition of an exact representation.

Definition 1. *(An exact representation). Given a graph $G = (V_G, E_G)$, suppose V_G can be partitioned into groups, i.e., $V_G = V_{G_1} \cup V_{G_2} \cup \cdots \cup V_{G_k}$, such that $V_{G_i} \cap V_{G_j} = \emptyset$ $(1 \leq i \neq j \leq k)$. Now we can summarize G into S, where S has*

exactly k (super)nodes $V_1, V_2, \ldots, V_k$ that corresponds to each group ($V_{G_i} \mapsto V_i$), and each (super)edge $(V_i, V_j) \in E_S$ represents the edges between all pair of nodes in V_i and V_j. The edge correction list C contains entries of the form $'-(v_i, v_j)'$ for the edges represented by E_S that are not present in G, and entries of the form $'+(v_i, v_j)'$ for the edges that are in G but are not shown by (super)edges in S. Pair (S,C) is an exact representation of G.

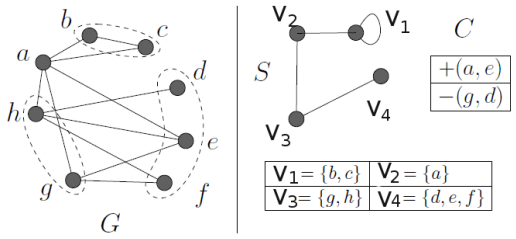

Fig. 1. A simple example of one graph's representation. Source: [1].

Here, we give an illustrated example. Figure 1 shows the original graph (left) and its representation (right). The original graph is divided into four parts, and its size (number of edges) decreases from 11 to 6 (4 edges in the graph summary and 2 in edge correction).

Note that in Definition 1, each node v in G belongs to exactly one (super)node in S. That is, (super)nodes in S form disjoint subsets of nodes of G and the union covers the set V_G. Meanwhile, a self-edge can also exist in the compressed graph, if, inside a subgraph, the nodes densely connect with each other.

The idea of this graph compression structure is to exploit the similarity of the link structure of the nodes in the graph to realize space saving. The goal is that S is a compact graph summary which highlights the key structure and patterns of G, and C helps re-create the original graph. Intuitively, it is easy to re-create the original graph from an exact representation, since it only needs to expand each (super)node and correct edge connections through C.

2.2 MDL representation

Constructing a representation R is one form of space-saving approach. Here we define the cost of a representation as the sum of the storage costs of its two components, i.e., $cost(R) = |E_S| + |C|$. (The cost of mapping of nodes onto (super)nodes are ignored, since the cost are quite small compared to the storage costs of the edge sets E_S and C.) The edge sets E_S and the edge corrections C are solely determined by the internal structure of (super)nodes.

In information theory, the concept of Minimum Description Length (MDL) [11] states that the best representation for a given set of data is the one that leads

to the largest compression of the data. Therefore, the representation of a given graph which has the minimum cost is the best summary of that graph. We denote this best graph summary as MDL representation $\hat{R}$.

Theorem 1. *(MDL representation). Given a graph G, an MDL representation $\hat{R}$ for G is one whose cost is the minimum, i.e., $\nexists R', cost(R') < cost(R)$.*

Here, we simplify the problem, assuming V_S is fixed, and explain how to construct a representation with minor cost. Consider any two (super)nodes V_i and V_j. Π_{ij} denotes all possible edges between all pairs of nodes (v_m, v_n), $v_m \in V_i$, $v_n \in V_j$; that is, $|\Pi_{ij}| = |V_i| * |V_j|$, where $|V_i|$ represents the number of nodes belonging to (super)node V_i. The edges actually present between V_i and V_j is denoted by A_{ij}, i.e., $A_{ij} = \Pi_{ij} \cap E_G$. Now we have two ways to represent A_{ij} in a compressed graph. One is adding all present edges in C, so the cost is $|A_{ij}|$. This is suitable in the case where two (super)nodes are loosely connected. Another is to add one (super)edge between two (super)nodes and record other nonexistent edges into C, so the cost is $1 + |\Pi_{ij}| - |A_{ij}|$. This can be used when two (super)nodes have a dense connection. An extreme case is a complete bi-partite subgraph, where only one (super)edge is needed.

