Network Compression by Node and Edge Mergers[*]

Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka

Department of Computer Science and HIIT, University of Helsinki, Finland
firstname.lastname@cs.helsinki.fi

Abstract. We give methods to compress weighted graphs (i.e., networks or BisoNets) into smaller ones. The motivation is that large networks of social, biological, or other relations can be complex to handle and visualize. Using the given methods, nodes and edges of a give graph are grouped to supernodes and superedges, respectively. The interpretation (i.e. decompression) of a compressed graph is that a pair of original nodes is connected by an edge if their supernodes are connected by one, and that the weight of an edge equals the weight of the superedge. The compression problem then consists of choosing supernodes, superedges, and superedge weights so that the approximation error is minimized while the amount of compression is maximized.

In this chapter, we describe this task as the 'simple weighted graph compression problem'. We also discuss a much wider class of tasks under the name of 'generalized weighted graph compression problem'. The generalized task extends the optimization to preserve longer-range connectivities between nodes, not just individual edge weights. We study the properties of these problems and outline a range of algorithms to solve them, with different trade-offs between complexity and quality of the result. We evaluate the problems and algorithms experimentally on real networks. The results indicate that weighted graphs can be compressed efficiently with relatively little compression error.

1 Introduction

Graphs and networks are used in numerous applications to describe relationships between entities, such as social relations between persons, links between web pages, flow of traffic, or interactions between proteins. We are also interested in conceptual networks called BisoNets which allow creative information exploration and support bisociative reasoning [2]. In many applications, including most BisoNets, relationships have weights that are central to any use or analysis of graphs: how frequently do two persons communicate or how much do they influence each other's opinions; how much web traffic flows from one page

[*] This chapter is a modified version of article "Compression of Weighted Graphs" in the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2011 [1].

to another or how many cars drive from one crossing to another; or how strongly does one protein regulate another one?

We describe models and methods for the compression of BisoNets (weighted graphs) into smaller ones that contain approximately the same information. In this process, also known as graph simplification in the context of unweighted graphs [3, 4], nodes are grouped to supernodes, and edges are grouped into superedges between supernodes. A superedge then represents all possible edges between two nodes, one from each of the conneceted supernodes.

This problem is different from graph clustering or partitioning where the aim is to find groups of strongly related nodes. In graph compression, nodes are grouped based on the similarity of their relationships to other nodes, not by their (direct) mutual relations.

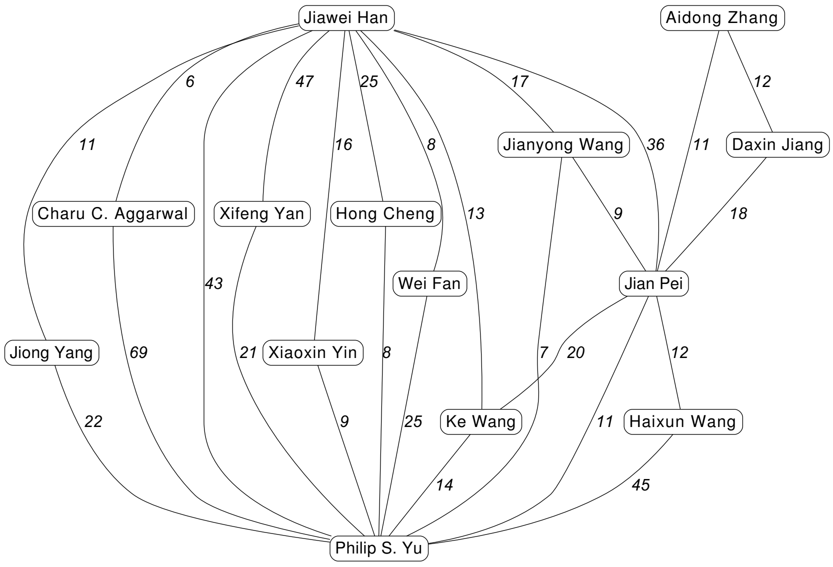
As a small example, consider the co-authorship social network in Figure 1a. It contains an excerpt from the DBLP Computer Science Bibliography[1], a subgraph containing *Jiawei Han* and *Philip S. Yu* and a dozen related authors. Nodes in this graph represent authors and edges represent co-authorships. Edges are weighted by the number of co-authored articles.

Compressing this graph by about 30% gives a simpler graph that highlights some of the inherent structure or roles in the original graph (Figure 1b). For instance, *Ke Wang* and *Jianyong Wang* have identical sets of co-authors (in this excerpt from DBLP) and have been grouped together. This is also an example of a group that would not be found by traditional graph clustering methods, since the two nodes grouped together are not directly connected. *Daxin Jiang* and *Aidong Zhang* have been grouped, but additionally the self-edge of their supernode indicates that they have also authored papers together.

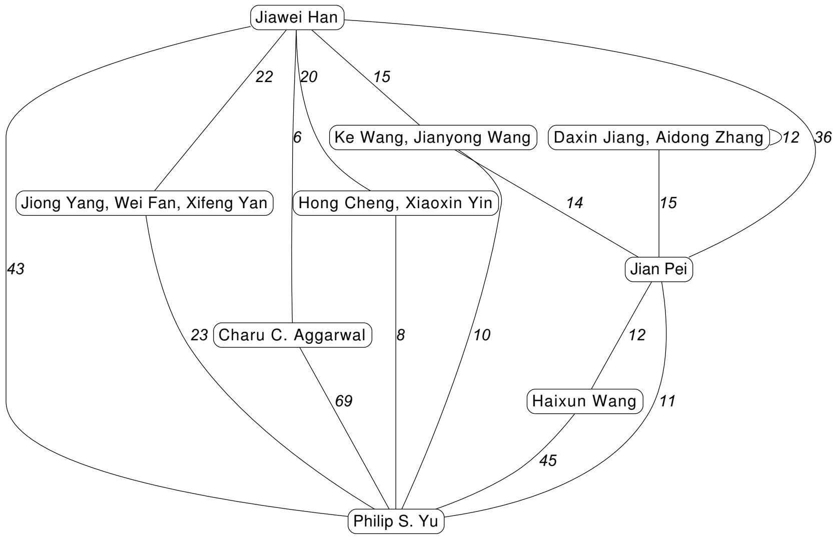
Groups that could not be obtained by the existing compression algorithms of [3, 4] can be observed among the six authors that (in this excerpt) only connect to *Jiawei Han* and *Philip S. Yu*. Instead of being all grouped together as structurally equivalent nodes, we have three groups that have different weight profiles. *Charu C. Aggarwal* is a group by himself, very strongly connected with *Philip S. Yu*. A second group includes *Jiong Yang*, *Wei Fan*, and *Xifeng Yan*, who are roughly equally strongly connected to both *Jiawei Han* and *Philip S. Yu*. The third group, *Hong Cheng* and *Xiaoxin Yin*, are more strongly connected to *Jiawei Han*. Such groups are not found with methods for unweighted graphs [3, 4].

