Simplification of Networks by Edge Pruning*

Fang Zhou, Sébastien Mahler, and Hannu Toivonen

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

Abstract. We propose a novel problem to simplify weighted graphs by pruning least important edges from them. Simplified graphs can be used to improve visualization of a network, to extract its main structure, or as a pre-processing step for other data mining algorithms.

We define a graph connectivity function based on the best paths between all pairs of nodes. Given the number of edges to be pruned, the problem is then to select a subset of edges that best maintains the overall graph connectivity. Our model is applicable to a wide range of settings, including probabilistic graphs, flow graphs and distance graphs, since the path quality function that is used to find best paths can be defined by the user. We analyze the problem, and give lower bounds for the effect of individual edge removal in the case where the path quality function has a natural recursive property. We then propose a range of algorithms and report on experimental results on real networks derived from public biological databases.

The results show that a large fraction of edges can be removed quite fast and with minimal effect on the overall graph connectivity. A rough semantic analysis of the removed edges indicates that few important edges were removed, and that the proposed approach could be a valuable tool in aiding users to view or explore weighted graphs.

1 Introduction

Graphs are frequently used to represent information. Some examples are social networks, biological networks, the World Wide Web, and so called BisoNets, used for creative information exploration [2]. Nodes usually represent objects, and edges may have weights to indicate the strength of the associations between objects. Graphs with a few dozens of nodes and edges may already be difficult to visualize and understand. Therefore, techniques to simplify graphs are needed. An overview of such techniques is provided in reference [3].

In this chapter, we propose a generic framework and methods for simplification of weighted graphs by pruning edges while keeping the graph maximally connected. In addition to visualization of graphs, such techniques could have applications in various network design or optimization tasks, e.g., in data communications or traffic.

* This chapter is a modified version of article "Network Simplification with Minimal Loss of Connectivity" in the 10th IEEE International Conference on Data Mining (ICDM), 2010 [1].

M.R. Berthold (Ed.): Bisociative Knowledge Discovery, LNAI 7250, pp. 179–198, 2012.

The framework is built on two assumptions: the connectivity between nodes is measured using the best path between them, and the connectivity of the whole graph is measured by the average connectivity over all pairs of nodes. We significantly extend and generalize our previous work [4]. The previous work prunes edges while keeping the full original connectivity of the graph, whereas here we propose to relax this constraint and allow removing edges which result in loss of connectivity. The intention is that the user can flexibly choose a suitable trade-off between simplicity and connectivity of the resulting network. The problem then is to simplify the network structure while minimizing the loss of connectivity.

We analyze the problem in this chapter, and propose four methods for the task. The methods can be applied to various types of weighted graphs, where the weights can represent, e.g., distances or probabilities. Depending on the application, different definitions of the connectivity are possible, such as the shortest path or the maximum probability.

The remainder of this article is organized as follows. We first formalize the problem of lossy network simplification in Section 2, and then analyze the problem in Section 3. We present a range of algorithms to simplify a graph in Section 4, and present experimental results in Section 5. We briefly review related work in Section 6, and finally draw some conclusions in Section 7.

2 Lossy Network Simplification

Our goal is to simplify a given weighted graph by removing some edges while still keeping a high level of connectivity. In this section we define notations and concepts, and also give some example instances of the framework.

2.1 Definitions

Let $G = (V, E)$ be a weighted graph. We assume in the rest of the chapter that G is undirected. An *edge* $e \in E$ is a pair $e = \{u, v\}$ of nodes $u, v \in V$. Each edge has a *weight* $w(e) \in \mathbb{R}$. 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, or equivalently, to say that u_1 and u_k are the endvertices of P. A path P can be regarded as the concatenation of several sub-paths, i.e., $P = P_1 \cup \ldots \cup P_n$, where each P_i is a path.

We parameterize our problem and methods with a *path quality function* $q(P) \rightarrow \mathbb{R}^+$. The form of the path quality function depends on the type of graph and the application at hand. For example, in a probabilistic or random graph, it can be the probability that a path exists. Without loss of generality, we assume that the value of any path quality function is positive, and that a larger value of q indicates better quality.

Given two nodes u and v in a weighted graph, they might be linked by a direct edge or a path, or none in a disconnected graph. A simple way to quantify how

strongly they are connected is to examine the quality of the best path between them [4]. Thus, the *connectivity between two nodes* u and v in the set E of edges is defined as

$$C(u, v; E) = \begin{cases} \max_{P \subset E : u \overset{P}{\rightsquigarrow} v} q(P) & \text{if such } P \text{ exists} \\ -\infty & \text{otherwise.} \end{cases} \tag{1}$$

A natural measure for the *connectivity of a graph* is then the average connectivity over all pairs of nodes,

$$C(V, E) = \frac{2}{|V|(|V| - 1)} \sum_{u,v \in V, u \neq v} C(u, v; E), \tag{2}$$

where $|V|$ is the number of nodes in the graph. Without loss of generality, in the rest of the chapter we assume the graph is connected, so $C(V, E) > 0$. (If the graph is not connected, we simplify each connected component separately, so the assumption holds again.)

Suppose a set of edges $E_R \subset E$ is removed from the graph. The connectivity of the resulting graph is $C(V, E \setminus E_R)$, and the *ratio of connectivity kept* after removing E_R is

$$rk(V, E, E_R) = \frac{C(V, E \setminus E_R)}{C(V, E)}. \tag{3}$$

Clearly, connectivity can not increase when removing edges. $rk = 1$ means the removal of edges does not affect the graph's connectivity. $0 < rk < 1$ implies that the removal of edges causes some loss of connectivity, while $rk = -\infty$ implies the graph has been cut into two or more components.

Our goal is to remove a fixed number of edges while minimizing the loss of connectivity. From the definitions in Equations (1)–(3) it follows that cutting the input graph drops the ratio to $-\infty$. In this chapter, we thus want to keep the simplified graph connected (and leave simplification methods that may cut the graph for future work). Under the constraint of not cutting the input graph, possible numbers of edges remaining in the simplified graph range from $|V| - 1$ to $|E|$. This follows from the observation that a maximally pruned graph is a spanning tree, which has $|V| - 1$ edges. Thus numbers of removable edges range from 0 to $|E| - (|V| - 1)$.

