Fast Discovery of Reliable k-terminal Subgraphs

Melissa Kasari, Hannu Toivonen, and Petteri Hintsanen

Department of Computer Science and
Helsinki Institute for Information Technology HIIT
PO Box 68, FI–00014 University of Helsinki, Finland
{melissa.kasari,hannu.toivonen,petteri.hintsanen}@cs.helsinki.fi

Abstract. We present a novel and efficient algorithm for solving the most reliable subgraph problem with multiple query nodes on undirected random graphs. Reliable subgraphs are useful for summarizing connectivity between given query nodes. Formally, we are given a graph $G = (V, E)$, a set of query (or terminal) nodes $Q \subset V$, and a positive integer B. The objective is to find a subgraph $H \subset G$ containing Q, such that H has at most B edges, and the probability that H is connected is maximized. Previous algorithms for the problem are either computationally demanding, or restricted to only two query nodes. Our algorithm extends a previous algorithm to handle k query nodes, where $2 \leq k \leq |V|$. We demonstrate experimentally the usefulness of reliable k-terminal subgraphs, and the accuracy, efficiency and scalability of the proposed algorithm on real graphs derived from public biological databases.

1 Introduction

Graphs and networks are powerful means of representing information in various domains such as biology, sociology, and communications. However, large graphs are difficult to understand and use by humans. Given that the user is interested in some particular nodes and their connectivity, a large fraction of the original graph is often irrelevant. Subgraph extraction addresses this problem.

As an example application, consider *Biomine*, a biological graph consisting roughly of a million nodes and eight million edges [10]. One form of a query to Biomine is to specify a small number of query nodes, such as a gene and a disease, and extract a small subgraph that maximally connects the gene to the disease. A subgraph of few dozens of nodes typically already gives a good picture of the connectivity—not only the best paths, but a subgraph describing the network that connects the given nodes. At the same time, almost all of the millions of edges and nodes are irrelevant to how the gene is related to the disease.

In the *most reliable subgraph problem* [3], the user gives query nodes (also called terminals) and a budget, and the task is to extract a subgraph maximally relevant with respect to the given query nodes, but with a size within the given budget. The problem is defined for simple (Bernoulli) random graphs, where edge weights are interpreted as probabilities of the edges, and where a natural definition for "relevance" is network reliability (see Section 2).

In this paper, we propose a novel, efficient algorithm for extracting a reliable subgraph given an arbitrary number of query nodes. Previous work on the most reliable subgraph problem suffers either from a limitation to exactly two query nodes, or from computational complexity. Our work builds on a recent, efficient method for two query nodes, called *Path Covering* [5].

2 The Most Reliable *k*-terminal Subgraph Problem

We define the problem of finding the most reliable k-terminal subgraph, loosely following conventions and notations from previous work [4]. Let $G = (V, E)$ be an undirected graph where V is the set of nodes and E the set of edges. G is a Bernoulli random graph where each edge e 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 $1 - p_e$. Given edge probabilities, the states of edges are mutually independent. Nodes are static.

Given a set $Q \subset V$ of nodes, or *terminals*, the *network reliability* $\mathrm{R}(G, Q)$ of G is defined as the probability that Q is connected, i.e., that any node in Q can be reached from any other node in Q [1]. In *the most reliable subgraph problem* we are looking for a subgraph $H \subset G$ connecting the terminals in Q, such that H has at most B edges and a maximal reliability with respect to the terminals, i.e., find $H^* = \arg\max_{H \subset G, ||H|| \leq B} \mathrm{R}(H, Q)$. Although the problem can be defined for directed graphs as well [3], we focus on undirected graphs in this paper. This problem, like reliability problems in general [12], is inherently difficult: efficient solutions are available only for restricted classes of graphs, but cases on general graphs are most likely intractable [3].

We now introduce some additional notation used in the later sections.

Given a graph G, $V(G)$ is the node set of G and $E(G)$ the edge set of G. Given a set of edges $S \subset E$, we say S *induces* a graph $G(S) = (V', S)$, where V' consists of the endpoints of the edges in S.

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, trees and edges as (induced) graphs when there is no risk of confusion. This makes it notationally easy, e.g., to add a path P to a graph G by writing simply $G \cup P$ instead of $G \cup G(P)$, or to denote the edges of a tree T as $E(T)$ instead of $E(G(T))$.

Finally, a path with endpoints u and v is said to be a u–v-path.

