

Finding reliable subgraphs from large probabilistic graphs

Petteri Hintsanen · Hannu Toivonen

Received: 19 June 2008 / Accepted: 20 June 2008 / Published online: 9 July 2008
Springer Science+Business Media, LLC 2008

Abstract Reliable subgraphs can be used, for example, to find and rank nontrivial links between given vertices, to concisely visualize large graphs, or to reduce the size of input for computationally demanding graph algorithms. We propose two new heuristics for solving the most reliable subgraph extraction problem on large, undirected probabilistic graphs. Such a problem is specified by a probabilistic graph G subject to random edge failures, a set of terminal vertices, and an integer K . The objective is to remove K edges from G such that the probability of connecting the terminals in the remaining subgraph is maximized. We provide some technical details and a rough analysis of the proposed algorithms. The practical performance of the methods is evaluated on real probabilistic graphs from the biological domain. The results indicate that the methods scale much better to large input graphs, both computationally and in terms of the quality of the result.

Keywords Link discovery and analysis · Graph mining · Graph visualization · Reliability

1 Introduction

Consider information search or discovery in a large graph of concepts and their weighted relationships. The user initiates a query by specifying some concepts (“search

Responsible editors: Walter Daelemans, Bart Goethals, and Katharina Morik.

P. Hintsanen (✉) · H. Toivonen
Helsinki Institute for Information Technology, Department of Computer Science,
University of Helsinki, P.O. Box 68, 00014 Helsinki, Finland
e-mail: petteri.hintsanen@cs.helsinki.fi

H. Toivonen
e-mail: hannu.toivonen@cs.helsinki.fi

terms”), and wishes to obtain other concepts and relationships that are related to the search concepts.

An application example is in analysis of biological information, conveniently represented as a graph of biological concepts (genes, proteins, tissues, phenotypes, etc.) and their relations, often uncertain since based on statistical or computational predictions. For instance, imagine a life scientist has arrived at a novel hypothesis that a particular gene might have an effect on the phenotype she is studying. A search engine we envision would now allow the scientist to search for information connecting the gene and the phenotype, in order to find possible evidence for the hypothesis, or to discover more detailed hypothesis about the mechanisms of the relationship.

The search problem can be formulated as a task of finding a small subgraph, of some limited size, of maximal relevance to both search terms (Faloutsos et al. 2004). For probabilistic graphs, a natural choice for defining the relevance of a subgraph is to measure the probability (*reliability*) that the search terms are connected in the subgraph, taking into account the uncertainty of edges (Hintsanen 2007; De Raedt et al. 2008). Roughly speaking, the resulting subgraph is more likely to contain strong edges, shorter paths, and independent paths; on the other hand, edges or paths that add little to the connectivity of the search terms are not likely to be included, reflecting the notion of relevance. Importantly, since the definition favors robust subgraphs and thus alternative paths, the results have an implicit bias towards graphs with more variety and less redundancy—also a desirable feature for information search and data mining.

Extracting a subgraph of maximal reliability is a fundamental task. In the search problem described above, it can be used to find and rank search results. Given that the search is non-trivial and that the results often contain unobvious, indirect relationships, it clearly is also a data mining problem. Subgraph extraction is useful for visualization of large graphs, and it can be used as a preprocessing step to reduce the amount of data for network analysis methods that do not scale well.

We use subgraph extraction as a component in Biomine¹, a search engine prototype for information discovery in biological databases. Biomine currently integrates and indexes information from eight major databases (Entrez gene, UniProt, Gene Ontology, OMIM, NCBI HomoloGene, InterPro, STRING, and KEGG) and consists of about 1 million vertices and 6 million edges.

In this paper, we address the problem of extracting a maximally reliable subgraph. We next present the problem more formally and then outline the contributions of this article.

2 The most reliable subgraph problem

We use standard probabilistic graphs. Let $G = (V, E)$ be a graph, where V is the set of vertices, $E \subseteq V \times V$ is the set of edges, and each edge has an associated probability p_e . The interpretation is that edge $e \in E$ exists with probability p_e , and conversely e does not exist, or is not true with probability $1 - p_e$. Edges are mutually independent.

¹ <http://biomine.cs.helsinki.fi>.

Given two vertices $s, t \in V$, the (*two-terminal network*) reliability $R(G, \{s, t\})$ of G is defined as the probability that there exists a path (a sequence of true edges) between s and t in G . A classical application of reliability is in communication networks, where each communication link (edge) may fail with some probability. The reliability then gives the probability that s and t can reach each other in the network. A more recent example is in analysis of uncertain information in biological databases (Sevon et al. 2006): biological concepts s and t are indirectly related if there exists a chain of indirect links between them.

The problem of finding the *most reliable subgraph* (MRSP) was introduced recently (Hintsanen 2007). Given a probabilistic graph G , a set of terminals vertices $U \subset V$, and a positive integer K , $1 \leq K \leq |E|$, the task is to find a subgraph $H \subset G$ connecting the terminals in U , such that H has $|E| - K$ edges and a maximal reliability with respect to the terminals.

We focus on the undirected, two-terminal variant of the MRSP, where G is undirected and $U = \{s, t\}$. While the problem above is well-defined for larger sets of terminals (find a subgraph that connects all terminals), some modifications may be more useful in practice. For any number of terminals, the problem may be unsolvable with the defined number of edges. For more than two terminals, a more useful definition of the problem would try to maximize those connections that can be made, rather than giving up completely. This is, however, outside the scope of this paper.

The MRSP is an inherently difficult problem: efficient solutions are available only for restricted classes of graphs, but cases on general graphs are most likely intractable (Hintsanen 2007).

Notation Let G be a graph. We denote the number of vertices in G by $|G|$, and the number of edges in G by $\|G\|$. The degree of a vertex v is denoted by $\deg(v)$, and the set of edges adjacent to v by $\text{adj}(v)$. For a given tree T and its vertex $v \in T$, we denote the parent of v by $\text{par}(v)$, and the set of children of v by $\text{ch}(v)$.

The union between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a new graph $H = G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. Other set operations for graphs are defined analogously. For notational convenience, we treat paths and edges as graphs in set operations, making it notationally easy, e.g., to add a path P to a graph G by writing simply $G \cup P$.

A path with endpoints u and v is said to be a u - v -path. Finally, we denote the (two-terminal) reliability of G by $R(G)$, and for the sake of simplicity, we do not usually mention terminals explicitly as they are clear from the context.

Contributions In this paper, we propose two efficient heuristics, BPI and SPA, for the two-terminal MRSP on general, undirected graphs. BPI is based on simple use of best paths to span a subgraph, whereas SPA involves a more elaborate construction of series-parallel graphs. Unlike previous methods that prune the original graph until it reaches the given size, the proposed methods are based on greedy strategies for incrementally constructing the subgraph. These strategies result in methods that are much less sensitive to the size of the original graph. We provide some technical details and a rough analysis of the algorithms. We also evaluate the proposed methods on

a number of real test cases from the biological domain. The results indicate that the proposed methods scale much better to large input graphs, both computationally and in terms of the quality of the result.