In order to allow users to specify different simplification scales, we introduce a parameter γ, with values in the range from 0 to 1, to indicate the strength of pruning. Value 0 indicates no pruning, while value 1 implies that the result should be a spanning tree. Thus, the number of edges to be removed by an algorithm is $|E_R| = \lceil \gamma(|E| - (|V| - 1)) \rceil$. Based on notations and concepts defined above, we can now present the problem formally.

Given a weighted graph $G = (V, E)$, a path quality function q, and a parameter γ, the *lossy network simplification* task is to produce a simplified graph $H = (V, F)$, where $F \subset E$ and $|E \setminus F| = \lceil \gamma(|E| - (|V| - 1)) \rceil$, such that $rk(V, E, E \setminus F)$ is maximized. In other words, the task is to prune the specified amount of edges while keeping a maximal ratio of connectivity.

2.2 Example Instances of the Framework

Consider a random (or uncertain) graph where edge weight $w(e)$ gives the probability that edge e exists. A natural quality of a path P is then its probability, i.e., the probability that all of its edges co-exist: $q(P) = \Pi_{\{u,v\}\in P} w(\{u,v\})$. Intuitively, the best path is the one which has the highest probability.

If edge weights represent lengths of edges, then the shortest path is often considered as the best path between two given nodes. Since in this case smaller values (smaller distances) indicate higher quality of paths, one can either reverse the definitions where necessary, or simply define the path quality as the inverse of the length, i.e., $q(P) = 1/\text{length}(P)$.

A flow graph is a directed graph where each edge has a capacity $w(e)$ to transport a flow. The capacity $c(P)$ of a path is limited by the weakest edge along that path: $c(P) = \min_{\{u,v\}\in P} w(\{u,v\}) = q(P)$. The best path is one that has the maximal flow capacity. If the flow graph is undirected, the graph can be simplified without any loss of quality to a spanning tree that maximizes the smallest edge weight in the tree.

3 Analysis of the Problem

In this section, we investigate some properties of the problem of lossy network simplification. We first note that the ratio of connectivity kept $rk(V,E,E_R)$ is multiplicative with respect to successive removals of sets of edges. Based on this we then derive two increasingly fast and approximate ways of bounding $rk(V,E,E_R)$. These bounds will be used by algorithms we give in Section 4.

3.1 Multiplicativity of Ratio of Connectivity Kept

Let E_R be any set of edges to be removed. Consider an arbitrary partition of E_R into two sets E_R^1 and E_R^2, such that $E_R = E_R^1 \cup E_R^2$ and $E_R^1 \cap E_R^2 = \emptyset$. Using Equation (3), we can rewrite the ratio of connectivity kept by E_R as

$$\begin{aligned} &rk(V,E,E_R^1 \cup E_R^2) \\ &\quad = \tfrac{C(V,E\setminus(E_R^1\cup E_R^2))}{C(V,E)} \\ &\quad = \tfrac{C(V,E\setminus E_R^1)}{C(V,E)} \cdot \tfrac{C(V,E\setminus E_R^1\setminus E_R^2)}{C(V,E\setminus E_R^1)} \\ &\quad = rk(V,E,E_R^1) \cdot rk(V,E\setminus E_R^1, E_R^2). \end{aligned}$$

In other words, the ratio of connectivity kept $rk(\cdot)$ is multiplicative with respect to successive removals of sets of edges.

An immediate consequence is that the ratio of connectivity kept after removing set E_R of edges can also be represented as the product of ratios of connectivity kept for each edge, in any permutation:

$$rk(V,E,E_R) = \Pi_{i=1}^{|E_R|} rk(V, E\setminus E_{i-1}, e_i),$$

where e_i the ith edge in the chosen permutation and $E_i = \{e_1, \ldots, e_i\}$ is the set of i first edges of E_R.

Note that the ratio of connectivity kept is *not* multiplicative for the ratios $rk(V, E, \{e_i\})$ of connectivity kept with respect to the original set E of edges. It is therefore not straightforward to select an edge set whose removal keeps the maximal $rk(V, E, E_R)$ value among all possible results.

The multiplicativity directly suggests, however, to greedily select the edge maximizing $rk(V, E \setminus E_{i-1}, e_i)$ at each step. The multiplicativity property tells that the exact ratio of connectivity kept will be known throughout the process, even if it is not guaranteed to be optimal. We will use this approach in the brute force algorithm that we give in Section 4. Two other algorithms will use the greedy search too, but in a more refined form that uses results from the next subsections.

3.2 A Bound on the Ratio of Connectivity Kept

Recall that the connectivity of a graph is the average connectivity among all pairs of nodes. In principle, the removal of an edge may cause the connectivity between any arbitrary pair of nodes to decrease. We now derive a lower bound for the connectivity kept, based on the effect of edge removal only on the endpoints of the edge itself.

Many path quality functions are recursive in the sense that sub-paths of a best path are also best paths between their own endpoints. (This is similar to the property known as *optimal substructure* in dynamic programming.) Additionally, a natural property for many quality functions q is that the effect of a local change is at most as big for the whole path P as it is for the modified segment $R \subset P$.

Formally, let $P = \arg\max_{P \subset E : u \overset{P}{\rightsquigarrow} v} q(P)$ be a best path (between any pair of nodes u and v), let $m \in P$ be a node on the path, let $R \subset P$ be a subpath (segment) of P and S a path (not in P) with the same endvertices as R. Function q is a *local recursive path quality function* if

$$q(P) = q\Big(\arg\max_{P_1 \subset E : u \overset{P_1}{\rightsquigarrow} m} q(P_1) \cup \arg\max_{P_2 \subset E : m \overset{P_2}{\rightsquigarrow} v} q(P_2) \Big)$$

and

$$\frac{q(P \setminus R \cup S)}{q(P)} \geq \frac{q(S)}{q(R)}.$$

Examples of local recursive quality functions include the (inverse of the) length of a path (when edge weights are distances), the probability of a path (when edge weights are probabilities), and minimum edge weight on a path (when edge weights are flow capacities). A negative example is average edge weight.

The local recursive property allows to infer that over all pairs of nodes, the biggest effect of removing a particular edge will be seen on the connectivity of the edge's own endvertices. In other words, the ratio of connectivity kept for any pair of nodes is at least as high as the ratio kept for the edge's endvertices.