Therefore, we simply choose, for each pair of V_i, V_j of (super)nodes, the smaller cost c_{ij} of coding information A_{ij}, i.e., $c_{ij} = \min\{|A_{ij}|, 1+|\Pi_{ij}|-|A_{ij}|\}$. So, for a given graph, the representation cost is determined solely by the set of (super)nodes chosen. However, finding the best set of (super)nodes for an MDL representation is computationally expensive. We will discuss this issue in Section 3.

2.3 Approximate representation

We have previously discussed an exact representation which can be used to reconstruct the original graph. However, in most cases, users do not need the original graph exactly. They can accept some tolerable errors in order to have a more compact graph summary. A reasonably accurate construction is enough. So, the exact neighbor set of a node can be replaced by an approximate neighbor set. The error means that a neighbor of node v may not be v's neighbor in the representation, or vice-versa. We next define an approximate representation formally.

Definition 2. *(An approximate representation). Given a graph G and a user-specified bounded error ϵ $(0 \leq \epsilon \leq 1)$, an ϵ-approximate representation $R_\epsilon = (S_\epsilon, C_\epsilon)$ satisfies the condition that, for each $v \in G$, the $error(v) = |N'_v - N_v| + |N_v - N'_v| \leq \epsilon|N_v|$. Here, N_v and N'_v denote the set of v's neighbors in G and R_ϵ respectively.*

In Definition 2, the term $|N'_v - N_v|$ is the number of v's edges in the representation which are not in the original graph, and the term $|N_v - N'_v|$ is the number of v's edges which can not be reconstructed. For each node, it is allowed

to have $\epsilon|N_v|$ errors at most. Therefore, ϵ-approximate representation R_ϵ shows at least $(1-\epsilon)$ fraction of all edges correctly.

Notice that, the MDL representation $\hat{R}$ is a representaion with the error $\epsilon = 0$ and has a minimum cost. So, one simple and direct way to construct an approximate representation is to remove some edges from C and S of an MDL representation, where the removal action does not violate the approximatation guarantee. Removing edges in C is easy; we just check whether removing edge $e = (u, v)$ guarantees that at least $(1-\epsilon)$ fraction of the orignal edges are still adjancent to u and v. Removing a (super)edge in S is more complex, since we need to check whether such removal will violate the guarantee of all edges' endvertices that have been merged into the two (super)nodes. Navlakha et al. [1] propose the ApxGreeDY method to build an ϵ-approximate representation directly from an original graph. However, we will not elaborate it here.

2.4 Label compatible representations

In above sections, the representation model deals with unlabeled graphs. However, edges often have labels. As an example, consider biological networks, where nodes are biological entites, such as genes, protein, and edges have meaning, such as "codes for", "functionally associated to", "belongs to". Therefore, it is more meaningful when compressing a graph that these labels are taken into account. Here, I describe two representations presented by Tian [10].

Given a graph $G = (V_G, E_G)$, $\Upsilon = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ denotes the set of edge types and $\Lambda = \{a_1, a_2, \ldots a_t\}$ represents the set of nodes' attributes. The value of the attribute a_m of node v_i is denoted as $a_m(v_i)$, and $\gamma_i(G)$ represents the set of edges of type γ_i. $NeighborGroups_{\gamma_m}(v_i) = \{V_{v_t} | (v_i, v_t) \in \gamma_i(G) \text{ and } v_t \in V_G\}$ represents the group of (super)nodes to which v_i's neighbors belong and their edge type is γ_m. Here, V_{v_t} denotes the (super)node to which node v_t belongs.

We first describe a compressed graph that consists of a set of (super)nodes, in which each node has the same attributes. Its definition is as follows.