3 Related work

The above formulation for the most reliable subgraph problem was introduced by [Hintsanen \(2007\)](#). A similar problem was introduced by [De Raedt et al. \(2008\)](#) for the compression of ProbLog theories, essentially a first-order logical form of the same problem. Both problems can be seen as instances of the generic connection subgraph problem introduced by [Faloutsos et al. \(2004\)](#).

Greedy heuristics were proposed by [Hintsanen \(2007\)](#) and [De Raedt et al. \(2008\)](#) for pruning the original graph to the desired size. These methods estimate for each edge $e \in E$ the effect of its removal, $R_d(e) = R(G) - R(G - e)$, also known as Birnbaum reliability importance for disconnection ([Birnbaum 1969](#)). They then proceed by removing least relevant edges. They apply two extreme variants: one removes an edge at a time and re-estimates edge relevances after each removal ([De Raedt et al. 2008](#)), the other does not re-estimate relevances, but finds and removes edges with zero relevance ([Hintsanen 2007](#)).

Obviously, the first, iterative variant usually gives a better result, but at a higher computational cost. Given that accurate estimation of the reliability is not easy in itself, a repeated computation of the reliability of large graphs is prohibitive. Approaches that avoid this largely redundant computation have a better behavior. The approach adopted in ProbLog theory compression ([De Raedt et al. 2008](#)) uses binary decision diagrams (BDD) ([Bryant 1986](#)) to compute the reliability. While the cost of constructing the BDD for the original graph is high, this BDD can be used very efficiently to compute edge relevances during the process of pruning edges. Unfortunately, the time to construct a BDD can be unpredictable, and is prohibitive for very large graphs.

We next describe in some detail the method (“Monte Carlo Pruning”, Algorithm 1) implementing the other extreme approach ([Hintsanen 2007](#)). First, edge relevances are estimated by a Monte Carlo simulation. A large number of non-probabilistic graphs are generated by randomly deciding for each edge whether it is in the graph or not. Edge relevance $R_d(e)$ is then estimated for each edge as the fraction of graphs where removing e disconnects s and t . Next, the estimates are sorted into ascending order: now the least critical edges for the connection are at the top of the sorted list. In order to remove K edges from the graph, one could just remove the first K edges on the list. However, it is beneficial to do this iteratively by removing one edge at a time, and check at each iteration if there are any edges with a non-terminal endpoint of degree one. Such edges have zero relevance as they do not occur on any acyclic path connecting the terminals, and are removed first.

While [Faloutsos et al. \(2004\)](#) describe the connection subgraph problem in a generic form, they also propose to model the graph as an electrical resistor network, where the delivered current between terminals s and t is to be maximized. The goals are very similar to ours, but the assumptions made by the model are quite different.

Algorithm 1 Monte Carlo pruning (MCP)

Input: Probabilistic graph G , integer K
Output: Subgraph $H \subset G$ such that $\|H\| = \|G\| - K$
1: Generate randomly a large number of non-probabilistic graphs from G
2: **for all** $e \in E$ **do**
3: $C[e] \leftarrow$ MC-estimate of $R(G - e)$
4: Sort table C into descending order
5: $H \leftarrow G$
6: **while** $K > 0$ **do**
7: **while** there exists a vertex $v \in H, v \notin \{s, t\}$ such that $\deg(v) = 1$ **do**
8: Remove $\text{adj}(v)$ and v from H
9: $K \leftarrow K - 1$
10: **if** $K = 0$ **then**
11: **return** H
12: Remove the first element (edge) e from C
13: Remove e from H
14: $K \leftarrow K - 1$

4 Algorithms

Our objective is to solve the MRSP efficiently on large graphs. In this section, we describe two new methods, “Best Paths Incremental” (BPI) and “Series-Parallel Augmentation” (SPA), for the problem. Both methods are based on incremental approaches.

4.1 BPI: An incremental algorithm using best paths

The “Best Paths Incremental” algorithm (Algorithm 2) is based on a simple idea: find the most probable, or *best* paths between the terminal vertices s and t , and let them span a subgraph. The BPI algorithm adds best paths to the solution until it has at least $\|G\| - K$ edges. In order to have exactly $\|G\| - K$ edges, it then calls Monte Carlo Pruning to remove the possible excess edges.

The number of paths needed to span a subgraph of the desired size depends on the structure of G . We used initially $k_0 = 2 \cdot (\|G\| - K)$ best paths; this number was chosen experimentally, and it usually gave a sufficient number of paths. In those cases

Algorithm 2 Best paths incremental (BPI)

Input: Probabilistic graph G , integer K
Output: Subgraph $H \subset G$ such that $\|H\| = \|G\| - K$
1: $H \leftarrow \emptyset$
2: $k \leftarrow k_0$ {the initial number of best paths looked for}
3: **while** $\|H\| < \|G\| - K$ **do**
4: Let \mathcal{P} be the set of k most probable s - t -paths, arranged in descending order of probability
5: **while** $\|H\| < \|G\| - K$ **and** $\mathcal{P} \neq \emptyset$ **do**
6: Remove the first element (path) P from \mathcal{P}
7: Add P to H
8: $k \leftarrow k \cdot k_1$
9: Monte Carlo Pruning($H, \|H\| - \|G\| + K$)

where there are not enough paths to span a subgraph, the algorithm is restarted with a larger number $k \cdot k_1$ of paths. In our experiments, we set $k_1 = 2$.

Best paths can be found by any k shortest paths algorithm, by simply transforming edge probabilities to edge weights: $w_e = -\log(p_e)$. A multitude of polynomial time algorithms for finding k shortest paths have been proposed in the literature (Eppstein 1998; Roditty 2007; Hershberger et al. 2007). For our experiments, we have implemented a straightforward extension to Dijkstra's algorithm for finding k shortest simple paths between two vertices s and t . Instead of maintaining a single shortest s - v -path for each vertex $v \in V$, we keep a record of k shortest s - v -paths. The time complexity of our implementation is $O((k^2|V|^2 + k|V||E|) \log k|V|)$. This could be improved to $O(k|V|(|E| + |V| \log |V|))$ by Lawler's algorithm (Lawler 1972). With our implementation, the total time complexity of the BPI (excluding the last MCP call) is bounded by $O((|E| - K)(k^2|V|^2 + k|V||E|) \log k|V|)$, under the assumption that the chosen number of paths k is sufficient to span a subgraph of desired size.

Finally, note that the algorithm adds paths blindly in the sense that their effect on the reliability is not evaluated. A greedy version would produce better results, but the cost of repetitive evaluation of reliability could be prohibitive.

4.2 SPA: An incremental algorithm using series-parallel graphs

The next algorithm is based on direct optimization of the reliability of the result subgraph H in a greedy, iterative manner. The cost of evaluating reliability is greatly reduced by constructing *series-parallel graphs*, a restricted class of graphs for which the reliability can be evaluated very efficiently (Colbourn 1987). There are various definitions for directed and undirected series-parallel graphs in the literature. For our purposes, the following recursive definition using composition rules is useful (Valdes et al. 1982):

Definition 1 The class of (undirected, two-terminal) *series-parallel graphs* is defined by the following rules:

1. A probabilistic graph with two vertices s and t joined by a single edge is series-parallel with terminals s and t .
2. If G_1 and G_2 are series-parallel graphs with terminals $\{s_1, t_1\}$ and $\{s_2, t_2\}$, then so is the multigraph $H = G_1 \cup G_2$ constructed by one of the following operations:
 - (a) Identify t_1 with s_2 , and let $\{s_1, t_2\}$ be the terminals of H (series composition).
 - (b) Identify s_1 with s_2 and t_1 with t_2 , and let $\{s_1, t_1\}$ be the terminals of H (parallel composition).