To formalize this bound, we denote by $\kappa(E, e)$ the ratio of connectivity kept between the endvertices of an edge $e = \{u, v\}$ after removing it from the set E of edges:

$$\kappa(E,e) = \begin{cases} -\infty & \text{if } C(u,v;E\setminus\{e\}) = -\infty; \\ \frac{C(u,v;E\setminus\{e\})}{q(\{e\})} & \text{if } C(u,v;E\setminus\{e\}) < q(\{e\}); \\ 1 & \text{if } C(u,v;E\setminus\{e\}) \geq q(\{e\}). \end{cases} \quad (4)$$

The first two cases directly reflect the definition of ratio of connectivity kept (Equation 3) when edge e is the only path (case one) or the best path (case two) between its endpoint. The third case applies when $\{e\}$ is not the best path between between its endpoints. Then, its absence will not cause any loss of connectivity between u and v, and $\kappa(E, e) = 1$.

Theorem 1. *Let $G = (V, E)$ be a graph and $e \in E$ an edge, and let q be a local recursive path quality function. The ratio of connectivity kept if e is removed is lower bounded by $rk(V, E, e) \geq \kappa(E, e)$.*

Sketch of a proof. The proof is based on showing that the bound holds for the ratio of connectivity kept for any pair of nodes. (1) Case one: $\kappa(E, e) = -\infty$ clearly is a lower bound for any ratio of connectivity kept. (2) Case two: Consider any pair of nodes u and v. In the worst case the best path between them contains e and, further, the best alternative path between u and v is the one obtained by replacing e by the best path between the endvertices of e. Since q is local recursive, even in this case at least fraction $\kappa(E, e)$ of connectivity is kept between u and v. (3) Case three: edge e has no effect on the connectivity of its own endvertices, nor on the connectivity of any other nodes.

Theorem 1 gives us a fast way to bound the effect of removing an edge and suggests a greedy method to the lossy network simplification problem by removing an edge with the largest κ. Obviously, only based on $\kappa(E, e) < 1$, we can not infer the exact effect of removing edge e, nor the relative difference between removing two alternative edges. However, computing κ is much faster than computing rk, since only the best path between the edge's endvertices needs to be examined, not all-pairs best paths.

3.3 A Further Bound on the Ratio of Connectivity Kept

Previously, we suggested two ways to compute or approximate the best alternative path for an edge [4]. The *global best path search* finds the best path with unlimited length and thus gives the exact $C(u, v; E \setminus \{e\})$ and κ values. However, searching the best path globally takes time. A faster alternative, called *triangle search*, is to find the best path of length two, denoted by $S_2(e)$. That is, let $S_2(e) = \{\{u, w\}\{w, v\}\} \subset E$, $e \notin S_2(e)$, be a path between the endvertices u, v such that $q(S_2(e))$ is maximized. Obviously, path $S_2(e)$ may not be the best path between the edge's endvertices, and therefore $q(S_2(e))$ is a lower bound for the quality of the best path between the endvertices of e.

To sum up the results from this section, we have two increasingly loose lower bounds for the ratio of connectivity kept for local recursive functions. The first one is based on only looking at the best alternative path for an edge. The second one is a further lower bound for the quality of this alternative path. Denoting by $S_2(e)$ the best path of length two as defined above, we have

$$rk(V, E, e) \geq \kappa(E, e) \geq \min(\frac{q(S_2(e))}{q(\{e\})}, 1).$$

In the next section, we will give algorithms that use these lower bounds to complete the simplification task with different trade-offs between connectivity kept and time complexity.

4 Algorithms

We next present four algorithms to simplify a given graph by pruning a fixed number of edges while aiming to keep a high connectivity. All algorithms take as input a weighted graph G, a path function q and a ratio γ. They prune $n = \gamma(|E| - (|V| - 1))$ edges. The first algorithm is a naive approach, simply pruning a fraction of the weakest edges by sorting edges according to the edge weight. The second one is a computationally demanding brute-force approach, which greedily removes an edge with the highest rk value in each iteration. The third and fourth algorithms are compromises between these extremes, aimed at a better trade-off between quality and efficiency. The third one iteratively prunes the edge which has the largest κ value through global search. The fourth algorithm prunes edges with the combination of triangle search and global search.

4.1 Naive Approach

Among the four algorithms that we present, the simplest approach is the naive approach (NA), outlined in Algorithm 1. It first sorts edges by their weights in an ascending order (Line 1). Then, it iteratively checks the edge from the top of the sorted list (Line 7), and prunes the one whose removal will not lead to disconnected components (Line 8). The algorithm stops when the number of edges removed reaches n, derived from G and γ.

The computational cost of sorting edges is $O\big(|E| \log |E|\big)$ (Line 1). On Line 7, we use Dijkstra's algorithm with a complexity of $O\big((|E| + |V|) \log |V|\big)$ to check whether there exists a path between the edge's endvertices. So, the total computational complexity of the naive approach is $O\big(|E| \log |E| + n(|E| + |V|) \log |V|\big)$.

4.2 Brute Force Approach

The brute force approach (BF), outlined in Algorithm 2, prunes edges in a greedy fashion. In each iteration, it picks the edge whose removal best keeps the connectivities, i.e., has the largest rk value. It first calculates the $rk(V, F, e)$ value

Algorithm 1. NA algorithm

Input: A weighted graph $G = (V, E)$, q and γ
Output: Subgraph $H \subset G$

```
1: Sort edges E by weights in an ascending order.
2: F ← E
3: n ← γ(|E| − (|V| − 1))
4: { Iteratively prune the weakest edge which does not cut the graph }
5: i ← 1, j ← 1 { j is an index to the sorted list of edges }
6: while i ≤ n do
7:    if C(u, v; F \ {e_j}) is not −∞ then
8:       F ← F \ {e_j}
9:       i ← i + 1
10:   j ← j + 1
11: Return H = (V, F)
```

for every edge e (Line 10), and then stores the information of the edge whose $rk(V, F, e)$ value is the highest at the moment (Line 11), and finally prunes the one which has the highest rk value among all existing edges (Line 16). As an optimization, set M is used to store edges that are known to cut the remaining graph (Lines 9 and 15), and the algorithm only computes $rk(V, F, e)$ for the edges which are not in M (Line 8).