In what we define as the *simple weighted graph compression problem*, the approximation error of the compressed graph with respect to original edge weights is minimized by assigning each superedge the mean weight of all edges it represents. For many applications on weighted graphs it is, however, important to preserve relationships between faraway nodes, too, not just individual edge weights. Motivated by this, we also introduce the *generalized weighted graph compression problem* where the goal is to produce a compressed graph that maintains connectivities across the graph: the best path between any two nodes should be approximately equally good in the compressed graph as it is in the original graph, but the path does not have to be the same.

[1] `http://dblp.uni-trier.de/`



(a) Before compression (14 nodes, 26 edges)



(b) After some compression (9 nodes, 16 edges)

Fig. 1. A neighborhood graph of *Jiawei Han* in the DBLP bibliography

Compressed weighted graphs can be utilized in a number of ways. Graph algorithms can run more efficiently on a compressed graph, either by considering just the smaller graph consisting of supernodes and superedges, or by decompressing parts of it on the fly when needed. An interesting possibility is to provide an interactive visualization of a graph where the user can adjust the abstraction level of the graph on the fly.

The rest of this chapter is organized as follows. We formulate and analyze the weighted graph compression problems in Section 2. Related work is briefly reviewed in Section 3. We give algorithms for the weighted graph compression problems in Section 4 and evaluate them experimentally in Section 5. Section 6 contains concluding remarks.

2 Problem Definition

The goal is to compress a given weighted graph (BisoNet) into a smaller one. We address two variants of this problem. In the first one, the simple weighted graph compression problem, the goal is to produce a compressed graph that can be decompressed into a graph similar to the original one. In the second variant, the generalized weighted graph compression problem, the decompressed graph should approximately preserve the strengths of connections between all nodes.

2.1 Weighted and Compressed Graphs

We start by defining concepts and notations common to both problem variants of weighted graph compression.

Definition 1. *A* weighted graph *is a triple $G = (V, E, w)$, where V is a set of vertices (or nodes), $E \subset V \times V$ is a set of edges, and $w : E \to \mathbb{R}^+$ assigns a (non-negative) weight to each edge $e \in E$. For notational convenience, we define $w(u, v) = 0$ if $(u, v) \notin E$.*

In this chapter, we actually assume that graphs and edges are undirected, and in the sequel use notations such as $\{u, v\} \in V \times V$ in the obvious way. The definitions and algorithms can, however, be easily adapted for the directed case. In the compressed graph we also allow self-edges, i.e., an edge from a node back to itself. The following definition of a compressed graph largely follows the definition of graph summarization for the unweighted case [3]. The essential role of weights will be defined after that.

Definition 2. *A weighted graph $S = (V', E', w')$ is a* compressed representation (or compressed graph) of *G if $V' = \{v'_1, \ldots, v'_n\}$ is a partition of V (i.e., $v'_i \subset V$ for all i, $\cup_i v'_i = V$, and $v'_i \cap v'_j = \emptyset$ for all $i \neq j$). The nodes $v' \in V'$ are also called* supernodes*, and edges $e' \in E'$ are also called* superedges.

We use the notation $com : V \to V'$ to map original nodes to the corresponding supernodes: $com(u) = v'$ if and only if $u \in v' \in V'$.

The idea is that a supernode represents all original nodes within it, and that a single superedge represents all possible edges between the corresponding original nodes, whether they exist in G or not. Apparently, this may cause structural errors of two types: a superedge may represent edges that do not exist in the original graph, or edges in the original graph are not represented by any superedge. (Our algorithms in Section 4 only commit the first kind of errors, i.e., they will not miss any edges, but may introduce new ones.) In addition, edge weights may have changed in compression. We will next formalize these issues using the concepts of decompressed graphs and graph dissimilarity.

Definition 3. *Given G and S as above, the* decompressed graph *$dec(S)$ of S is a weighted graph $dec(S) = (V, E'', w'')$ such that $E'' = \{\{u,v\} \in V \times V \mid \{com(u), com(v)\} \in E'\}$ and $w''(\{u,v\}) = w'(\{com(u), com(v)\})$. (By the definition of compressed representation, $V = \cup_{i=1}^{n} V_i'$.)*

In other words, a decompressed graph has the original set of nodes V, and there is an edge between two nodes exactly when there is a superedge between the corresponding supernodes. The weight of an edge equals the weight of the corresponding superedge.

Definition 4. *Given G and S as above, the* compression ratio *of S (with respect to the original graph G) is defined as $cr(S) = \frac{|E'|}{|E|}$.*

The compression ratio measures how much smaller the compressed graph is. The number of supernodes vs. original nodes is not included in the definition since nodes are actually not compressed, in the sense that their identities are preserved in the supernodes and hence no space is saved. They are also always completely recovered in decompression.

2.2 Simple Weighted Graph Compression

Compression ratio does not consider the amount of errors introduced in edges and their weights. This issue is addressed by a measure of dissimilarity between graphs. We first present a simple distance measure that leads to the simple weighted graph compression problem.

Definition 5. *The* simple distance *between two graphs $G_a = (V, E_a, w_a)$ and $G_b = (V, E_b, w_b)$, with an identical set of nodes V, is*

$$dist_1(G_a, G_b) = \sqrt{\sum_{\{u,v\} \in V \times V} (w_a(\{u,v\}) - w_b(\{u,v\}))^2}. \tag{1}$$

This distance measure has an interpretation as the Euclidean distance between G_a and G_b in a space where each pair of nodes $\{u,v\} \in V \times V$ has its own dimension. Given the distance definition, the dissimilarity between a graph G and its compressed representation S can then be defined simply as $dist_1(G, dec(S))$.