In our algorithm, the subgraph H is first initialized to a single s - t -path; for example, the most probable path. Then H is trivially series-parallel. Next, we expand H by adding *augmenting paths*:

Definition 2 Let G be a graph, H a subgraph of G , and $\{u, v\}$ two vertices of H . Let $P \subset G$ be a u - v -path. Then P is an *augmenting path* of H if and only if $H \cap P = \{u, v\}$.

In other words, augmenting paths are paths between two vertices of H that visit vertices only from $G \setminus H$, except for their endpoints.

Algorithm 3 Series–parallel augmentation (SPA)

Input: Probabilistic graph G , integer K
Output: Subgraph $H \subset G$ such that $\|H\| = \|G\| - K$
1: Let H be the most probable s – t –path in G
2: **while** $\|H\| < \|G\| - K$ **do**
3: Let U be the set of connectible vertex pairs
4: **if** $U = \emptyset$ **then**
5: **return** H
6: $r_{\max} \leftarrow 0$
7: **for all** $(u, v) \in U$ **do**
8: Let P be the most probable valid augmenting path between u and v
9: $r \leftarrow R(H \cup P)$
10: **if** $r > r_{\max}$ **then**
11: $r_{\max} \leftarrow r$
12: $P^* \leftarrow P$
13: Add P^* to H
14: Monte Carlo Pruning($H, \|H\| - \|G\| + K$)

Since we restrict H to be series–parallel, we cannot use every possible augmenting path, but only those paths P for which $H \cup P$ is series–parallel. We call such paths *valid*, and their endpoints *connectible*. Let \mathcal{P} be a set of valid augmenting paths of H . As $H \cup P$ is series–parallel, we can efficiently decide which path gives the maximum increase in reliability, i.e., find $P^* = \arg \max_{P \in \mathcal{P}} R(H \cup P)$.

The ‘‘Series–Parallel Augmentation’’ algorithm (Algorithm 3) now simply greedily adds the best valid augmenting path P^* to H until the required number of edges has been reached, or there are no valid paths available. Finally, it calls Monte Carlo pruning to remove the possible excess edges.

A problem with series–parallel graphs is that they can be too restricted to produce good subgraphs. To alleviate this problem, we can produce several different smaller series–parallel graphs and output their union, which is not necessarily series–parallel. For example, consider a case where the first few iterations in Algorithm 3 quickly grow the reliability of the subgraph H_1 , but the rest of the iterations give only a marginal increase to the reliability of H_1 . In such case, it can be beneficial to stop the algorithm and start over using a different initial solution $P \not\subset H_1$ to produce another subgraph H_2 . Finally, we simply combine the solutions to get a final result $H = H_1 \cup H_2$.

Next, we modify Algorithm 3 to implement these ideas (Algorithm 4). First, building a series–parallel graph H is stopped as soon as $R(H \cup P^*) < C \cdot R(H)$ for a given constant $C > 1$, where H is the result subgraph and P^* is the optimal augmenting path. Second, a new initial solution $P \not\subset H$ is chosen and a new series–parallel subgraph H' is grown. The choice $P \not\subset H$ guarantees that the new subgraph is not included in the current partial solution, ensuring an increase in the size of the result, introducing variance, and in effect leading to a non-series–parallel end result. We always choose the most probable such path P from the set \mathcal{P} of the k most probable s – t –paths in G .

The number of initial solutions needed to construct a subgraph of the desired size depends on the structure of G and the chosen constant C . With larger values of C , more initial solutions are needed. We used initially $k_0 = \|G\| - K$ best paths, which

Algorithm 4 Series–parallel augmentation with multiple initial solutions

Input: Probabilistic graph G , integer K , real number $C > 1$.
Output: Subgraph $H \subset G$ such that $\|H\| = \|G\| - K$

- 1: $H \leftarrow \emptyset$
- 2: $k \leftarrow k_0$ {the initial number of best paths used as initial solutions}
- 3: Let \mathcal{P} be the set of k most probable s – t -paths
- 4: **while** $\|H\| < \|G\| - K$ **do**
- 5: Let $P \in \mathcal{P}$ be the most probable path such that $P \not\subset H$
- 6: **if** $P = \emptyset$ **then** {increase k and restart}
- 7: $k \leftarrow k \cdot k_1$
- 8: Let \mathcal{P} be the set of k most probable s – t -paths
- 9: $H \leftarrow \emptyset$
- 10: **go to** 4
- 11: $H' \leftarrow P$
- 12: **while** $\|H'\| < \|G\| - K$ **do**
- 13: Let U be the set of connectible vertex pairs in H'
- 14: **if** $U = \emptyset$ **then**
- 15: **go to** 26 {break from loop}
- 16: $r_{\max} \leftarrow 0$
- 17: **for all** $(u, v) \in U$ **do**
- 18: Let P be the most probable valid augmenting path between u and v with respect to H' and G
- 19: $r \leftarrow R(H' \cup P)$
- 20: **if** $r > r_{\max}$ **then**
- 21: $r_{\max} \leftarrow r$
- 22: $P^* \leftarrow P$
- 23: **if** $r_{\max} < C \cdot R(H')$ **then**
- 24: **go to** 26 {break from loop}
- 25: Add P^* to H'
- 26: Add H' to H
- 27: Monte Carlo Pruning($H, \|H\| - \|G\| + K$)

was chosen experimentally as in the BPI algorithm. Algorithm 4 is restarted with a larger number $k \cdot k_1$ of initial solution paths, if there are not enough paths to construct a subgraph with the desired size. In our experiments, we set $k_1 = 2$.

4.3 Remarks on the implementation and complexity of SPA

There are a few non-trivial implementation issues on Algorithm 3. Next, we will show how connectible vertex pairs can be found efficiently (line 3), how to find the most probable valid augmenting path (line 8), and how to calculate the reliability of a series–parallel graph (line 9). We also discuss the total time complexity of SPA algorithm.

4.3.1 Finding connectible vertex pairs

Following Definition 1, a series–parallel graph G can be conveniently represented with a *decomposition tree* (Valdes et al. 1982). A decomposition tree T of G is a binary tree, where each leaf vertex represents an edge of G . Inner vertices have always two children and represent one of the composition operations of Definition 1. If an inner vertex represents a series composition, we call it *S-vertex*, otherwise it is *P-vertex*.