When computing $rk(V, F, e)$ for an edge (Line 10), all-pairs best paths need to be computed with a cost of $O\big(|V|(|E| + |V|)\log|V|\big)$. (This dominates the connectivity check on Line 9.) Inside the loop, $rk(V, F, e)$ is computed for all edges in each of n iterations, so the total time complexity is $O\big(n|E||V|(|E| + |V|)\log|V|\big)$.

Algorithm 2. BF algorithm

Input: A weighted graph $G = (V, E)$, q and γ
Output: Subgraph $H \subset G$

```
1: F ← E
2: n ← γ(|E| − (|V| − 1))
3: { Iteratively prune the edge with the highest rk value. }
4: M ← ∅ { edges whose removal is known to cut the graph. }
5: for r = 1 to n do
6:    rk_largest ← −∞
7:    e_largest ← null
8:    for e = {u, v} in F and e ∉ M do
9:       if graph (V, F \ {e}) is connected then
10:         compute rk(V, F, e) = C(V,F\{e}) / C(V,F)
11:         if rk(V, F, e) > rk_largest then
12:            rk_largest ← rk(V, F, e)
13:            e_largest ← e
14:      else
15:         M ← M + e
16:   F ← F \ {e_largest}
17: Return H = (V, F)
```

4.3 Path Simplification

The outline of the path simplification approach (PS) is in Algorithm 3. The main difference to the brute force approach is that PS calculates κ instead of $rk(V, F, e)$ for each edge.

The method finds, for each edge, the best possible alternative path S globally (Line 9). It then prunes in each loop the edge with the largest lower bound κ of connectivity kept. As an efficient shortcut, as soon as we find an edge whose κ is equal to 1, we remove it immediately. Again, list M is used to store information of those edges whose removal cuts the graph.

Algorithm 3. PS algorithm

Input: A weighted graph $G = (V, E)$, q and γ
Output: Subgraph $H \subset G$

```
 1: F ← E
 2: n ← γ(|E| − (|V| − 1))
 3: {Iteratively prune the edge with the largest κ value. }
 4: M ← ∅
 5: for r = 1 to n do
 6:    κ_largest ← −∞
 7:    e_largest ← null
 8:    for e = {u, v} in F and e ∉ M do
 9:       Find path S such that q(S) = C(u, v; F \ {e})
10:       if q(S) ≥ q({e}) then
11:          κ ← 1
12:          F ← F \ {e}
13:          break
14:       else if 0 < q(S) < q({e}) then
15:          κ ← q(S) / q({e})
16:       else
17:          κ ← −∞
18:          M ← M + e
19:       if κ > κ_largest then
20:          κ_largest ← κ
21:          e_largest ← e
22:    F ← F \ {e_largest}
23: Return H = (V, F)
```

The complexity of the innermost loop is dominated by finding the best path between the edge's endvertices (Line 9), which has time complexity $O\big((|E| + |V|) \log |V|\big)$. This is done n times for $O(|E|)$ edges, so the total time complexity is $O\big(n|E|(|E| + |V|) \log |V|\big)$. While still quadratic in the number of edges, this is a significant improvement over the brute force method.

4.4 Combinational Approach

The fourth and final algorithm we propose is the combinational approach (CB), outlined in Algorithm 4. The difference to the path simplification (PS) method

Algorithm 4. CB algorithm

Input: A weighted graph $G = (V, E)$, q and γ
Output: Subgraph $H \subset G$

1: $F \leftarrow E$
2: $n \leftarrow \gamma(|E| - (|V| - 1))$
3: { Iteratively prune the edge with the largest κ using triangle search }
4: $r \leftarrow 1$
5: $find \leftarrow$ **true**
6: **while** $r \leq n$ and $find =$ **true do**
7: $\quad \kappa\_largest \leftarrow -\infty$
8: $\quad e\_largest \leftarrow null$
9: $\quad$ **for** $e = \{u, v\}$ in F **do**
10: $\quad\quad$ Find path $S_2 = \{\{u, w\}\{w, v\}\} \subset F \setminus \{e\}$ that maximizes $q(S_2)$
11: $\quad\quad$ **if** $q(S_2) \geq q(\{e\})$ **then**
12: $\quad\quad\quad \kappa \leftarrow 1$
13: $\quad\quad\quad F \leftarrow F \setminus \{e\}$
14: $\quad\quad\quad r \leftarrow r + 1$
15: $\quad\quad\quad$ **break**
16: $\quad\quad$ **else if** $0 < q(S_2) < q(\{e\})$ **then**
17: $\quad\quad\quad \kappa \leftarrow \frac{q(S_2)}{q(\{e\})}$
18: $\quad\quad$ **else**
19: $\quad\quad\quad \kappa \leftarrow -\infty$
20: $\quad\quad$ **if** $\kappa > \kappa\_largest$ **then**
21: $\quad\quad\quad \kappa\_largest \leftarrow \kappa$
22: $\quad\quad\quad e\_largest \leftarrow e$
23: $\quad$ **if** $\kappa\_largest > 0$ **then**
24: $\quad\quad F \leftarrow F \setminus \{e\_largest\}$
25: $\quad\quad r \leftarrow r + 1$
26: $\quad$ **else**
27: $\quad\quad find \leftarrow$ **false**
28: **if** $r < n$ **then**
29: $\quad$ apply the path simplification (PS) method in Algorithm 3 to prune $n - r$ edges
30: Return $H = (V, F)$

above is that the best path search is reduced to triangle search (Line 10). However, triangle search is not always able to identify a sufficient number of edges to be removed, depending on the number and quality of triangles in the graph. Therefore the combinational approach invokes the PS method to remove additional edges if needed (Line 29).

The computational complexity of triangle search for a single edge is $O(|V|)$ (Line 10). Thus, if we only apply triangle search, the total cost is $O(n|E||V|)$. However, if additional edges need to be removed, the worst case computational complexity equals the complexity of the path simplification method (PS).

5 Experiments

To assess the problem and methods proposed in this chapter, we carried out experiments on real graphs derived from public biological databases. With the

experiments, we want to evaluate the trade-off between the size of the result and the loss of connectivity, compare the performances of the proposed algorithms, study the scalability of the methods, and assess what the removed edges are like semantically in the biological graphs.