Edges introduced by the compression/decompression process are considered in this measure, since edge weight $w(\{u,v\}) = 0$ if edge $\{u,v\}$ is not in G.

The distance can be seen as the cost of compression, whereas the compression ratio represents the savings. Our goal is to produce a compressed graph which optimizes the balance between these two. In particular, we will consider the following form of the problem.

Definition 6. *Given a weighted graph G and a compression ratio cr, $0 < cr < 1$, the* simple weighted graph compression problem *is to produce a compressed representation S of G with $cr(S) \leq cr$ such that $dist_1(G, dec(S))$ is minimized.*

Other forms can be just as useful. One obvious choice would be to give a maximum distance as parameter, and then seek for a minimum compression ratio. In either case, the problem is complex, as the search space consists of all partitions of V. However, the compression ratio is non-increasing and graph distance non-decreasing when nodes are merged to supernodes, and this observation can be used to devise heuristic algorithms for the problem, as we do in Section 4.

2.3 Generalized Weighted Graph Compression

We next generalize the weighted graph compression problem. In many applications, it is not the individual edge weights but the overall connectivity between nodes that matters, and we propose a model that takes this into account. The model is based on measuring the best paths between nodes, and trying to preserve these qualities. We start with some preliminary definitions and notations.

Definition 7. *Given a graph $G = (V, E, w)$, a* path *P is a set of edges $P = \{\{u_1, u_2\}, \{u_2, u_3\}, \ldots, \{u_{k-1}, u_k\}\} \subset E$. We use the notation $u_1 \overset{P}{\rightsquigarrow} u_k$ to say that P is a path between u_1 and u_k, and that u_1 and u_k are the endnodes of P.*

The definition of how good a path is and which is the best one depends on the kind of graph and the application. For the sake of generality, we parameterize our formulation by a path quality function q. For example, in a flow graph where edge weights are capacities of edges, path quality q can be defined as the maximum flow through the path (i.e., as the minimum edge weight on the path). In a probabilistic or uncertain graph where edge weights are probabilities that the edge exists, q often is defined as the probability that the path exists (i.e., as the product of the edge weights). Without loss of generality, we assume that the value of any path quality function is positive, that a larger value of q indicates better quality, and that q is monotone in the sense that a path with a cycle is never better than the same path without the cycle. We also parameterize the generalized definition by a maximum path length λ. The goal of generalized weighted graph compression will be to preserve all pairwise connectivities of length at most λ.

Definition 8. *Given a weighted graph $G = (V, E, w)$, a path quality function q, and a positive integer λ, the* λ-connection *between a pair of nodes u and v is defined as*

$$Q_\lambda(u,v;G) = \begin{cases} \max_{P \subset E: u \overset{P}{\rightsquigarrow} v, |P| \leq \lambda} q(P) & \textit{if such } P \textit{ exists} \\ 0 & \textit{otherwise,} \end{cases}$$

i.e., as the quality of the best path, of length at most λ, between u and v. If G is obvious in the context, we simply write $Q_\lambda(u,v)$.

Definition 9. *Let G_a and G_b be weighted graphs with an identical set V of nodes, and let λ be a positive integer and q a path quality function as defined above. The* generalized distance between G_a and G_b (with respect to λ and q) *is*

$$dist_\lambda(G_a, G_b) = \sqrt{\sum_{\{u,v\} \in V \times V} (Q_\lambda(u,v;G_a) - Q_\lambda(u,v;G_b))^2}. \qquad (2)$$

Definition 10. *Given a weighted graph G and a compression ratio cr, $0 < cr < 1$, the* generalized weighted graph compression problem *is to produce a compressed representation S of G with $cr(S) \leq cr$ such that $dist_\lambda(G, dec(S))$ is minimized.*

The simple weighted graph compression problem defined earlier is an instance of this generalized problem with $\lambda = 1$ and $q(\{e\}) = w(e)$. In this chapter, we will only consider the two extreme cases with $\lambda = 1$ and $\lambda = \infty$. For notational convenience, we often write $dist(\cdot)$ instead of $dist_\lambda(\cdot)$ if the value of λ is not significant.

2.4 Optimal Superedge Weights and Mergers

Given a compressed graph structure, it is easy to set the weights of superedges to optimize the simple distance measure $dist_1(\cdot)$. Each pair $\{u,v\} \in V \times V$ of original nodes is represented by exactly one pair $\{u',v'\} \in V' \times V'$ of supernodes, including the cases $u = v$ and $u' = v'$. In order to minimize Equation 1, given the supernodes V', we need to minimize for each pair $\{u',v'\}$ of supernodes the sum $\sum_{\{u,v\} \in u' \times v'} (w(\{u,v\}) - w'(\{u'v'\}))^2$. This sum is minimized when the superedge weight is the mean of the original edge weights (including "zero-weight edges" for those pairs of nodes that are not connected by an edge):

$$w'(\{u',v'\}) = \frac{\sum_{\{u,v\} \in u' \times v'} w(\{u,v\})}{|u'|\,|v'|}, \qquad (3)$$

where $|x|$ is the number of original nodes in supernode x.

The compression algorithms that we propose below work in an incremental, often greedy fashion, merging two supernodes at a time into a new supernode (following the ideas of references [3, 4]). The merge operation that these algorithms use is specified in Algorithm 1. It takes a graph and two of its nodes as parameters, and it returns a graph where the given nodes are merged into one and the edge weights of the new supernode are set according to Equation 3. Line 6 of the merge operation sets the weight of the self-edge for the supernode.

When $\lambda = 1$, function $W(x, y)$ returns the sum of weights of all original edges between x and y using their mean weight $Q_1(\{x, y\}; S)$. The weight of the self-edge is then zero and the edge non-existent if neither u nor v has a self-edge and if there is no edge between u and v.