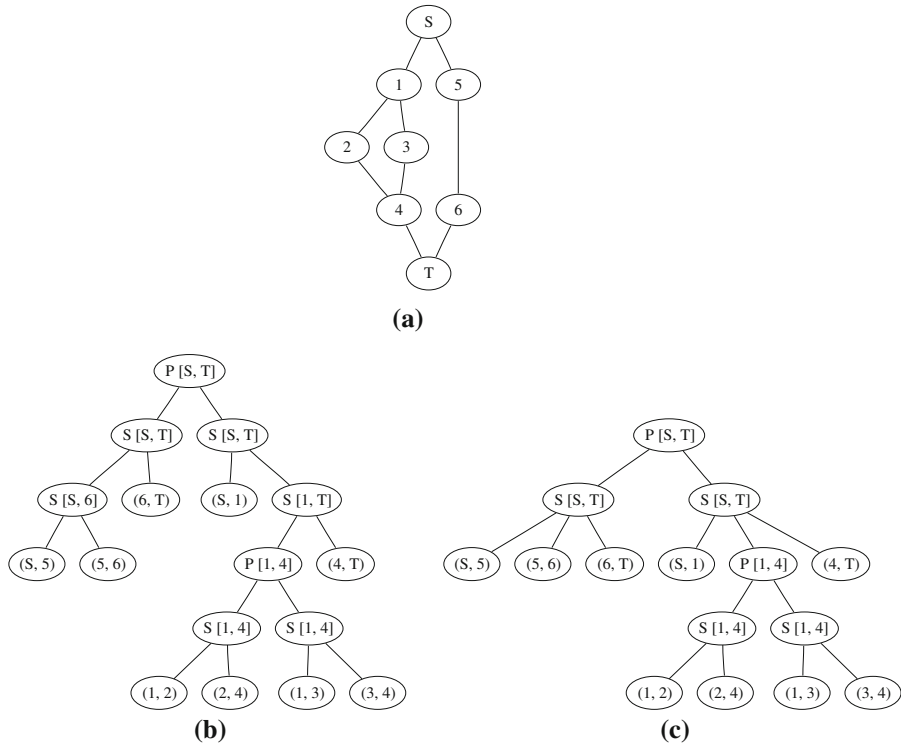


Fig. 1 A series–parallel graph (a), its decomposition tree (b), and its compressed decomposition tree (c). Leaf vertices represent single edges, while inner vertices represent series decompositions (S) or parallel decompositions (P). Terminals of subgraphs represented by inner vertices are enclosed in brackets

For each vertex $v \in T$, the subtree rooted at v represents a series–parallel subgraph $G_v \subset G$ with two terminals, which we denote by $\tau(v)$. If v is a leaf vertex, the subgraph is simply the edge stored at v , and its terminals are the endpoints of the edge. At each inner vertex u with children v and w , the associated composition operation joins the two subgraphs G_v and G_w , and the terminals of $G_u = G_v \cup G_w$ are determined by the composition operation from the terminals of G_v and G_w (see Fig. 1 for an example of a decomposition tree).

To quickly discover connectible vertex pairs, we use a *compressed* decomposition tree T_c , which differs from a regular decomposition tree in two ways:

1. An inner vertex $v \in T_c$ has two or more children. The composition operation of v is applied to all subgraphs G_{c_i} in succession, where c_i are the children of v , enumerated in order from left to right.
2. If a non-root vertex $u \in T_c$ is S-vertex (or P-vertex), then $\text{par}(u)$ is P-vertex (S-vertex).

We can easily convert a regular decomposition tree T to a compressed decomposition tree T_c by recursively combining $\{u, \text{par}(u)\}$ pairs of inner vertices having the same composition operation. Let $u \in T$ be an inner vertex, and suppose

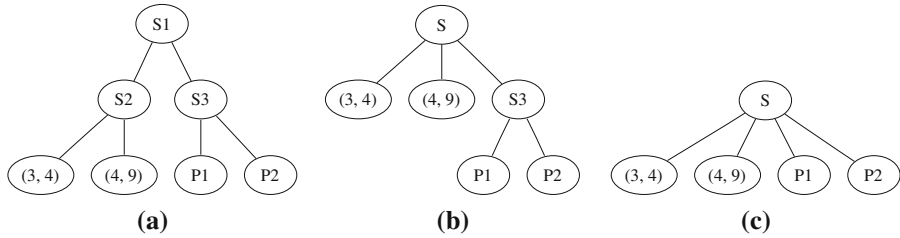


Fig. 2 Compression of a (partial) decomposition tree. Root S -vertex S_1 has two children of type S (a). We compress the tree by replacing S_2 with its child edges $(3, 4)$ and $(4, 9)$ in (b), and S_3 with its child compositions P_1 and P_2 in (c)

that a non-leaf child $v \in \text{ch}(u)$ has the same type (S or P) as u . We replace v with its children, and repeat this for all children of u having the same type. After repeating this procedure for all inner vertices $u \in T$, we have a compressed decomposition tree T_c (see Figs. 1, 2 for illustrative examples).

The relationship between the terminals in a compressed decomposition tree and connectible vertex pairs is characterized by the following lemma:

Lemma 1 *Let G be a probabilistic graph, T a compressed decomposition tree of G , and let $\{u, v\} \subset V$ be two connectible vertices. Then there exists a vertex $x \in T$ such that*

$$\{u, v\} = \tau(x), \tag{1}$$

or

$$\{u, v\} \subset \bigcup_{c \in \text{ch}(x)} \tau(c). \tag{2}$$

Proof If $(u, v) \in E$, (1) holds trivially. Thus, we assume $(u, v) \notin E$. Consequently, x is an inner vertex. Now assume the contrary of (2):

$$\forall x \in T: \{u, v\} \not\subset \bigcup_{c \in \text{ch}(x)} \tau(c). \tag{3}$$

Let y be the highest vertex in T such that $u \in \tau(y)$. Similarly, let z be the highest vertex in T such that $v \in \tau(z)$. Finally, let $x \in T$ be the most recent common ancestor of y and z . We separate two cases.

First suppose that $x \neq y$ and $x \neq z$. Since T is compressed and y and z are as high in T as possible, y and z are P -vertices. Furthermore, they lie in different subtrees rooted at two children of x : denote these children by c_1 and c_2 . If x is P -vertex, then connecting u and v connects G_{c_1} and G_{c_2} in such a way that G_x contains an embedded Wheatstone bridge. Thus, G_x is no longer series-parallel (Duffin 1965), which is a contradiction. On the other hand, if x is S -vertex, c_1 and c_2 are P -vertices connecting subgraphs formed by series compositions. If u and v are connected by an

augmenting path, it does not pass through any vertex in $\tau(c_1)$. In such case G_x is not series-parallel, which is again a contradiction.

In the second case we have $x = y$ or $x = z$. Assume $x = y$, the case $x = z$ is identical. Since T is compressed, z is P -vertex and $\text{par}(z)$ is S -vertex connecting two or more parallel compositions. Furthermore, $\tau(\text{par}(z)) = \{a, b\}$ does not contain v (z is as high as possible in T), so all s - v -paths go through a or b . By (3), $a \neq u$ and $b \neq u$; hence, if we connect u and v by an augmenting path, we bypass a or b . Then G_y is no longer series-parallel: a contradiction. \square

The significance of Lemma 1 is that the search for connectible vertex pairs in a graph reduces to a simple recursion in the corresponding compressed decomposition tree. This is summarized in Theorem 1 below.

Theorem 1 *Let G be a series-parallel graph, and let T be a compressed decomposition tree of G . Then the vertex pair collection*

$$\mathcal{C} = \left\{ \{u, v\} \subset \bigcup_{c \in \text{ch}(x)} \tau(c) \cup \tau(x) : x \in T \right\} \tag{4}$$

consists of all connectible vertex pairs of G .