5.1 Experimental Setup

We have adopted the data and test settings from Toivonen et al. [4]. The data source is the Biomine database [5] which integrates information from twelve major biomedical databases. Nodes are biological entities such as genes, proteins, and biological processes. Edges correspond to known or predicted relations between entities. Each edge weight is between 0 and 1, and is interpreted as the probability that the relation exists. The path quality function is the probability of the path, i.e., the product of weights of the edges in the path. This function is local recursive.

For most of the tests, we use 30 different graphs extracted from Biomine. The number of nodes in each of them is around 500, and the number of the edges ranges from around 600 to 900. The graphs contain some parallel edges that can be trivially pruned. For more details, see reference [4]. For scalability tests, we use a series of graphs with up to 2000 nodes, extracted from the same Biomine database.

The algorithms are coded in Java. All tests were run on standard PCs with x86_64 architecture with Intel Core 2 Duo 3.16GHz, running Linux.

5.2 Results

Trade-Off between Size of the Result and Connectivity Kept. By construction, our methods work on a connected graph and keep it connected. As described in Section 2, maximally simplified graphs are then spanning trees, with $|V| - 1$ edges. The number of edges removed is algorithm independent: they all remove fraction γ of the $|E| - (|V| - 1)$ edges that can be removed. The distribution of the number of edges to be removed in our test graphs, relative to the total number of edges, are shown as a function of γ in Figure 1. These graphs are relatively sparse, and approximately at most 35% of edges can be removed without cutting the graph.

In this chapter, we extend a previous simplification task [4] from lossless to lossy simplification (with respect to the connectivity of the graph). In other words, in the previous proposal the ratio of connectivity kept must always stay at 1. We now look at how many more edges and with how little loss of connectivity our new methods can prune. We use the path simplification method as a representative here (and will shortly compare the proposed methods).

In Figure 2, we plot the ratio of connectivity kept by the four methods of Toivonen et al. [4] for two different graphs, randomly selected from our 30 graphs. Four different types of points are positioned horizontally according to n, the number of edges pruned by the previous methods. The x-axis shows the number of edges pruned in terms of γ, computed as $\gamma = n/(|E|-(|V|-1))$. Results of the

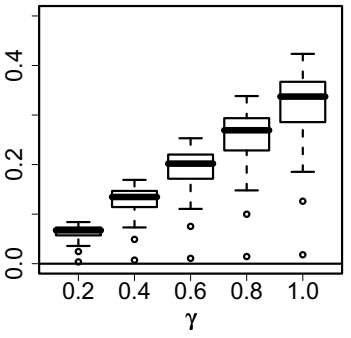


Fig. 1. Fraction of edges removed by different γ value. Each boxplot shows the distribution of results over 30 test graphs.

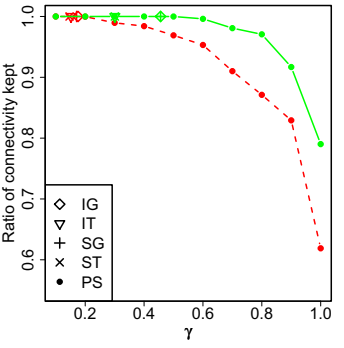


Fig. 2. Ratio of connectivity kept by the four methods of Toivonen et al. [4] and by the path simplification method for two graphs (green and red). IG=Iterative Global, IT=Iterative Triangle, SG=Static Global, ST=Static Triangle.

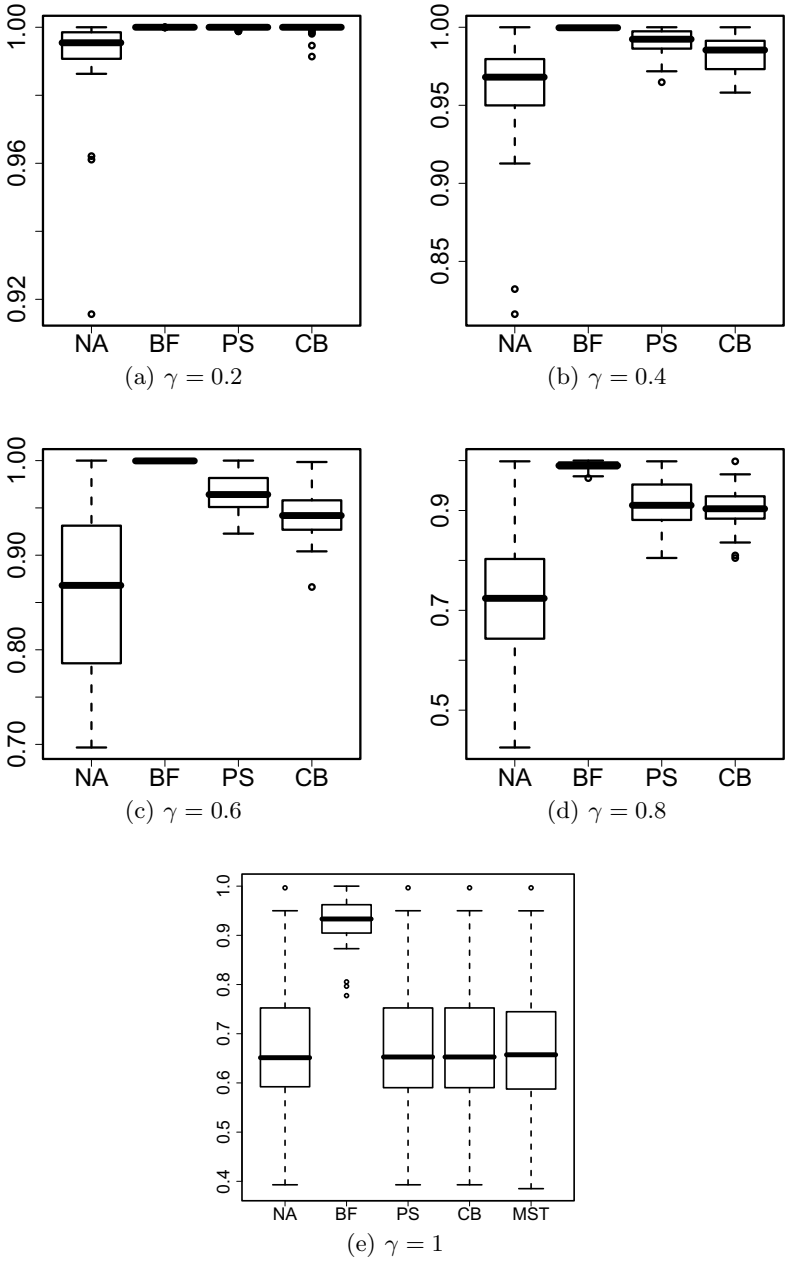
path simplification method proposed in this chapter are shown as lines. Among the four previous methods, the Iterative Global (IG) method prunes the maximal number of edges. Significantly more edges can be pruned, with larger values of γ, while keeping a very high ratio of connectivity. This indicates that the task of lossy network simplification is useful: significant pruning can be achieved with little loss of connectivity.