Algorithm 1. merge(u, v, S)

```
Input: Nodes $u$ and $v$, and a compressed graph $S = (V, E, w)$ s.t. $u, v \in V$
Output: A compressed graph $S'$ obtained by merging $u$ and $v$ in $S$
 1: $S' \leftarrow S$ {i.e., $(V', E', w') \leftarrow (V, E, w)$}
 2: $z \leftarrow \{u \cup v\}$
 3: $V' \leftarrow V' \setminus \{u, v\} \cup \{z\}$
 4: for all $x \in V$ s.t. $u \neq x \neq v$, and $\{u, x\}$ or $\{v, x\} \in E$ do
 5:     $w'(\{z, x\}) = \frac{|u| Q_\lambda(\{u,x\};S) + |v| Q_\lambda(\{v,x\};S)}{|u|+|v|}$
 6: $w'(\{z, z\}) = \frac{W(u,u)+W(v,v)+W(u,v)}{|z|(|z|-1)/2}$
 7: return $S'$
 8: function $W(x, y)$:
 9: if $x \neq y$ then
10:     return $Q_\lambda(\{x, y\}; S)|x||y|$
11: else
12:     return $Q_\lambda(\{x, x\}; S)|x|(|x| - 1)/2$
```

Setting superedge weights optimally is much more complicated for the generalized distance (Equation 2) when $\lambda > 1$: edge weights contribute to best paths and therefore distances up to λ hops away, so the distance cannot be optimized in general by setting each superedge weight independently. We use the merge operation of Algorithm 1 as an efficient, approximate solution also in these cases, and leave better solutions for future work.

2.5 Bounds for Distances between Graphs

Our compression algorithms produce the compressed graph S by a sequence of merge operations, i.e., as a sequence $S_0 = G, S_1, \ldots, S_n = S$ of increasingly compressed graphs. Since the distance function $dist(\cdot)$ is a metric and satisfies the triangle inequality (recall its interpretation as Euclidian distance), the distance of the final compressed graph S from the original graph G can be upper-bounded by

$$dist(G, dec(S)) \leq \sum_{i=1}^{n} dist(dec(S_{i-1}), dec(S_i)).$$

An upper bound for the distance between two graphs can be obtained by considering only the biggest distance over all pairs of nodes. Let G_a and G_b be weighted graphs with an identical set V of nodes, and denote the maximum distance for any pair of nodes by

$$d_{\max}(G_1, G_2) = \max_{\{u,v\} \in V \times V} |(Q_\lambda(u, v; G_a) - Q_\lambda(u, v; G_b))|.$$

We now have the following bound:

$$\begin{aligned} dist(G_1, G_2) &\leq \sqrt{\sum_{\{u,v\} \in V \times V} d_{\max}(G_1, G_2)^2} \\ &\propto d_{\max}(G_1, G_2). \end{aligned} \tag{4}$$

This result can be used by compression algorithms to bound the effects of potential merge operations.

2.6 A Bound on Distances between Nodes

We now derive an upper bound for $d_{\max}(S, S')$ for the case where $S' =$ merge(u, v, S) for some nodes u and v in V (cf. Algorithm 1). Let $d_{\max}(u, v; S)$ be the maximum difference of weights between any two edges merged together as the result of merging u and v:

$$\begin{aligned} &d_{\max}(u, v; S) = \\ &\quad \max\{ \max_{x:\{u,x\} \text{ or } \{v,x\} \in E} (|Q_\lambda(u, x; S) - Q_\lambda(v, x; S)|), \\ &\qquad |Q_\lambda(u, u; S) - Q_\lambda(v, v; S)|, \\ &\qquad |Q_\lambda(u, u; S) - Q_\lambda(u, v; S)|, \\ &\qquad |Q_\lambda(v, v; S) - Q_\lambda(u, v; S)| \ \}. \end{aligned} \tag{5}$$

The first element is the maximum over all edges to neighboring nodes x, and the rest are the differences between edges that are merged into the self-edge.

For $\lambda = 1$ it is fairly obvious that we have the bound

$$d_{\max}(S, \text{merge}(u, v, S)) \leq d_{\max}(u, v; S), \tag{6}$$

since all effects of merge operations are completely local to the edges adjacent to the merged nodes. The situation is more complicated for $\lambda = \infty$ since a merger can also affect arbitrary edges. Luckily, many natural path quality functions q have the property that a change in the weight of an edge (from $w(e)$ to $w'(e)$) changes the quality of the whole path (from $q(P)$ to $q'(P)$) at most as much as it changes the edge itself:

$$\frac{|q(P) - q'(P)|}{q(P)} \leq \frac{|w(e) - w'(e)|}{w(e)}.$$

Path quality functions q that have this property include the sum of edge weights (e.g., path length), product of edge weights (e.g., path probability), minimum edge weight (e.g., maximum flow), maximum edge weight, and average edge weight. Based on this property, we can infer that the biggest distance after merging u and v will be seen on the edges connecting u, v and their neighbors, i.e., that the bound of Equation 6 holds also for $\lambda = \infty$ for many usual path quality functions.

Based on Equations 4 and 6, we have a fast way to bound the effect of merging any two nodes. We will use this bound in some of our algorithms.

3 Related Work

Graph compression as presented in this chapter is based on merging nodes that have similar relationships to other entities i.e., that are structurally most equivalent — a classic concept in social network analysis [5]. Structural equivalence and many other types of relations between (super)nodes have been considered in social networks under block modeling (see, e.g., [6]), where the goal is both to identify supernodes and to choose among the different possible types of connections between them. Our approach (as well as that of references [3, 4], see below) uses only two types: "null" (no edges) and "complete" (all pairs are connected), as these seem to be best suited for compression.

Graph compression has recently attracted new interest. The work most closely related to ours is by Navlakha et al. [3] and Tian et al. [4], who independently proposed to construct graph summaries of unweighed graphs by grouping nodes and edges to supernodes and superedges. We generalize these approaches in two important and related directions: to weighted graphs, and to long-range, indirect (weighted) connections between nodes.

Both above-mentioned papers also address issues we do not consider here. Navlakha et al. [3] propose a representation which has two parts: one is a graph summary (in our terminology, an unweighted compressed graph), the other one is a set of edge corrections to fix the errors introduced by mergers of nodes and edges to superedges. Tian et al. [4] consider labeled graphs with categorical node and edge attributes, and the goal is to find relatively homogeneous supernodes and superedges. This approach has been generalized by Zhang et al. [7] to numerical node attributes which are then automatically categorized. They also addressed interactive drill-down and roll-up operations on graphs. Tian et al. used both top-down (divisive) and bottom-up (agglomerative) algorithms, and concluded that top-down methods are more practical in their problem [4], whereas Navlakha et al. had the opposite experience [3]. This difference is likely due to different use of node and edge labels. The methods we propose work bottom-up since we have no categorical attributes to guide a divisive approach like Tian et al. had.