Proof By Lemma 1, all connectible vertex pairs of G are contained in \mathcal{C} . On the other hand, \mathcal{C} contains only connectible vertex pairs. To see this, consider an arbitrary vertex pair $\{u, v\} \in \mathcal{C}$, and let $x \in T$ be the corresponding vertex in (4).

Suppose first that x is a leaf vertex representing a single edge $e \in E$. It is obvious that the endpoints of e can be connected with an augmenting path P : namely, P forms a new series composition which, in turn, forms a new parallel composition with e , and $G \cup P$ remains series-parallel. Hence, u and v are connectible.

Now assume that x is an internal vertex. Since T is compressed, there exist child vertices y and z such that $u \in \tau(y)$ and $v \in \tau(z)$. Let P be an augmenting path connecting u and v . Again, P forms a new series composition S . Assume first that x is P -vertex. Then $\tau(c)$ are identical for all $c \in \text{ch}(x)$, and S is simply a new series component to the existing parallel composition of x . Conversely, if x is S -vertex, we get a new parallel composition consisting of S together with a new series composition of y, c_1, \dots, c_k, z , where c_i are the children of x between y and z . In both cases, $G \cup P$ is series-parallel, and u and v are connectible. \square

In conclusion, we can implement line 3 of Algorithm 3 efficiently by maintaining a separate compressed decomposition tree T for the solution subgraph H . Then, all possible connectible vertex pairs can be found by recursively traversing T and forming the vertex pairs according to (4). This can be done in $O(\|H\|^2)$ time: although the size of T is $O(\|H\|)$, in the worst case almost every vertex pair is connectible (for example, after the initialization when H consists of a single s - t -path). Finally, after expanding H with the optimal augmenting path, we need to update the compressed decomposition tree, which can be done in constant time.

4.3.2 Finding the most probable augmenting path

After forming the collection of connectible vertex pairs, we need to find the most probable augmenting path for each connectible vertex pair. Let $\{u, v\} \subset H$ be such a pair. A straightforward way to proceed is to attach u and v into $G \setminus H$ using only edges from a set $F = \{(x, y) : (x, y) \in E, x = u \vee x = v, y \in G \setminus H\}$, that is, edges with one endpoint at u or v and another at $G \setminus H$. Clearly, this can be done in $O(|E|)$ time. The most probable augmenting path connecting u and v can be then sought on the resulting graph. With a standard implementation of Dijkstra’s algorithm using priority queues, the running time is $O(|E|) + O(\|G \setminus H\| + |G \setminus H|) \log |G \setminus H|$. After repeating the procedure for every connectible vertex pair, the total running time is $O(\|H\|^2(|E| + (\|G \setminus H\| + |G \setminus H|) \log |G \setminus H|))$.

A few optimizations can be readily done to speed up the implementation. First, for each $v \in H$, the set of neighbor vertices $U_v \subset G \setminus H$ of v can be separately maintained to facilitate rapid attachment of v to $G \setminus H$. To minimize relatively expensive shortest path computations, we can cache all connectible vertex pairs for which there was no augmenting path found. Such pairs can be skipped in later iterations, since successive graphs $G \setminus H$ are strictly decreasing. Finally, if we use a single-source shortest paths algorithm (such as Dijkstra’s algorithm), we can also cache the shortest paths from $u \in H$ to all $w \in G \setminus H$, and consult these caches when finding the most probable augmenting paths between u and $v, v \in H$, during the same iteration. This effectively halves the number of shortest path computations per iteration.

4.3.3 Calculating the reliability of a subgraph

The reliability of a series–parallel graph G is easy to calculate with the corresponding (regular or compressed) decomposition tree T . We perform a post-order traversal in T , and in each vertex $v \in T$, the reliability of the corresponding subgraph is calculated from the reliabilities of its children:

$$R(G_v) = \begin{cases} p_e & \text{if } v \text{ is leaf} \\ \prod_{c \in \text{ch}(v)} R(G_c) & \text{if } v \text{ is } S\text{-vertex} \\ 1 - \prod_{c \in \text{ch}(v)} (1 - R(G_c)) & \text{if } v \text{ is } P\text{-vertex} \end{cases} \tag{5}$$

Since the size of the decomposition tree is $O(\|H\|)$, we can implement line 9 of Algorithm 3 to work in $O(\|H \cup P\|)$ time.

4.3.4 Complexity of SPA

By substituting upper bounds $O(\|H\|) = O(|E| - K)$ and $O(\|H \cup P\|) = O(|E|)$ to the observations made in the previous sections, we can give a very rough upper bound $O((|E| - K)^3(E + (|E| + |V|) \log |V|))$ for the total time complexity of SPA. This bound is not tight, but it demonstrates the significant (inverse) effect of K on the running time. As SPA is an incremental algorithm, the running time decreases as the size of H decreases, i.e., as K increases. In typical applications, $\|H\| = |E| - K$ is small and thus also the cubic factor is small.

5 Experimental evaluation

We have conducted a series of experiments with the proposed methods (Best Paths Incremental, BPI; Series–Parallel Augmentation, SPA) as well as with the recently proposed method (Monte Carlo Pruning, MCP) (Hintsanen 2007). The goals of this experimental evaluation are the following: to assess the quality of the results (reliability of the extracted subgraphs), to test the scalability of the methods, and to test the effect of using multiple initial solutions for SPA.

5.1 Test setup

Test data The experiments have been carried out on different real biological graphs, illustrating different usage scenarios.

Our data source is the Biomine database (Sevon et al. 2006). It currently indexes information from eight major databases (Entrez gene, UniProt, Gene Ontology, OMIM, NCBI HomoloGene, InterPro, STRING, and KEGG) and consists of about 1 million vertices and 6 million edges. Biomine stores biological entities and their relationships as a probabilistic graph. Vertices of the graph represent biological entities such as genes, proteins and molecular processes, as well as more abstract concepts such as molecular loci and article abstracts. Edges represent annotated relationships between entities. Edges are obtained as cross-references from the original databases; their probabilities are determined by various factors taking account of the source database reliability, subjective relevance, and informativeness (Sevon et al. 2006).

For the experiments, we designed three different types of scenarios, and then extracted several graph datasets for each scenario (Table 1). We describe the scenarios first and then outline the graph extraction method.

- Scenario 1 (Dys): The user is interested in a known relationship between entities. The test case is built around a dyslexia-related gene vertex (Entrez gene id 6091)

Table 1 Description of the subgraphs used in the experiments

Graph id	Query vertices	Number of edges	Number of vertices
Dys1	Entrez gene 6091, OMIM 127700	141	64
Dys2	Entrez gene 6091, OMIM 127700	1,113	452
Dys3–Dys7	Entrez gene 6091, OMIM 127700	3,818–11,861	1,775–4,163
Conn1	PubMed 11960552, Entrez gene 2719	853	487
Conn2	GO 51605, Entrez gene 286971	2,727	989
Conn3	Entrez gene 181788, Entrez gene 169026	1,800	591
Conn4	UniProt P84751, STRING ENSP297185	4,504	908
Conn5	UniProt Q8C3X4, UniProt Q921W0	3,513	1,172
Gene1	Entrez gene 284467, Entrez gene 26231	51,672	16,811
Gene2	Entrez gene 54758, Entrez gene 389874	41,377	13,875
Gene3	Entrez gene 55222, Entrez gene 652968	10,082	3,815
Gene4	Entrez gene 728731, Entrez gene 79645	10,922	3,781

and a dyslexia phenotype vertex (OMIM id 127700), to be used as query vertices. We extracted graphs of different sizes for this pair of vertices, to test the methods on differently sized graphs.