Related Work The most reliable subgraph problem was introduced recently [3], but algorithms were given only for the two-terminal case. We are aware of two previous solutions for the general case. One, proposed by Kroese et al., is based on the cross-entropy method [7]. De Raedt et al. give other solution to the general case in the setting of theory compression for ProbLog [9, 8]. Unfortunately, these methods do not scale well to large databases, where input graphs may have hundreds or thousands of edges. Other closely related work includes connection subgraphs [2], center-piece subgraphs [11], and proximity graphs [6].

Recently, a novel Monte-Carlo sampling based algorithm Path Covering (PC) has been proposed for the two-terminal case [5]. The method proposed in this paper is based on the ideas of PC, so we briefly review its principles. The algorithm has two phases: a path sampling phase and a subgraph construction phase. In the path sampling phase, the goal is to identify a small set C of paths that have high probabilities and are relatively independent of each other. This is achieved by approximately maximizing the probability that at least one path $P \in C$ is true. We denote this probability by $\Pr(C) = \Pr(\bigvee_{P \in C} P)$.

In the subgraph construction phase, PC chooses a set of solution paths $S \subset C$ such that the probability $\Pr(S) = \Pr(\bigvee_{P \in S} P)$ is maximized and $||G(S)|| \leq B$, where $||G||$ denotes the number of edges in G. PC does not maximize $\mathrm{R}(G(S))$ directly, but works on its lower bound $\Pr(S)$ instead. Concisely put, PC generates S iteratively by choosing at each iteration the path P^* which gives the maximal per-edge increase to the (estimated) probability $\Pr(S)$, that is

$$P^* = \underset{P \in C \setminus S}{\arg\max} \frac{\Pr(S \cup P) - \Pr(S)}{|E(P) \setminus E(H)|}, \tag{1}$$

where $H = G(S)$ is the result subgraph being constructed. At each iteration, paths that become included into H are removed from C. To satisfy the budget constraint, paths $P \in C$ for which $||H|| + |E(P) \setminus E(H)| > B$ are also removed. The algorithm stops when $||H|| = B$ or $C \setminus S = \emptyset$, and returns the subgraph H.

3 Algorithms

We propose a novel, efficient algorithm for the problem of extracting a reliable k-terminal subgraph from an undirected graph. The proposed algorithm is a generalization of the Path Covering (PC) method [5] (see Section 2 for a brief overview) to more than two query nodes. The basic principles remain the same: the two phases, use of Monte Carlo simulations, as well as many subtle details. However, whereas PC uses paths connecting the two given query nodes as its building blocks (set C) in the subgraph construction phase, here we consider spanning trees connecting the k query nodes, with $2 \leq k \leq |V|$. Similarly, set S in the objective function (1) consists of spanning trees instead of paths as in PC.

In the first phase, the algorithm extracts a set of trees from the original graph G. Each of the trees connects the given k query nodes; by construction, they are spanning trees having the query nodes as leaves. In the second phase, these trees are used as building blocks to construct the result of the algorithm just like PC uses paths as its building blocks. We focus on the novel aspects of the proposed algorithm, the ones that allow solving the k-terminal problem. For brevity, we omit technical details shared with Path Covering and described in depth elsewhere [5]. We begin by presenting the general aspects of the new algorithm and then proceed to more detailed description.

Input and output data. The first phase of the algorithm (Algorithm 1) takes a random graph G and a set $Q \subset V$ of query nodes as its input. The algorithm outputs a set C of trees such that each tree connects all the query nodes. We call these trees *candidate trees.* C is used as an input in the second phase of the algorithm (Algorithm 2).

Producing candidate trees. At the first iteration, $(|Q|^2 - |Q|)/2$ new candidate trees are generated (Lines 2–3). Each tree connects one pair of query nodes and each query node pair is connected by one tree. Later, as the algorithm proceeds, new trees are added one by one; each of the later trees is also initially formed as a path between two query nodes. During the algorithm, individual trees are created and grown iteratively. In each iteration, either a branch is added to an existing *incomplete tree* (a tree that does not yet connect all query nodes) so that a new query node is connected to the tree, or a new initial tree is generated. At the end of the algorithm we output only *complete trees* (trees that connect all query nodes).

Edge sampling. The algorithm is stochastic. At each iteration it randomly decides, according to the probabilities p_e, which edges exist and which do not (Line 5). Only edges that are included in at least one candidate tree are decided. All other edges are considered to exist. The next step is to determine if any of the previous candidate trees exist in the current graph realization (Line 9). If one does not exist a new candidate tree is generated (Lines 14–15). If a previously discovered tree exists, the first such tree is taken into examination (Line 10). If the tree is complete, the algorithm proceeds directly to the next iteration. Otherwise the tree is extended (Lines 17–21) before continuing to the next iteration.