Unweighted graph compression techniques have been used to simplify graph storage and manipulation. For example, Chen et al. [8] successfully applied a graph compression method to reduce the number of embeddings when searching frequent subgraphs in a large graph. Navlakha et al. [9] revealed biological modules with the help of compressed graphs. Furthermore, Chen et al. [10] incorporated the compressed graph notion with a generic topological OLAP framework to realize online graph analysis.

There are many related but subtly different problems. Graph partitioning methods (e.g. [11, 12]) aim to find groups of nodes that are more strongly connected to each other than to nodes in other groups. Extraction of a subgraph, whether based on a user query (e.g. [13, 14]) or not (e.g., [15–17]) produces a smaller graph by just throwing out edges and nodes. Web graph compression algorithms aim to produce as compact a representation of a graph as possible, in different formats (e.g., [18, 19]). For more related work, we refer to the good overviews given in references [3, 4]. A wider review of network abstraction techniques is available in [20].

4 Algorithms

We next propose a series of algorithms for the weighted graph compression problem. All of the proposed algorithms work more or less in a greedy fashion, merging two (super)nodes and their edges at a time until the specified compression rate is achieved. All these algorithms have the following input and output:

Input: weighted graph $G = (V, E, w)$, compression ratio cr $(0 < cr < 1)$, path quality function q, and maximum path length $\lambda \in \mathbb{N}$.

Output: compressed weighted graph $S = (V', E', w')$ with $cr(S) \leq cr$, such that $dist(G, dec(S))$ is minimized.

Brute-force greedy algorithm. The brute-force greedy method (Algorithm 2) computes the effects of all possible pairwise mergers (Line 4) and then performs the best merger (Line 5), and repeats this until the requested compression rate is achieved. The algorithm generalizes the greedy algorithm of Navlakha et al. [3] to distance functions $dist_\lambda(\cdot)$ that take the maximum path length λ and the path quality function q as parameters.

Algorithm 2. Brute-force greedy search

```
1: $S \leftarrow G$ {i.e., $(V', E', w') \leftarrow (V, E, w)$}
2: while $cr(S) > cr$ do
3:     for all pairs $\{u, v\} \in V' \times V'$ do {(*)}
4:         $d_{\{u,v\}} \leftarrow dist(G, dec(merge(u, v, S)))$
5:     $S \leftarrow merge(\arg\min_{\{u,v\}} d_{\{u,v\}}, S)$
6: return $S$
(*) 2-hop optimization can be used, see text.
```

The worst-case time complexity for simple weighted graph compression is $O(|V|^4)$, and for generalized compression $O(|V|^3|E| \log |V|)$. We omit the details for brevity.

2-hop optimization. The brute-force method, as well as all other methods we present here, can be improved by 2-hop optimization. Instead of arbitrary pairs of nodes, the 2-hop optimized version only considers u and v for a potential merger if they are exactly two hops from each other. Since 2-hop neighbors have a shared neighbor that can be linked to the merged supernode with a single superedge, some compression may result. The 2-hop optimization is safe in the sense that any merger by Algorithm 1 that compresses the graph involves 2-hop neighbors.

The time saving by 2-hop optimization can be significant: for the brute-force method, for instance, there are approximately $O(deg\,|E|)$ feasible node pairs with the optimization, where *deg* is the average degree, instead of the $O(|V|^2)$ pairs in the unoptimized algorithm.

For the randomized methods below, a straight-forward implementation of 2-hop optimization by random walk has a nice property. Assume that one node has been chosen, then find a random pair for it by taking two consequtive random hops starting from the first node. Now 2-hop neighbors with many shared neighbors are more likely to get picked, since there are several 2-hop paths to them, and a merger between nodes with many shared neighbors will lead to better compression. A uniform selection among all 2-hop neighbors does not have this property.

Thresholded algorithm. We next propose a more practical algorithmic alternative, the thresholded method (Algorithm 3). It iterates over all pairs of nodes and merges all pairs (u, v) such that $d_{\max}(u, v; S) \leq T_i$ (Lines 5–6). The threshold value T_i is increased iteratively in a heuristic manner whenever no mergers can be done with the current threshold (Lines 2 and 4).

Algorithm 3. Thresholded algorithm

```
1: for all $0 \leq i \leq K$ do
2:     $T_i \leftarrow 2^{-K+i}$
3: $S \leftarrow G$ {i.e., $(V', E', w') \leftarrow (V, E, w)$}
4: for all $i = 0, \ldots, K$ do
5:     while there exists a pair $\{u, v\} \in V' \times V'$ such that $d_{\max}(u, v; S) \leq T_i$ do {(*)}
6:         $S \leftarrow merge(u, v, S)$
7:         if $cr(S) \leq cr$ then
8:             return $S$
```

(*) 2-hop optimization can be used, see text.

Different schemes for setting the thresholds would give different results and time complexity. The heuristic we have used has $K = 20$ exponentially growing steps and aims to produce relatively high-quality results faster than the brute-force method. Increasing the threshold in larger steps would give a faster method, but eventually a random compression (cf. Algorithm 5 below). We will give better informed, faster methods below.

The time complexity is $O(|V|^4)$ for the simple and $O(|V|^4 + |V|^2|E| \log |V|)$ for the generalized problem. These are upper bounds for highly improbable worst cases, and in practice the algorithm is much faster. See experiments in Section 5 for details on real world performance.

Randomized semi-greedy algorithm. The next algorithm is half random, half greedy (Algorithm 4). In each iteration, it first picks a node v at random (Line 3). Then it chooses node u so that the merge of u and v is optimal with respect to $d_{\max}(u, v; S)$ (Line 6). This algorithm, with 2-hop optimization, is a generalized version of the randomized algorithm of Navlakha et al. [3].

The worst-case time complexity of the algorithm is $O(|V|^3)$ for the simple and $O(|V|^2|E| \log |V|)$ for the generalized problem.

Random pairwise compression. Finally, we present a naive, random method which simply merges pairs of nodes at random without any aim to produce

Algorithm 4. Randomized semi-greedy algorithm

```
1: $S \leftarrow G$ {i.e., $(V', E', w') \leftarrow (V, E, w)$}
2: while $cr(S) > cr$ do
3:     randomly choose $v \in V'$
4:     for all nodes $u \in V'$ do {(*)}
5:         $d_u \leftarrow d_{\max}(v, u; S)$
6:     $S \leftarrow merge(\arg\min_u d_u, v, S)$
7: return $S$
```

(*) 2-hop optimization can be used, see text.