- Scenario 2 (Conn): The user is interested in a putative relationship between entities. For each test case in this scenario, we selected two random vertices with a shortest connecting path of three edges.
- Scenario 3 (Gene): The user is interested in entities of no known connection. For each test case, we selected two random gene vertices, both having degree 10.

Given a query vertex pair $\{s, t\}$, the corresponding test graph was retrieved from Biomine with a best-first search from both vertices. The probability threshold was 0.01, i.e., all vertices and edges were retrieved from which the best path to either of the original vertices had at least probability 0.01. After retrieval, leaf vertices (non-query vertices with degree 1) and vertices occurring only on cyclic s – t -paths, along with their adjacent edges, were removed. Such vertices and edges have zero relevance for reliability.

For the scalability test sets (Dys1–7), however, we used different strategies. Dys1 and Dys2 were obtained with different search radiuses in order to obtain graphs of different sizes. Dys3–7 graphs, in turn, were obtained from a large graph of 13,828 edges by randomly removing roughly 2,000 edges at a time, so that every new (smaller) graph is a subgraph of the previous one.

Test queries In each test, the two query terminals are the ones specified above (Table 1). Our emphasis is on extracting small subgraphs, ideally suitable for interactive exploration. For such use, the subgraphs should have at most dozens of edges. In most of our tests, we vary the resulting subgraph size from 10 to 160 edges for a fixed graph, to study the effect of the subgraph size on reliability and running time. To study the effect of input graph size, we vary the graph and constantly extract a subgraph of 50 edges.

Parameters For both Best Paths Incremental and Series–Parallel Augmentation, the last step of the algorithms involves pruning (few) excess edges by the Monte Carlo Pruning algorithm. For this, we used 1 million iterations, which should be a safe choice without significant computational overhead for an extracted graph of less than 200 edges.

For SPA, we did not use alternative initial solutions except in the tests devoted to their effects.

MCP was run with 1 million iterations, unless otherwise stated.

For MCPre-est we used different re-estimation strategies depending on the case. The parameters are always specified with the results; the notation 10000/50 indicates that 10,000 MC iterations were performed every 50 edge removals to re-estimate the edge relevances.

Test environment The algorithms have been implemented in a mixture of Python and C. The input graph was a simple text file. For performance analysis, we have measured

the total elapsed (wall clock) time in seconds. All tests have been run on standard PCs running Linux, with little other activity during the tests.

5.2 Results

Scalability We first study the scalability of the methods, both in terms of the reliability of the produced subgraphs as well as the running times. For the experiments we used the Dys3–Dys7 graphs (see Table 1). We extracted a relevant subgraph of 50 edges from each of these subgraphs using BPI, SPA, MCP, and MCPre-est.

Both BPI and SPA produce similar reliabilities, approximately in the range 0.04–0.58 (Fig. 3a). Smaller input graphs result in smaller reliabilities in the extracted subgraphs, since the original graphs have smaller reliabilities themselves. MCPre-est produces poor results, with reliabilities in the range 0–0.05. MCP (without re-estimation) failed completely, and in these tests never returned a connected subgraph. Actually, in all our tests involving graphs with over 1,000 edges, MCP either failed to connect the terminals or produced clearly inferior subgraphs compared to BPI and SPA. This happened despite the fact that we used a very large number (1 million) of Monte Carlo iterations in order to have maximally good results for MCP.

The poor performance of MCP and to some extent of MCPre-est is due to the fact the true $R(G - e)$ values approach zero when the number of edges grows, and the accuracy of the corresponding estimates is not sufficient to separate edges. Effectively, MCP ends up removing edges almost by random. MCPre-est partially avoids this deficiency by re-estimating $R(G - e)$ values at some intervals.

The computational scalability of the proposed methods (BPI, SPA) is good (Fig. 3b). Running times are in the range 43–134 s. The Monte Carlo based alternatives, MCPre-est and MCP, are clearly slower: their running times are in the order of 300–3,700 s. These running times depend on the frequency of re-estimation and the number of MC iterations used. However, even with these high computational costs, they were not able to produce reliable results.

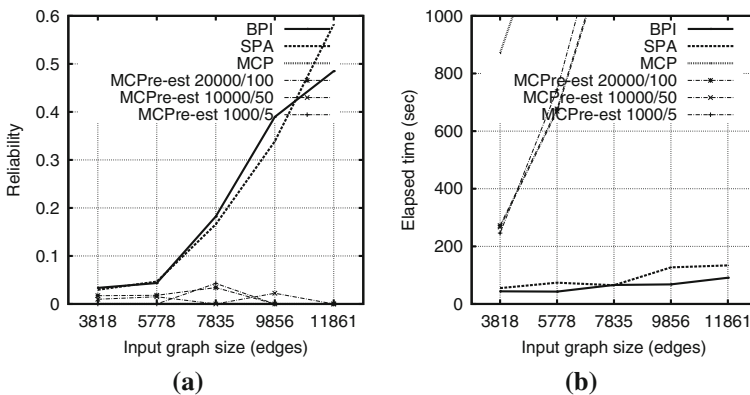


Fig. 3 Reliabilities (a) and running times (b) with a fixed subgraph size (50 edges) and varying input graph size (x -axis)

Based on the results, the proposed methods (BPI, SPA) clearly outperform the simpler methods (MCP, MCPPre-est). With the reasonable ranges of parameter values used in the experiments, this was the case in terms of both the quality of the results, as well as the running time. For the Monte Carlo based methods, the quality/time trade-off can be adjusted by their parameters, but improving one will further degrade the other one. It does not seem plausible that these methods could be useful in practice for large input graphs.

Performance on small datasets While our emphasis is on scalability and large input graphs, it is also interesting to test the methods on a small dataset: (1) For small datasets, it is feasible for MCPPre-est to re-estimate edge relevances after each edge removal, leading to more optimal results. This is an interesting reference for analyzing the quality of the results of BPI and SPA, even if the running time of MCPPre-est may be too large for practical applications. (2) We proposed to use MCP to fine tune the results of BPI and SPA to the requested size, and we are interested in studying how well MCP works in this setting.

The performance of all methods is relatively uniform when the task is to remove edges from a relatively small graph (Dys1 graph of 141 edges). The best reliabilities are obtained by MCPPre-est, when edge relevances are re-estimated with 100,000 MC iterations after each edge removal (Fig. 4). However, this quality comes at a substantial computational cost. The running times of MCPPre-est 100000/1 illustrate its dependence on the number of edges removed, a property that makes the method unsuitable for large graphs.

BPI and SPA have a consistent performance, both in terms of reliability of the result and the computation time. Further, the result indicates that MCP, without re-estimation, is useful on small graphs when relatively few edges are removed (Fig. 4). This is good news for BPI and SPA, which use MCP to adjust their initial result to the required size.