Tree construction. A new tree is formed by the best path connecting two query nodes (Line 15). A previously established incomplete tree is extended by connecting a new query node to it with the best path between some node in the tree and the new query node (Lines 18–20). The probabilities of all edges in the tree are set to 1 prior to the search of the best path (Line 17), while the probabilities of other edges remain the same. As a result the new branch is formed by the best path between the new query node and the tree. Edges that do not exist at the iteration are not used. All edge weights are set to their original values before proceeding to the next iteration (Line 21).

Choosing query nodes. When a new tree is formed, the algorithm decides randomly which two query nodes are included in the initial tree. Later on when an incomplete tree is extended, the algorithm again randomly selects the new query node to connect to the tree. This is to avoid unnecessary complexity: in our experiments this solution produced better results and shorter running times than selecting the node to be added based on its distance from the tree (results not shown).

Discovering strong trees. The collection C of candidate trees is organized as a queue, i.e., the oldest candidate trees are always considered first. This drives the algorithm to complete some trees first (the oldest ones) rather than extending them in random order and not necessarily up to a completion. On the other hand, the stochasticity of the algorithm favors strong trees: they are more likely to be true at any given iteration and thus more likely to be extended. The algorithm also has a tendency to avoid similar trees: when two (partial) trees are true at the same time, only the oldest one is potentially extended.

Algorithm 1 Sample trees

```
Input: Random graph $G$, set $Q \subset V$ of query nodes, number of trees to be generated
Output: Collection $C$ of trees connecting all query nodes
 1: $C \leftarrow \emptyset$
 2: for each node pair $v_i$, $v_j \in Q$, $v_i \neq v_j$ do
 3:    Add the best $v_i$–$v_j$-path from $G$ to $C$
 4: repeat
 5:    Decide each $e \in E(C)$ by flipping a coin biased by the probability of $e$
 6:    Let $E_S$ contain the successful edges and $E_F$ contain the failed edges
 7:    $T_S \leftarrow \emptyset$
 8:    for $i = 1$ to $|C|$ do
 9:       if $E(C_i) \subset E_S$ then
10:          if $Q \subset V(C_i)$ then
11:             continue at line 5
12:          $T_S \leftarrow C_i$
13:          continue at line 17
14:    Randomly select $u$ and $v \in Q$, $u \neq v$
15:    Add the best $u$–$v$-path from $G - E_F$ to $C$
16:    continue at line 5
17:    Set the probability of $e$ to 1 for all $e \in E(T_S)$
18:    Randomly select $u \in Q \cap V(T_S)$ and $v \in Q \setminus V(T_S)$
19:    $P_S \leftarrow$ the best $u$–$v$-path from $G - E_F$
20:    Add all $e \in E(P_S)$ to $E(T_S)$ and all $v \in V(P_S)$ to $V(T_S)$
21:    Reset edge weights for all $e \in E(T_S)$
22: until the stopping condition is satisfied
23: return $C$ (after removing all incomplete trees)
```

Stopping condition. We use the number $|C|$ of complete trees generated as the stopping condition for the first phase. The number of iterations would be another alternative (see Section 4). Using the number of trees seems a better choice than using the number of iterations, since the minimum number of iterations needed to produce a single complete tree increases when the number of query nodes increase.

4 Experiments

We have implemented and experimentally evaluated the proposed algorithm on random graphs derived from public biological databases. In this section, we examine the usefulness of k-terminal subgraphs: do they maintain the k-terminal reliability of input graphs, and what is the amount of random variance in the results? We demonstrate the efficiency and scalability of our algorithm for large input graphs. Finding a suitable stopping criterion for Algorithm 1 is difficult; we also address this issue. Finally, we compare the algorithm against a baseline method based on enumerating the best paths between the query nodes.

Algorithm 2 Select trees

```
Input: Random graph $G$, collection $C$ of trees, budget $B \in \mathbb{N}$
Output: Subgraph $H \subset G$ with at most $B$ edges
$S \leftarrow \emptyset$
while $||G(S)|| \leq B$ and $C \setminus S \neq \emptyset$ do
    Find $T = \arg\max_{T \in C \setminus S} \frac{\Pr(S \cup T) - \Pr(S)}{|E(T) \setminus E(G(S))|}$
    $S \leftarrow S \cup T$
    for all $T \in C$ do {remove useless trees}
        if $T \subset G(S)$ or $||G(S)|| + |E(T) \setminus E(G(S))| > B$ then
            $C \leftarrow C \setminus \{T\}$
return $H = G(S)$
```

4.1 Test set-up

Test data and query nodes We use *Biomine* database [10] as our data source. Biomine is a large index of various interlinked public biological databases, such as EntrezGene, UniProt, InterPro, GO, and STRING. Biomine offers a uniform view to these databases by representing their contents as a large, heterogeneous biological graph. Nodes in this graph represent biological entities (records) in the original databases, and edges represent their annotated relationships. Edges have weights, interpreted as probabilities [10]. We evaluated the proposed method using six source graphs of varying sizes (Table 1) and a set of up to ten query nodes. They were obtained as follows.