Definition 3. *(An attributes compatible representation S_A.) Assume a graph $G = (V_G, E_G)$ and a set of attributes $A \subset \Lambda(G)$. If a representation $S_A = (V_1, V_2, \ldots, V_n)$ satisfies that: $\forall v_i, v_j \in V_G$, if $v_i \in V_m$ and $v_j \in V_m$, then $\forall a_k \in A$, $a_k(v_i) = a_k(v_j)$. Then S_A is compatible with A.*

The difference between the previous representations and this attributes-compatible representation is that the former focuses on the link structure of nodes, i.e., the similarity of nodes' neighborhood, and the latter focuses on the attributes of nodes. One easy way to construct a global simplest representation S_A is to classify nodes based on the value of a node's attribute.

Since S_A only accounts for the node attributes, it ignores the relationship between nodes. Therefore, we describe another representation S_{AL} that is compatible with attributes and relationships.

Definition 4. *(An attributes and relationships compatible representation S_{AL}.) Given a graph $G = (V_G, E_G)$, a set of attributes $A \subset \Lambda(G)$, and a set of relationship types $L \subset \Upsilon$, a representation $S_{AL} = (V_1, V_2, \ldots, V_n)$ is compatible*

with attributes A and relationship types L if: (1) $S_{AL} \subset S_A$*; (2)* $\forall v_i, v_j \in V_G$*, if* $v_i \in V_m$ *and* $v_j \in V_m$*, then* $\forall \gamma_m \in L$*, NeighborGroups* $_{\gamma_m}(v_i)$ = *NeighborGroups* $_{\gamma_m}(v_j)$.

In a S_{AL}, every node inside a (super)node has exactly the same value for A and is adjacent to the same set of groups for all relationships in R.

3 Algorithm

In this section, I mainly review one algorithm, Greedy, for computing the MDL representation. We first discuss how to choose which nodes to group together.

3.1 Cost of grouping nodes

Recall that the cost of storing the information between one (super)node V_i and one of its neighbors V_j is $c_{ij} = \min\{|A_{ij}|, 1 + |\Pi_{ij}| - |A_{ij}|\}$. If V_i and V_j are not neighbors, then $c_{ij} = 0$. So the cost of one (super)node c_i is the sum of the cost of storing the information between V_i and all its neighbors, i.e., $c_i = \sum_{k=1}^{N_{v_i}} c_{ik}$, where N_{v_i} is the number of v_i's neighbors. Therefore, we have the cost reduction $s(u, v)$ for any pair of nodes when combining node u and v into a new node w.

$$s(u, v) = \frac{c_u + c_v - c_w}{c_u + c_v}$$

The cost reduction $s(u, v)$ is the ratio of the reduction in cost as u and v are merged into w. Intuitively, any two nodes sharing common neighbors can give a cost reduction. The reason for using relative cost reduction is to give precedence to a pair of nodes whose common neighbors occupy a large proportion of their total neighbors. Furthermore, the value of $s(u, v)$ can be either positive or negative (or zero). The maximum value of $s(u, v)$ is 0.5 which occurs when two nodes have exactly same neighbors, and the minimum value is negative when the merger actually increases the cost. Here, we are only interested in the pair (u, v) that has a positive $s(u, v)$.

3.2 Greedy algorithm

The main idea of the Greedy algorithm [1] is to iteratively group two nodes with the highest cost reduction.

The Greedy algorithm has three phases: Initialization, Iterative merging and Output. It is outlined as Algorithm 1. In the initialization phase, the cost reduction s for all pairs of nodes who are 2 hops apart will be computed. When the value of s is greater than 0, the pair of nodes and s value will be stored in a standard max-heap strcture H (Line 5). The reason to add the "2 hops apart"restriction is to ensure that such pair of nodes has at least one common neighbor, since it is impossible to reduce the cost for a pair of nodes that have

no common neighbor. Furthermore, the heap strcture makes the calculation efficient.