Comparison of Algorithms. Let us next compare the algorithms proposed in this chapter. Each of them prunes edges in a somewhat different way, resulting in different ratios of connectivity kept. These ratios with respect to different γ are shown in Figure 3. For $\gamma = 1$ (Figure 3(e)), where the result of all methods is a spanning tree, we also plot the results of a standard maximum spanning tree method [6] that maximizes the sum of edge weights.

Among all methods, the brute force approach expectedly always keeps the highest ratio of graph connectivity. When γ is between 0.2 and 0.6, the brute force method can actually keep the original connectivity, and even when $\gamma = 1$ it still keeps around 93% connectivity.

Overall, the four proposed methods perform largely as expected. The second best method is path simplification, followed by the combinational approach. They both keep high connectivities for a wide range of values for γ, still approximately 90% with $\gamma = 0.8$. The naive approach is clearly inferior, but it also produces useful results for smaller values of γ.

An interesting observation can be made from Figure 3(e) where $\gamma = 1$. The maximum spanning tree has similar ratios of connectivity kept with all methods except the brute force method, which can produce significantly better results.



(a) $\gamma = 0.2$ (b) $\gamma = 0.4$

(c) $\gamma = 0.6$ (d) $\gamma = 0.8$

(e) $\gamma = 1$

Fig. 3. Ratio of connectivity kept by each of the four algorithmic variants. Each boxplot shows the distribution of results over 30 test graphs. NA = Naive approach, BF = Brute Force, PS = Path Simplification, CB = Combinational approach, MST = Maximum Spanning Tree.

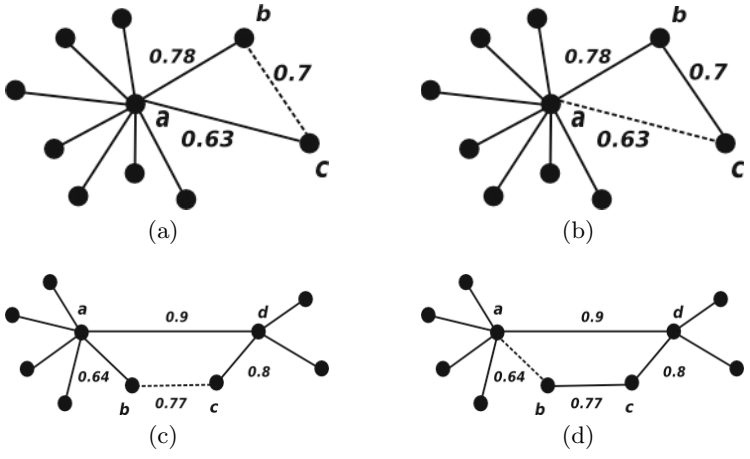


Fig. 4. Two examples where the brute force and path simplification methods remove different edges. In (a) and (c), dashed edges are removed by the brute force method. In (b) and (d), dashed edges are removed by the path simplification method.

This illustrates how the problem of keeping maximum connectivity in the limit ($\gamma = 1$) is different from finding a maximum spanning tree. (Recall that the lossy network simplification problem is parameterized by the path quality function q and can actually have quite different forms.)

Figure 4 shows two simple examples where the brute force method removes different edges than the path simplification method (or the maximum spanning tree method). The removed edges are visualized with dotted lines; Figures 4(a) and (c) are the results of the brute force method, and (b) and (d) are the results of the path simplification method. Consider the case in Figures 4(a) and (b). Since $\kappa(\{b, c\}) = \frac{0.63*0.78}{0.7} = 0.7$ and $\kappa(\{a, c\}) = \frac{0.78*0.7}{0.63} = 0.91$, edge $\{a, c\}$ is removed by the path simplification method. However, when considering the connectivity between node c and other nodes which are a's neighbors, removing $\{b, c\}$ keeps connectivity better than removing edge $\{a, c\}$.

We notice that the brute force method has a clear advantage from its more global viewpoint: it may select an edge whose weight is higher than the weight of the edge removed by the other methods that work more locally. We will next address the computational costs of the different variants.

Running Times. We next compare the running times of the four algorithms. Running times as functions of γ are shown in Figure 5. As we already know from the complexity analysis, the brute force method is quite time consuming. Even when γ is small, like 0.2, the brute force method still needs nearly one hundred minutes to complete. With the increase of γ, the time needed by the brute force increases from 100 to more than 400 minutes, while the other three methods only need a few seconds to complete. The second slowest method is the path simplification, which running time increases linearly with γ from 5 to 50 seconds. The naive approach always needs less than 1 second to complete.

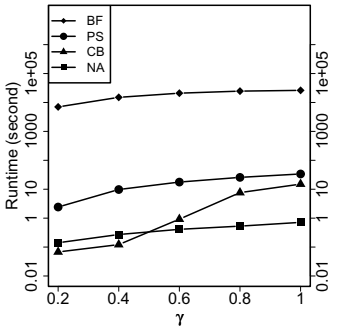


Fig. 5. Mean running times (in logarithmic scale) of 30 runs as functions of γ

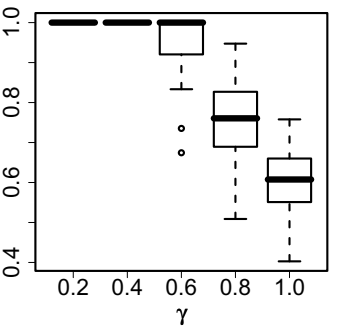


Fig. 6. Fraction of edges removed by triangle search in the combinational method

The combinational approach is the fastest one when γ is very small, but it comes close to the time the path simplification method needs when γ is larger. The reason for this behavior is that the combinational approach removes varying shares of edges using the computationally more intensive global search: Figure 6 shows that, with small values of γ, all or most edges are removed with the efficient triangle search. When γ increases, the fraction of edges removed by global search correspondingly increases.