Algorithm 5. Random pairwise compression

```
1: $S \leftarrow G$ {i.e., $(V', E', w') \leftarrow (V, E, w)$}
2: while $cr(S) > cr$ do
3:     randomly choose $\{u, v\} \in V' \times V'$ {(*)}
4:     $S \leftarrow merge(u, v, S)$
5: return $S$
```

(*) 2-hop optimization can be used, see text.

a good compression (Algorithm 5). The uninformed random method provides a baseline for the quality of other methods that make informed decisions about mergers.

The time complexity of the random algorithm is $O(|V|^2)$ for the simple and $O(|V||E| \log |V|)$ for the generalized problem. The random algorithm is essentially the fastest possible compression algorithm that uses pairwise mergers. It therefore provides a baseline (lower bound) for runtime comparisons.

Interactive compression. Thanks to the simple agglomerative structure of the methods, all of them lend themselves to interactive visualization of graphs where the abstraction level can be adjusted dynamically. This simply requires that the merge operations save the hierarchical composition of the supernodes produced. A drill-down operation then corresponds to backtracking merge operations, and a roll-up operation corresponds to mergers.

5 Experiments

We next present experimental results on the weighted graph compression problem using algorithms introduced in the previous section and real data sets. With these experiments we aim to address the following questions. (1) How well can weighted graphs be compressed: what is the trade-off between compression (lower number of edges) and distance to the original graph? (2) How do the different algorithms fare in this task: how good are the results they produce? (3) What are the running times of the algorithms? And, finally: (4) How does compression affect the use of the graph in clustering?

5.1 Experimental Setup

We extracted test graphs from the biological Biomine database[2] and from a co-authorship graph compiled from the DBLP computer science bibliography. Edge weights are in $[0, 1]$, and the path quality function is the product of weights of the edges in the path. Below we briefly describe how the datasets were obtained.

A set of 30 connection graphs, each consisting of around 1000 nodes and 2411 to 3802 edges (median 2987 edges; average node degree 2.94) was used in most of the tests. These graphs were obtained as connection graphs between three sets of related genes (different gene sets for each of the 30 replicates) so that they contain some non-trivial structure. We mostly report mean results over all 30 graphs.

A set of 30 smaller graphs was used for tests with the time-consuming brute-force method. These graphs have 50 nodes each and 76 to 132 edges (median 117 edges; average node degree 2.16).

Two series of increasingly larger graphs were used to compare the scalability of the methods. The sizes in one series range from 1000 to 5000 nodes and from 2000 to 17000 edges, and in the other series from 10 000 to 200 000 nodes and about 12 000 to 400 000 edges.

The algorithms were implemented in Java, and all the experiments were run on a standard PC with 4 GB of main memory and an Intel Core 2 Duo 3.16 GHz processor.

5.2 Results

Compressibility of weighted graphs. Figures 2a and 2b give the distance between the compressed and original graphs as a function of the compression ratio. For better interpretability, the distance is represented as the root mean square error (RMSE) over all pairs of nodes. Overall, the distances are small. Compression to half of the original size can be achieved with errors of 0.03 ($\lambda = 1$) or 0.06 ($\lambda = \infty$) per node pair. Especially for $\lambda = \infty$ graphs compress very nicely.

Comparison of algorithms. Figure 2c complements the comparison with results for the smaller graphs, and now including the brute-force method ($\lambda = 1$). The brute-force method clearly produces the best results (but is very slow as we will see shortly). Note also how small graphs are relatively harder to compress and the distances are larger than for the standard set of larger graphs.

The thresholded method is almost as good for compression ratios 0.8-0.9 but the gap grows a bit for smaller compression ratios. The semi-greedy version, on the other hand, is not as good with the larger compression ratios, but has a relatively good performance with smaller compression ratios. The random method is consistently the worst. A few early bad mergers already raise the distance for high compression ratios. Experiments on larger graphs could not be run with the brute force methods.

[2] `http://biomine.cs.helsinki.fi`

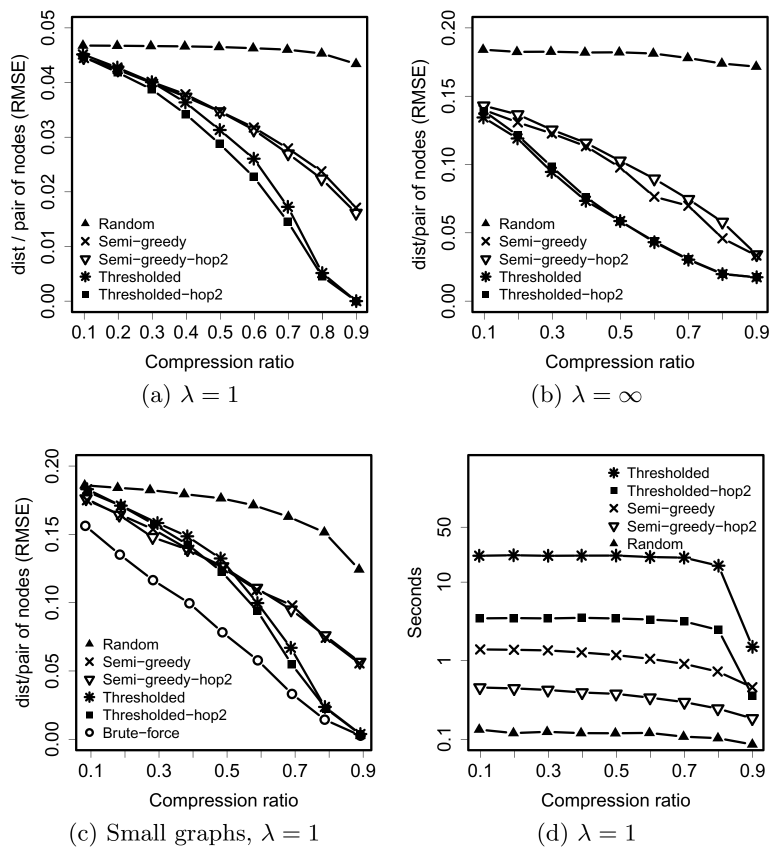


Fig. 2. (a)-(c): Distance between the compressed and the original graph as a function of compression ratio. (d): Running times of algorithms.