In each round of merging phase, the globally best pair of nodes will be chosen (Line 9), so node u and v will be removed from V_S, and a new (super)node w will be added into V_S (Line 11). Now, we need to update two situations. The first is to delete all pairs which have u or v in H, such as (u, x) and (v, x), since u and v are not in the compressed graph any more. Then, we add (w, x) to H when $s(w, x)$ is greater than 0 (Line 12-15). The second is to examine whether the removal of u and v will affect the cost reduction of their neighbors, denoted by N_w, which is the union of the u's neighbors N_u and v's neighbors N_v)(Line 16-19). We need to update the cost reduction value $s(x, y)$ for each x which is one of the new (super)node w's neighors. Meanwhile, y is 2 hops apart from x. The total iteration will stop when there is no pair of nodes whose cost reduction is bigger than 0.

In the output phase, the graph summary edge set, E_S, and the edge corrections, C, will be constructed. Recall that $c_{uv} = \min\{|A_{uv}|, 1 + |\Pi_{uv}| - |A_{uv}|\}$, so if $|A_{uv}| > (1 + |\Pi_{uv}|)/2$, we add a (super)edge (u, v) into E_S and other edge $-(a, b)$, which is not in the original graph, into C. Otherwise, we add all edges between (super)nodes into C.

The Greedy algorithm can construct a much higher-level compressed graph for a given graph. However, it has a high computational cost for finding the most suitable pair of nodes to merge in every iteration. In [1], the authors propose another light-weight algorithm, Randomized, which randomly picks a node and merges it with the best node that gives the maximum cost reduction into (super)nodes. The resulting graph of Randomized is not as compact as Greedy's. However, it is efficient. Here, we omit a detailed description of the procedure of the Randmonized method.

4 Experiments and Conclusion

In this section, I describe the performance [1] of the MDL model from three aspects. First, how much information can be compressed? Second, how efficient is the algorithm? Third, how does the cost reduction compare with other compression methods?

4.1 Experimental Data

For the experiments, the authors of article [1] choose the CNR dataset[1], a web-graph dataset which is extracted from a crawl of the CNR domain. The size of graphs increases from 5k to 100k (CNR-100k has 100k nodes and 405,586 edges). For performance comparison, they use the following three datasets: RouteView[2]

[1] http://law.dsi.unimi.it/
[2] http://www.routeviews.org/

Algorithm 1 Greedy Algorithm [1]

/* Initialization phase */
$V_S = V_G$; $H = 0$;
for all pairs $(u, v) \in V_S$ that are 2 hops apart **do**
  if $s(u, v) > 0$ **then**
    insert $(u, v, s(u, v))$ into H;

/* Iterative merging phase */
while $H \neq \emptyset$ **do**
  Choose pair $(u, v) \in H$ with the largest $s(u, v)$
  $w = u \cup v$; /* merge (super)nodes u and v into w */
  $V_S = V_S - \{u, v\} \cup \{w\}$
  for all $x \in V_S$ that are within 2 hops of u and v **do**
    Delete (u, x) and (v, x) from H;
    if $s(w, x) > 0$ **then**
      insert $(w, x, s(w, x))$ into H;
  for all pairs (x, y), such that x or y is in N_w **do**
    Delete (x, y) from H;
    if $s(x, y) > 0$ **then**
      insert $(x, y, s(x, y))$ into H;

/* Output phase */
$E_S = C = \emptyset$
for all pairs (u, v) such that $u, v \in V_S$ **do**
  if $(A_{uv} > (|\Pi_{uv}| + 1)/2)$ **then**
    Add (u, v) to E_S;
    Add $-(a, b)$ to C for all $(a, b) \in \Pi_{uv} - A_{uv}$;
  else
    Add $+(a, b)$ to C for all $(a, b) \in A_{uv}$;
return representation $R = (S = (V_S, E_S), C)$

(10,000 ndoes and 21,000 edges), WordNet[3](76,853 nodes and 121,307 edges), and Facebook[4](14,562 nodes and 601,735 edges).