In order to evaluate the scalability of the methods, we ran experiments with a series of graphs with up to 2000 nodes. The node degree is around 2.5. The running times as functions of graph size are shown in Figures 7 (with $\gamma = 0.4$) and 8 (with $\gamma = 0.8$).

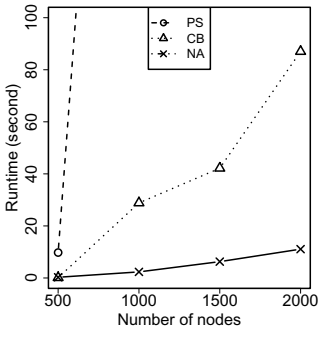


Fig. 7. Running times as functions of graph size (number of nodes) with $\gamma = 0.4$. The running time of the brute force method for a graph of 500 nodes is 15 000 seconds.

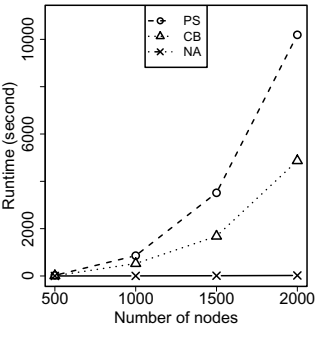


Fig. 8. Running times as functions of graph size (number of nodes) with $\gamma = 0.8$. The running time of the brute force method for a graph of 500 nodes is 25 000 seconds.

All methods have superlinear running times in the size of the graph, as is expected by the time complexity analysis. As such, these methods do not scale to very large graphs, at least not with large values of γ.

A Rough Semantic Analysis of Removed Edges. We next try to do a rough analysis of what kind of edges are pruned by the methods in the biological graphs of Biomine. The methods themselves only consider edge weights, but from Biomine we also have edges labels describing the relationships. We classify edges to important and irrelevant by the edge labels, as described below, and will then see how the methods of this chapter prune them.

In Biomine, certain edge types can be considered elementary: edges of an elementary type connect entities that strongly belong together in biology, such as a protein and the gene that codes for it. An expert would not like to prune these links. On the other hand, if they are both connected to a third node, such as a biological function, then one of these edges could be considered redundant. Since the connection between the protein and gene is so essential, any connections to either one could be automatically considered to hold also for the other one.

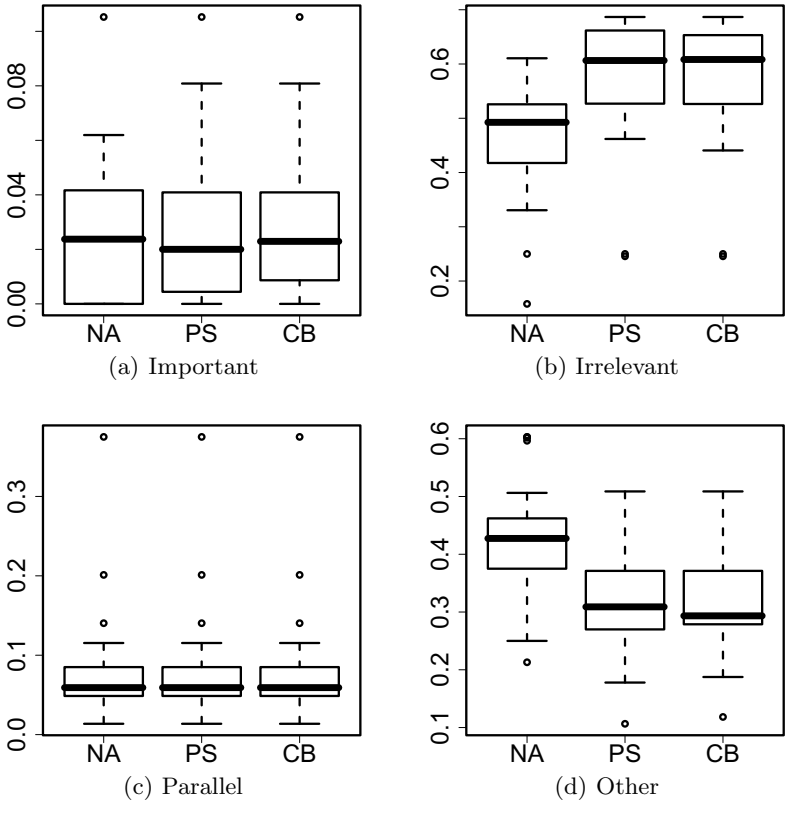


Fig. 9. Shares of different semantic categories among all removed edges with $\gamma = 0.8$

An explicit representation of such an edge would be considered "semantically irrelevant".

Following the previous setting [4], we considered the edge types *codes for*, *is homologous to*, *subsumes*, and *has synonym* as "important." Then, we computed the number of those edges that are "semantically irrelevant." Additionally, we marked the edges which have the same endvertices as "Parallel" edges. For the sake of completeness, we also counted the number of "other edges" that are neither important nor semantically irrelevant, nor parallel edges.

The semantic categories of the edges removed with $\gamma = 0.8$ are shown in Figure 9. Among edges removed by the naive approach, 3% are important, 45% are irrelevant, 8% are parallel and around 44% are other edges. The results of the path simplification and the combinational approach are quite similar: with edges removed by them there are around 2% important edges, 60% irrelevant edges, around 8% parallel edges and 30% other edges. (We do not analyze the semantic types of edges removed by the brute force method due to its time complexity.)

We notice that the path simplification and the combinational approach remove more irrelevant edges than the naive approach does. The reason is that these irrelevant edges may have a high weight, but they also have high κ value, in most cases, $\kappa = 1$.

The results indicate that the path simplification and the combinational approaches could considerably complement and extend expert-based or semantic methods, while not violating their principles.

6 Related Work

Network simplification has been addressed in several variants and under different names. Simplification of flow networks [7, 8] has focused on the detection of vertices and edges that have no impact on source-to-sink flow in the graph. Network scaling algorithms produce so-called Pathfinder networks [9–11] by pruning edges for which there is a better path of at most q edges, where q is a parameter. Relative Neighborhood Graphs [12] only connect relatively close pairs of nodes. They are usually constructed from a distance matrix, but can also be used to simplify a graph: indeed, relative neighborhood graphs use the triangle test only.