Effect of using several initial solutions for SPA We next study variants of the SPA algorithm. It has the option of building several different series-parallel graphs from different starting points, and using their union as the solution (which is further pruned

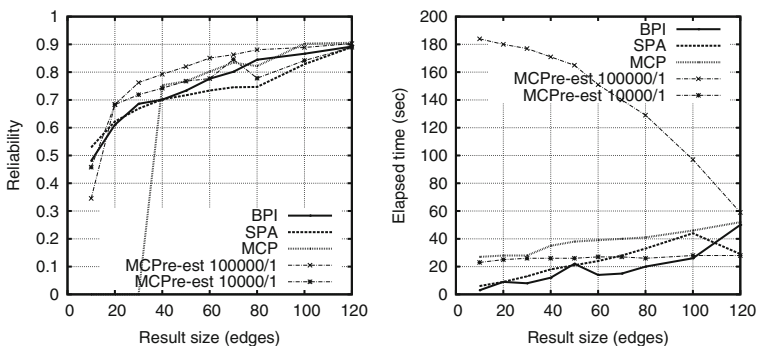


Fig. 4 Reliabilities (left) and running times (right) for the small input graph Dys1 of 141 edges

by MCP to the desired size). By construction (Algorithm 4), this union is unlikely to be series–parallel, and is thus less restricted. Recall that the use of several initial solutions is governed by parameter $C > 1$. As soon as the addition of any path fragment does not improve the reliability of the solution by factor C at least, the algorithm starts to build a new series–parallel graph.

We experimented with values $C = \{1.1, 1.2, 1.3\}$ and compared them against the standard SPA (i.e., $C = 1$). The observation is that $C > 1$, especially $C = 1.1$, often improves the quality (reliability) of the results (four out of five Conn graphs, two out of four Gene graphs, zero out of one Dys graph). However, the quality may actually also decrease (all other cases). Figure 5 gives some illustrative examples. In any case, the use of several initial solutions results in longer running times, with $C = 1.1$ being the most expensive of the tested variants. The reason is that a lower value of C results in fewer but longer constructions of series–parallel components than larger values.

Our conclusion from these experiments is that multiple starts with $C = 1.1$ may be useful to improve the quality of the results, but since the results may also deteriorate and there is a substantial increase in running time, the standard SPA is a safer choice.

Quality of results We now take a closer look at the performance of BPI and SPA (without several initial solutions), and will also consider results for MCPre-est 100000/25 for reference. The optimal reliability is not known, so we have to base the analysis mostly on comparison of the various methods.

We know, however, that the result has to be close to optimal if the reliability is close to one. This is indeed the case for the relatively well connected query nodes of graphs Conn1 (Fig. 6a), Conn2 (Fig. 5a), Conn3, Conn5, and Dys2. In these cases, the original graphs of 800–3,500 edges can be reduced to 20–160 edges while maintaining a reliability of at least 85%.

For the Conn graphs, BPI tends to give better results with a small but a clear margin (Fig. 6a is a representative example, Fig. 6b an exceptional case). Interestingly, in these cases SPA with $C = 1.1, 1.2$, or 1.3 gives practically equal results to BPI (not shown). With large values of C , SPA reduces to BPI: the series–parallel components consist of a single path each. This may happen already with the values of C we used, especially if there are very strong connections between the query vertices.

For the random gene pairs (Gene1–Gene4), the situation is completely the opposite: in most cases, SPA outperforms BPI by a large margin (Fig. 6c gives an example). The relatively poor performance of BPI is likely due to the fact that for distant connections best paths may be strongly correlated, and BPI fails to acquire many independent or partially independent connections in such cases. For the dyslexia vertex pair, the results are mixed (Figs. 3a, 4, and 6d).

In terms of running times, BPI tends to be faster, but this is not always the case. Both BPI and SPA behaved roughly linearly in the size of the extracted subgraph. The variation in the running times is caused by the Monte Carlo Pruning step in the end of the methods, and in the case of BPI, restarts of the algorithm with larger numbers of initial best paths. BPI and SPA are not very sensitive to the size of the input graph. In the scalability test, they both had sublinear running times in the size of the input graph (Fig. 3b). This is in great contrast with the unscalable Monte Carlo based methods we used.

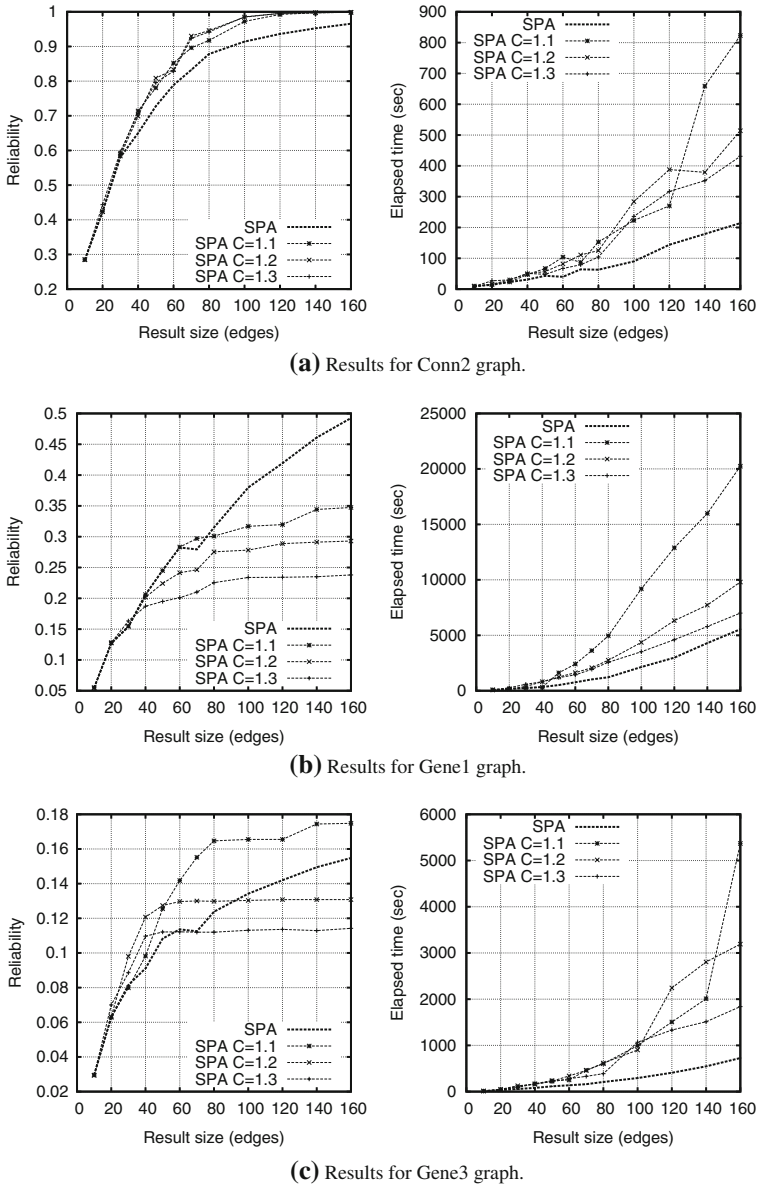
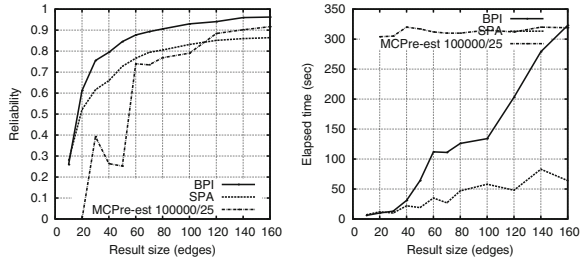