4.2 Results

4.2.1 How much graphs can be compressed? The results are in Figure 2. X-axis represents the number of nodes, and y-axis represents the number of edges. The original cost of the graphs increases from 30k to 400k, and the cost of the Greedy algorithm increases slowly from 10k to 110k. Notice that the Greedy algorithm always makes the best compression. It consistently produces a smaller compressed graph than the Randomized method does. When the size of the original graph becomes larger, the compression ratio also increases. The results indicate that a relatively large proportion of the original graph structure can be compressed without losing any information.

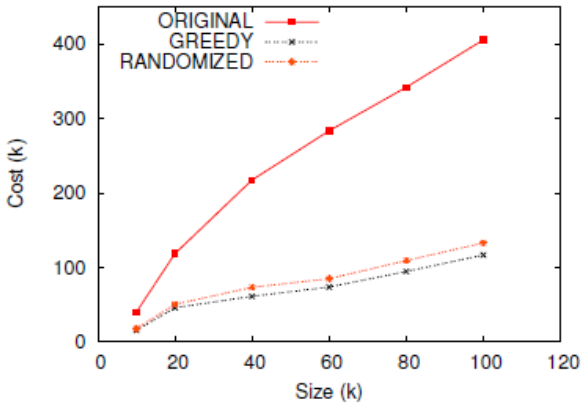

Fig. 2. Costs of representation. Source: [1].

4.2.2 How efficient is the algorithm? The results are shown in Figure 3. Observe that the running time of both methods climbs sharply when the size of the graphs increases from 10k to 40k, and moves slowly when the size increases from 40k to 100k. Furthermore, the Randomized method takes approximately half of the time of Greedy, since during each merging step in Greedy method, it first combines node pair that gives the maximum cost reduction, and then for each neighbor of the new (super)node, it updates the cost of all pairs that contains this neighbor.

[3] http://vlado.fmf.uni-lj.si/pub/networks/data/
[4] http://www.facebook.com/

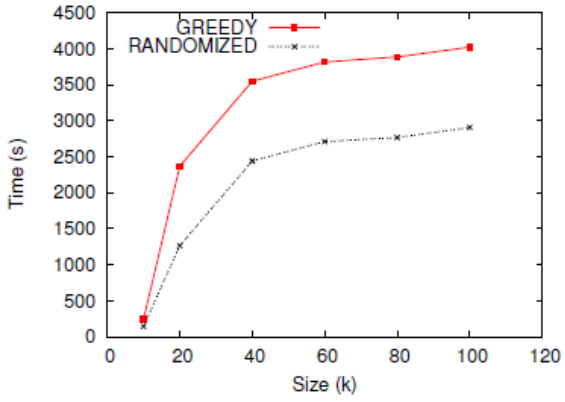

Fig. 3. Comparison of running time. Source: [1].

4.2.3 Compression performance comparison Finally, we see the MDL model compression performance compared with REF [12] and GRAC [13] on the four datasets in Figure 4. REF is a popular technique for web graph compression, and GRAC is a graph clustering algorithm. In Figure 4, the value of the y-axis is the cost of the compressed representation divided by the original cost. So, the lower the value is, the better the compression. Intuitively, the Greedy method constructs highly compressed graphs for all four datasets. Meanwhile, Randomized also performs better than REF and GRAC on all datasets.

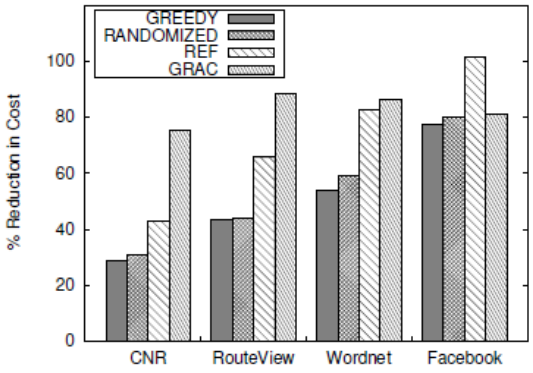

Fig. 4. Comparison of graph compression algorithms. Source: [1].