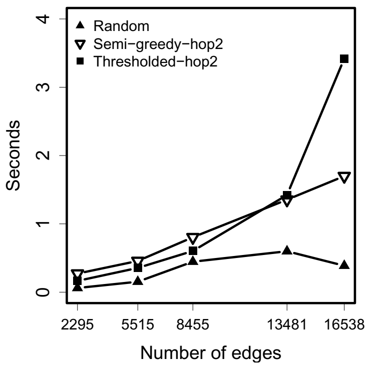
Efficiency of algorithms. Mean running times of the algorithms (except brute-force, see below) over the 30 standard graphs are shown in Figure 2d. The differences in the running times are big between the methods, more than two orders of magnitude between the extremes.

The 2-hop-optimized versions are an order of magnitude faster than the un-optimized versions while the results were equally good (cf. Figure 2c). 2-hop optimization thus very clearly pays off.

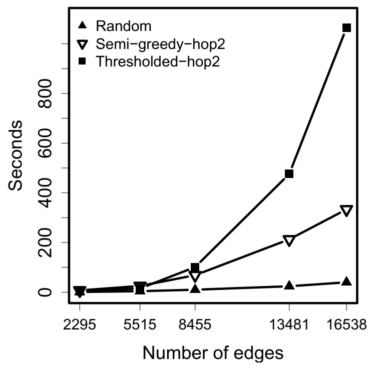
The brute-force method is very slow compared to the other methods (results not shown). Its running times for the small graphs were 1.5–5 seconds with $\lambda = 1$ where all other methods always finished within 0.4 seconds. With $\lambda = \infty$, the brute-force method spent 20–80 seconds whereas all other methods used less than 0.5 second.

Running times with $\lambda = \infty$ are larger than with $\lambda = 1$ by an order of magnitude, for the semi-greedy versions by two orders of magnitude (not shown).

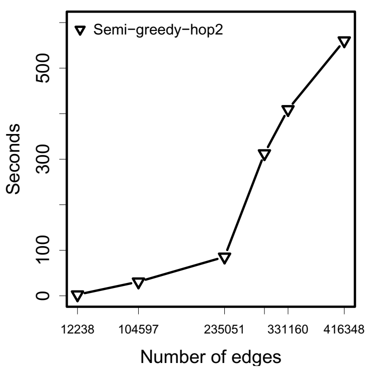
We evaluated the effect of graph size on running times of the three fastest algorithm, using the series of increasingly large graphs and a fixed compression ratio 0.8



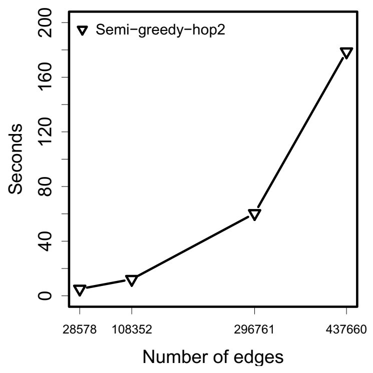
(a) Biological graphs, $\lambda = 1$



(b) Biological graphs, $\lambda = \infty$



(c) Large biological graphs, $\lambda = 1$



(d) Large co-authorship graphs, $\lambda = 1$

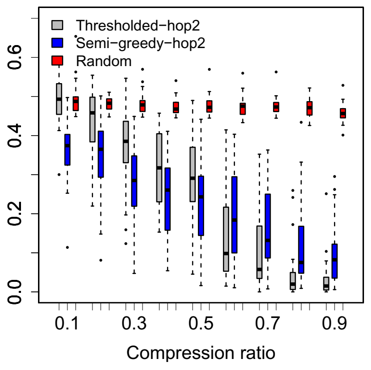
Fig. 3. Running times of weighted graph compression algorithms on graphs of various sizes from different sources

(Figures 3a and 3b). For $\lambda = 1$ the random method should be linear (and deviations are likely due to random effects). The thresholded method seems in practice approximately quadratic as is to be expected: for any value of λ, it will iterate over all pairs of nodes. The semi-greedy algorithm has a much more graceful behavior, even if slightly superlinear. Relative results are similar for $\lambda = \infty$.

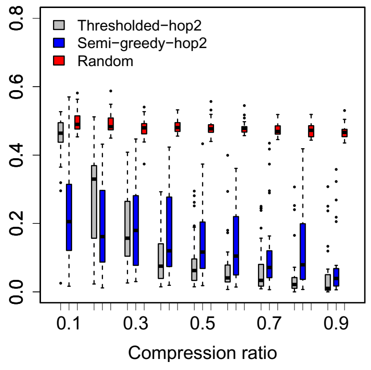
Additional scalability experiments were run with larger graphs from both biological and co-authorship domains, using the semi-greedy algorithm with 2-hop optimization, compression ratio $cr = 0.8$, and $\lambda = 1$ (Figures 3c and 3d). The algorithm compressed graphs of upto 400000 edges in less than 10 minutes (biology) or in less than 3 minutes (co-authorship). The biological graphs contain nodes with high degrees, and this makes the compression algorithms slower.

Effect of compression on node clustering results. We next study how errors introduced by weighted graph compression affect methods that work on graphs. As a case study, we consider node clustering and measure the difference of clusters in the original graph vs. clusters in (the decompressed version of) the compressed graph.

We applied the k-medoids clustering algorithm on the 30 standard graphs. We set $k = 3$, corresponding to the three gene groups used to obtain the graphs. The proximity between two nodes was computed as the product of weights (probabilities) of edges on the best path. We measure the difference between clusterings by the Rand index (more exactly, by 1 minus Rand index). In other words, we measure the fraction of node pairs that are clustered inconsistently in the clusterings, i.e., assigned to the same cluster in one graph and to different clusters in the other graph.



(a) $\lambda = 1$



(b) $\lambda = \infty$

Fig. 4. Effect of compression on node clustering. Y-axis is the fraction of node pairs clustered inconsistently in the original and the compressed graph.