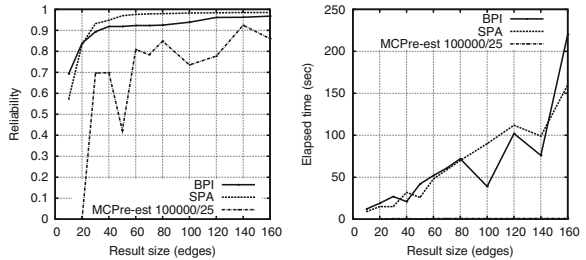
Fig. 5 Subgraph reliabilities (left) and running times (right) of different variants of SPA on some representative graphs

MCPre-est performed uniformly poorly (Fig. 6a, b and d). It was at its best with the smallest input graphs, Dys1 (141 edges, Fig. 4), Dys2 (1,113 edges, Fig. 6d), and Conn1 (853 edges, Fig. 6a), from which it was able to produce some reliable subgraphs. Its running time was usually one or two magnitudes higher than that of BPI or SPA.

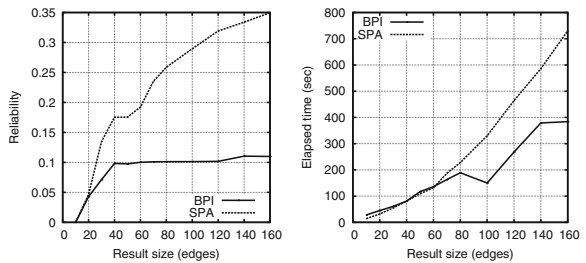
Fig. 6 Subgraph reliabilities (left) and running times (right) of BPI, SPA, and MCPre-est on some representative graphs



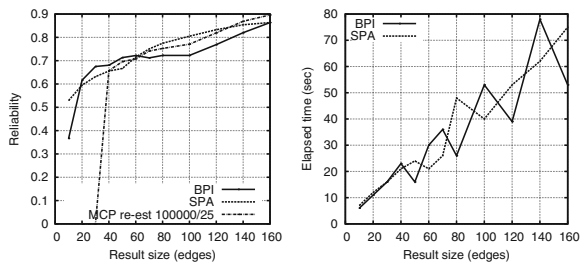
(a) Results for Conn1 graph.



(b) Results for Conn3 graph. MCPre-est elapsed time is approximately constant at 1400 sec.



(c) Results for Gene4 graph. MCPre-est was not run due to its long execution times.



(d) Results for Dys2 graph. MCPre-est elapsed time is approximately constant at 590 sec.

Overall, the series–parallel method SPA seems the most robust across the tests we performed. Using multiple starts with $C = 1.1$ may be useful, but increases the running time. The best path method BPI often produces equally good results, too, but seems clearly inferior for distantly connected vertex pairs. Of course, one can always run both SPA (in its different variants) and BPI, as both methods are relatively fast, and choose the better solution.

6 Conclusions

Graphs are versatile and powerful models for many kinds of multirelational data involving heterogeneous entities. Weighted and especially probabilistic graphs can be used to naturally represent irregular, uncertain, or probabilistic relationships. As graphs grow larger and larger—for example, the World Wide Web and various social networks—scalability of the methods for mining these graphs becomes paramount.

One strategy to cope with an enormous mass of data is to quickly extract a small subset, which can then be further studied with computationally more demanding methods. Obviously, the extracted subset should have the most prominent features of the data to be useful.

We proposed two new methods for subgraph extraction, Best Paths Incremental (BPI) and Series–Parallel Augmentation (SPA). They aim to solve the most reliable subgraph problem, and facilitate scalable and robust extraction of reliable subgraphs from large probabilistic graphs. BPI is tempting as it is conceptually simple, but SPA seems to have a more consistent performance across different situations. We benchmarked BPI and SPA against previous methods that estimate edge relevances by a Monte Carlo technique and prune the original graph. These methods only work for relatively small datasets.

Non-trivial, indirect links in probabilistic graphs can become particularly laborious to discover and analyze due to their potentially myriad dependencies. Focusing on small, robust subgraphs with many independent and non-redundant connections between the vertices of interest can greatly simplify the situation. Reliable subgraphs can be expected to have these features and, as suggested by our experiments on real biological graphs, the proposed methods are able to efficiently find these features using only a fraction of edges from the original input graphs.

In the future, efficient methods should be developed to support multiple terminal vertices, an important generalization of the two-terminal problem we have considered. Another interesting research topic is a generalization of reliable subgraphs: how to define and find, in an unsupervised manner, strongly connected subsets, or clusters of terminals?

Acknowledgements We would like to thank Lauri Eronen, Kimmo Kulovesi and Petteri Sevon for their help and co-operation in Biomine project. This work has been supported by the Academy of Finland Grant 118653 (Algodan) and Tekes, the Finnish Funding Agency for Technology and Innovation.

References

- Birnbaum ZW (1969) On the importance of different components in a multicomponent system. *Multivar Anal* 11:581–592
- Bryant RE (1986) Graph-based algorithms for boolean function manipulation. *IEEE Trans Comput* 35:677–691
- Colbourn CJ (1987) *The combinatorics of network reliability*. Oxford University Press, Oxford
- De Raedt L, Kersting K, Kimmig A, Revedo K, Toivonen H (2008) Compressing probabilistic Prolog programs. *Mach Learn* 70:151–168
- Duffin RJ (1965) Topology of series–parallel networks. *J Math Anal Appl* 10:303–318
- Eppstein D (1998) Finding the k shortest paths. *SIAM J Comput* 28:652–673

- Faloutsos C, McCurley KS, Tomkins A (2004) Fast discovery of connection subgraphs. In: Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining, pp 118–127
- Hershberger J, Maxel M, Suri S (2007) Finding the k shortest simple paths: a new algorithm and its implementation. *ACM Trans Algorithms* 3:45
- Hintsanen P (2007) The most reliable subgraph problem. In: Proceedings of the 11th European conference on principles and practice of knowledge discovery in databases, pp 471–478
- Lawler EL (1972) A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Manage Sci* 18:401–405
- Roditty L (2007) On the k -simple shortest paths problem in weighted directed graphs. In: Proceedings of the 18th annual ACM-SIAM symposium on discrete algorithms, pp 920–928
- Sevon P, Eronen L, Hintsanen P, Kulovesi K, Toivonen H (2006) Link discovery in graphs derived from biological databases. In: Proceedings of data integration in the life sciences. Third international workshop, pp 35–49
- Valdes J, Tarjan RE, Lawler EL (1982) The recognition of series–parallel digraphs. *SIAM J Comput* 11:298–313