5 Future Work

In this section, I discuss some issues which deserve to be studied based on what I have learned from Article [1]

The first one is that, in articles [1] and [10], the authors propose compressed graph models for unlabeled and labeled graphs, respectively, while both of them ignore the edge weight. A novel graph compression model could produce a compressed graph from a labeled and weighted graph. In a compressed graph, the (super)edge weight represents the average weight between all pairs of nodes, which is different from the meaning of (super)edge weight [10] which shows the proportion of pairs of nodes directly connected with each other.

In this new model, the (super)edge weight shows the average connection between pairs of nodes in two (super)nodes. So, the real connection between each pair of nodes may be modified, either increased or decreased, or left unchanged. We can measure to which extent such information has been modified and then construct a compressed graph with fewer (super)nodes and less quality modification. Or, we can allow the user to set the error bound, and the resulting compressed graph would then consist of the fewest (super)nodes when the information error is within bounds.

The second one is that, considering many real-world application domains are naturally equipped with hierarchically organized taxonomies, we can construct a taxonomy-based compressed graph, which can be a meaningful and understandable abstraction of the original graph. We can allow users to control the resolutions of the compressed graph according to its hierarchical structure, and provide "drill-down" and "roll-up" abilities to navigate through summaries with different resolutions. Similar work has been done by Bonchi [14] in sequence compression.

The last issue is that, the model in article [1] is a space-saving approach to store a large graph, but it does not deal with the semantic meaning of the original graph. Article [10] tries to extract the underlying information based on the original graph. Another solution is based on the intermediary. For example, we can first convert a graph into a text, and then do text mining, constructing a smaller graph from the text instead of using the original graph.

References

1. Navlakha, S., Rastogi, R., Shrivastava, N.: Graph summarization with bounded error. In: SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM (2008) 419–432
2. Chen, C., Lin, C., Fredrikson, M., Christodorescu, M., Yan, X., Han, J.: Mining graph patterns efficiently via randomized summaries. In: 2009 Int. Conf. on Very Large Data Bases, Lyon, France, VLDB Endowment (August 2009) 742–753
3. Navlakha, S., Schatz, M., Kingsford, C.: Revealing Biological Modules via Graph Summarization. Presented at the RECOMB Systems Biology Satellite Conference. J. Comp. Bio. **16** (2009) 253–264

4. Chen, C., Yan, X., Zhu, F., Han, J., Yu, P.: Graph OLAP: Towards online analytical processing on graphs. In: ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, Washington, DC, USA, IEEE Computer Society (2008) 103–112
5. Fjällström, P.O.: Algorithms for graph partitioning: A Survey. In: Linköping Electronic Atricles in Computer and Information Science, 3. (1998)
6. Elsner, U.: Graph partitioning - a survey. Technical Report SFB393/97-27, Technische Universität Chemnitz (1997)
7. Gert, S.: The centrality index of a graph. Psychometrika **31**(4) (December 1966) 581–603
8. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E **69** (2004)
9. Fiedler, M., Borgelt, C.: Subgraph support in a single large graph. In: ICDM '07: Proceedings of the Seventh IEEE International Conference on Data Mining Workshops, Washington, DC, USA, IEEE Computer Society (2007) 399–404
10. Tian, Y., Hankins, R., Patel, J.: Efficient aggregation for graph summarization. In: SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM (2008) 567–580
11. Rissanen, J.: Modelling by the shortest data description. Automatica **14** (1978) 465–471
12. Boldi, P., Vigna, S.: The webgraph framework i: compression techniques. In: WWW '04: Proceedings of the 13th international conference on World Wide Web, New York, NY, USA, ACM (2004) 595–602
13. Dhillon, I., Guan, Y., Kulis, B.: A fast kernel-based multilevel algorithm for graph clustering. In: KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, New York, NY, USA, ACM (2005) 629–634
14. Bonchi, F., Castillo, C., Donato, D., Gionis, A.: Taxonomy-driven lumping for sequence mining. Data Mining and Knowledge Discovery (2009) 227–244