According to the results, the thresholded and semi-greedy compression methods can compress a weighted graph with little effect on node clustering (Figure 4). The effect is small especially when $\lambda = \infty$, where the inconsistency ratio is less than 0.1 (the thresholded method) or 0.3 (the semi-greedy method) for a wide range of compression ratios. The effects of the thresholded and semi-greedy versions are larger for $\lambda = 1$, especially when the compression ratio cr becomes smaller. This is because a clustering based solely on immediate neighborhoods is more sensitive to individual edge weights, whereas $Q_\infty(\cdot)$ can find a new best path elsewhere if an edge on the current best path is strongly changed.

Surprisingly, the semi-greedy method performs best in this comparison with compression ratio $cr \leq 0.2$. With $\lambda = \infty$ even an aggressive compression introduced relatively little changes to node clustering. In the other extreme, clusters found in randomly compressed graphs are quite—but not completely—different from the clusters found in the original graph. Close to 50% of pairs are clustered inconsistently, whereas a random clustering of three equally sized clusters would have about 2/3 inconsistency.

6 Conclusions

We have discussed the problem of compressing BisoNets or, in more general, weighted graphs. We derived bounds for it and gave algorithms and experimental results on real datasets. We presented two forms of the problem: a simple one, where the compressed graph should preserve edge weights, and a generalized one, where the compressed graph should preserve strengths of connections of up to λ hops. The generalized form may be valuable especially for graph analysis algorithms that rely more on strengths of connections than individual edge weights.

The results indicate the following. (1) BisoNets can be compressed quite a lot with little loss of information. (2) The generalized weighted graph compression problem is promising as a pre-processing step for computationally complex graph analysis algorithms: clustering of nodes was affected very little by generalized compression. (3) BisoNets can be compressed efficiently. E.g., the semi-greedy method processed a 16 000 edge graph in 2 seconds.

An additional good property of the methods is that compressed graphs are graphs, too. This gives two benefits. First, some graph algorithms can be applied directly on the compressed graph with reduced running times. Second, representing graphs as graphs is user-friendly. The user can easily tune the abstraction level by adjusting the compression ratio (or the maximum distance between the compressed and the original graph). This can also be done interactively to support visual inspection of a graph.

There are several directions in which this work can be developed further. Different merge operations may be considered, also ones that remove edges. More efficient algorithms can be developed for even better scalability to large graphs. It could be useful to modify the methods to guarantee a bounded edge-wise or node pair-wise error, or to also accommodate categorical labels.

Acknowledgements. This work has been supported by the Algorithmic Data Analysis (Algodan) Centre of Excellence of the Academy of Finland (Grant 118653) and by the European Commission under the 7th Framework Programme FP7-ICT-2007-C FET-Open, contract no. BISON-211898.

References

1. Toivonen, H., Zhou, F., Hartikainen, A., Hinkka, A.: Compression of weighted graphs. In: The 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), San Diego, CA, USA (2011)
2. Kötter, T., Berthold, M.R.: From Information Networks to Bisociative Information Networks. In: Berthold, M.R. (ed.) Bisociative Knowledge Discovery. LNCS (LNAI), vol. 7250, pp. 33–50. Springer, Heidelberg (2012)

3. Navlakha, S., Rastogi, R., Shrivastava, N.: Graph summarization with bounded error. In: SIGMOD 2008: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 419–432. ACM, New York (2008)
4. Tian, Y., Hankins, R., Patel, J.: Efficient aggregation for graph summarization. In: SIGMOD 2008: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 567–580. ACM, New York (2008)
5. Lorrain, F., White, H.C.: Structural equivalence of individuals in social networks. Journal of Mathematical Sociology 1, 49–80 (1971)
6. Borgatti, S.P., Everett, M.G.: Regular blockmodels of multiway, multimode matrices. Social Networks 14, 91–120 (1992)
7. Zhang, N., Tian, Y., Patel, J.: Discovery-driven graph summarization. In: 2010 IEEE 26th International Conference on Data Engineering (ICDE), pp. 880–891. IEEE (2010)
8. 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, pp. 742–753. VLDB Endowment (August 2009)
9. Navlakha, S., Schatz, M., Kingsford, C.: Revealing biological modules via graph summarization. Presented at the RECOMB Systems Biology Satellite Conference; J. Comp. Bio. 16, 253–264 (2009)
10. Chen, C., Yan, X., Zhu, F., Han, J., Yu, P.: Graph OLAP: Towards online analytical processing on graphs. In: ICDM 2008: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, pp. 103–112. IEEE Computer Society, Washington, DC (2008)
11. Fjällström, P.O.: Algorithms for graph partitioning: A Survey. Linköping Electronic Atricles in Computer and Information Science, vol. 3 (1998)
12. Elsner, U.: Graph partitioning - a survey. Technical Report SFB393/97-27, Technische Universität Chemnitz (1997)
13. Faloutsos, C., McCurley, K.S., Tomkins, A.: Fast discovery of connection subgraphs. In: KDD 2004: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 118–127. ACM, New York (2004)
14. Hintsanen, P., Toivonen, H.: Finding reliable subgraphs from large probabilistic graphs. Data Mining and Knowledge Discovery 17, 3–23 (2008)
15. Toussaint, G.T.: The relative neighbourhood graph of a finite planar set. Pattern Recognition 12(4), 261–268 (1980)
16. Hauguel, S., Zhai, C.X., Han, J.: Parallel PathFinder algorithms for mining structures from graphs. In: 2009 Ninth IEEE International Conference on Data Mining, pp. 812–817. IEEE (2009)
17. Toivonen, H., Mahler, S., Zhou, F.: A Framework for Path-Oriented Network Simplification. In: Cohen, P.R., Adams, N.M., Berthold, M.R. (eds.) IDA 2010. LNCS, vol. 6065, pp. 220–231. Springer, Heidelberg (2010)
18. Adler, M., Mitzenmacher, M.: Towards compressing web graphs. In: Data Compression Conference, pp. 203–212 (2001)
19. Boldi, P., Vigna, S.: The webgraph framework I: compression techniques. In: WWW 2004: Proceedings of the 13th International Conference on World Wide Web, pp. 595–602. ACM, New York (2004)
20. Zhou, F., Mahler, S., Toivonen, H.: Review of BisoNet Abstraction Techniques. In: Berthold, M.R. (ed.) Bisociative Knowledge Discovery. LNCS (LNAI), pp. 166–178. Springer, Heidelberg (2012)