The approach most closely related to ours is path-oriented simplification [4], which removes edges that do not affect the quality of best paths between any pair of nodes. An extreme simplification that still keeps the graph connected, can be obtained by Minimum Spanning Tree (MST) [6, 13] algorithms. Our approach differs from all these methods in an important aspect: we measure and allow loss of network quality, and let the user choose a suitable trade-off.

There are numerous measures for edge importance. These can be used to rank and prune edges with varying results. Representative examples include edge betweenness [14], which is measured as the number of paths that run along the edge, and Birnbaum's component importance [15], defined as the probability that the edge is critical to maintain a connected graph.

The goal of extracting a subgraph graph is similar to the problem of reliable subgraph or connection subgraph extraction [16–18]). Their problem is, however, related to a set of (query) nodes, while our problem is independent of query nodes. They also prune least useful nodes, while we only prune edges.

We have reviewed related work more extensively in [3].

7 Conclusion

We have addressed the problem of network simplification given that the loss of connectivity should be minimized. We have introduced and formalized the task of selecting an edge set whose removal keeps the maximal ratio of the connectivity. Our framework is applicable to many different types of networks and path qualities. We have demonstrated the effect on random (or uncertain) graphs from a real-world application.

Based on our definition of ratio of connectivity kept, we have proposed a naive approach and a brute force method. Moreover, we have shown that the property of local recursive path quality functions allows to design a simpler solution: when considering the removal of one edge, the ratio of connectivity kept between the edge's endvertices can be used to bound the ratio for all pairs of nodes. Based on this observation, we have proposed two other efficient algorithms: the path simplification method and the combinational approach.

We have conducted experiments with 30 real biological networks to illustrate the behavior of the four methods. The results show that the naive approach is in most cases the fastest one, but it induces a large loss of connectivity. The brute force approach is very slow in selecting the best set of edges. The path simplification and the combinational approach were able to select a good set in few seconds for graphs with some hundreds of nodes. A rough semantic analysis of the simplification indicates that, in our experimental setting, both the path simplification and the combinational approach have removed very few important edges, and a relatively high number of irrelevant edges. We suggest those two approaches can well complement a semantic-based simplification.

Future work includes development of more scalable algorithms for the task of lossy network simplification. The problem and algorithms we proposed here are objective techniques: they do not take into account any user-specific emphasis on any region of the network. Future work may be to design query-based simplification techniques that would take user's interests into account when simplifying a network. It would also be interesting to combine different network abstraction techniques with network simplification, such as a graph compression method to aggregate nodes and edges [19].

Acknowledgment. We would like to thank Lauri Eronen for his help. 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. Zhou, F., Mahler, S., Toivonen, H.: Network Simplification with Minimal Loss of Connectivity. In: The 10th IEEE International Conference on Data Mining (ICDM), Sydney, Australia, pp. 659–668 (2010)
2. Dubitzky, W., Kötter, T., Schmidt, O., Berthold, M.R.: Towards Creative Information Exploration Based on Koestler's Concept of Bisociation. In: Berthold, M.R. (ed.) Bisociative Knowledge Discovery. LNCS (LNAI), vol. 7250, pp. 11–32. Springer, Heidelberg (2012)
3. Zhou, F., Mahler, S., Toivonen, H.: Review of BisoNet Abstraction Techniques. In: Berthold, M.R. (ed.) Bisociative Knowledge Discovery. LNCS (LNAI), vol. 7250, pp. 166–178. Springer, Heidelberg (2012)
4. 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)
5. Sevon, P., Eronen, L., Hintsanen, P., Kulovesi, K., Toivonen, H.: Link Discovery in Graphs Derived from Biological Databases. In: Leser, U., Naumann, F., Eckman, B. (eds.) DILS 2006. LNCS (LNBI), vol. 4075, pp. 35–49. Springer, Heidelberg (2006)
6. Kruskal Jr., J.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical society 7(1), 48–50 (1956)
7. Biedl, T.C., Brejová, B., Vinař, T.: Simplifying Flow Networks. In: Nielsen, M., Rovan, B. (eds.) MFCS 2000. LNCS, vol. 1893, pp. 192–201. Springer, Heidelberg (2000)
8. Misiołek, E., Chen, D.Z.: Efficient Algorithms for Simplifying Flow Networks. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 737–746. Springer, Heidelberg (2005)
9. Schvaneveldt, R., Durso, F., Dearholt, D.: Network structures in proximity data. In: The Psychology of Learning and Motivation: Advances in Research and Theory, vol. 24, pp. 249–284. Academic Press, New York (1989)
10. Quirin, A., Cordon, O., Santamaria, J., Vargas-Quesada, B., Moya-Anegon, F.: A new variant of the Pathfinder algorithm to generate large visual science maps in cubic time. Information Processing and Management 44, 1611–1623 (2008)
11. Hauguel, S., Zhai, C., Han, J.: Parallel PathFinder Algorithms for Mining Structures from Graphs. In: Proceedings of the 2009 Ninth IEEE International Conference on Data Mining, ICDM 2009, pp. 812–817. IEEE Computer Society, Washington, DC (2009)
12. Toussaint, G.T.: The Relative Neighbourhood Graph of a Finite Planar Set. Pattern Recognition 12(4), 261–268 (1980)
13. Osipov, V., Sanders, P., Singler, J.: The Filter-Kruskal Minimum Spanning Tree Algorithm. In: ALENEX, pp. 52–61. SIAM (2009)
14. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proc. Natl. Acad. Sci. U S A 99(12), 7821–7826 (2002)

15. Birnbaum, Z.W.: On the importance of different components in a multicomponent system. In: Multivariate Analysis - II, pp. 581–592 (1969)
16. Grötschel, M., Monma, C.L., Stoer, M.: Design of Survivable Networks. In: Handbooks in Operations Research and Management Science, vol. 7, pp. 617–672 (1995)
17. 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)
18. Hintsanen, P., Toivonen, H.: Finding reliable subgraphs from large probabilistic graphs. Data Min. Knowl. Discov. 17, 3–23 (2008)
19. 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)