Name of $G = (V, E)$	**400**	**500**	**700**	**1000**	**2000**	**5000**
Number of edges, $\|E\| = \|\|G\|\|$	395	499	717	1046	2019	4998
Number of nodes, $\|V\|$	153	189	260	336	579	1536
Reliability of G with $\|Q\| = 4$	0.46	0.46	0.50	0.56	0.59	0.60

Table 1. Sizes of source graphs used in the experiments

First, the largest subgraph, consisting of approximately 5000 edges and 1500 nodes, was retrieved from the Biomine database using *Crawler*, a subgraph retrieval component proprietary to Biomine. For this initial retrieval, we used eight randomly selected query nodes.

Second, a set of ten query nodes to be used in the experiments was defined by randomly picking nodes from the subgraph of 5000 edges. The query node identifiers are EntrezGene:348, EntrezGene:29244, EntrezGene:6376, EntrezGene:4137, UniProt:P51149, UniProt:Q91ZX7, EntrezGene:14810, UniProt: P49769, EntrezGene:11810, and UniProt:P98156.

Third, smaller subgraphs were retrieved with Crawler by a sequence of subgraph retrievals, always extracting the next smaller subgraph from the previous subgraph, using the ten query nodes given above.

We also used two additional source graphs when evaluating the scalability of the algorithm to large source graphs. The smaller one consisted of 51,448 edges and 15,862 nodes. The size of the larger graph was 130,761 edges and 66,990 nodes. The smaller graph was a subgraph of the larger one and all other source graphs used in the experiments were subgraphs of both.

Biomine Crawler The subgraph retrieval component of the Biomine system, "Crawler", was used to extract the source graphs, and it will also be used below in a comparative experiment to assess the effectiveness of the proposed algorithm. Crawler is currently undocumented. Given a source graph, a set of query nodes, and a node budget, it works roughly as follows. It first finds a large set of best paths between all pairs of query nodes. It then picks those paths sequentially between the node pairs, until the subgraph induced by the chosen paths reaches the specified number of nodes. The method is somewhat similar to the BPI algorithm [4], but works for multiple query nodes. Even though the Crawler works with random graphs, it does not try to optimize the k-terminal reliability.

Stopping condition We used the number of complete candidate trees generated as the stopping condition for Algorithm 1. Another obvious alternative would be the number of iterations. Neither condition is perfect: for instance, the number of query nodes has a strong effect on the number of trees needed to find a good subgraph. On the other hand, the number of query nodes has also a strong effect on the number of iterations needed to produce a sufficient amount of trees. A single fixed number of candidate paths is a suitable stopping condition for the two-terminal case [5] but it is problematic in the k-terminal case where the building blocks are trees consisting of multiple branches. For the current experiments, we believe a fixed number of candidate trees gives a fair impression of the performance of the method.

Parameters The experiments have been performed using the following parameter values; the default values we have used throughout the experiments, unless otherwise indicated, are given in boldface.

- Size of source graph G (Table 1): $||G||$ = 400, **500**, 700, 1000, 2000, 5000
- Number of query nodes: $|Q|$ = 1, 2, 3, **4**, . . ., 10
- Size of extracted subgraph: $||H||$ = 10, 20, 30, . . ., **60**, . . ., 100, 150, 200
- Number of complete trees (stopping condition of Algorithm 1): $|C|$ = 10, 20, 30, . . ., **100**, 150, 200

To control random variation, the values reported below are averages over 10 independent runs.

4.2 Results

Let us first take a look at how well the method manages to extract a reliable subgraph (Fig. 1). For three to four query nodes ($|Q|$), a subgraph of only 20–30 edges manages to capture 80% of the reliability of the source graph of 500 edges. For a large number of query nodes, the problem seems much more challenging. It seems obvious, that larger subgraphs are needed for a larger number of query nodes, if the reliability should be preserved.

The number $|C|$ of candidate trees produced in the first phase of the algorithm has an effect on the reliability of the extracted subgraph, but we discovered that sampling a relatively small number of trees is enough to produce good subgraphs (approximately 50 trees for four query nodes; results not shown). An experimental analysis of the running time indicates that the method scales linearly with respect to the number of candidate trees generated

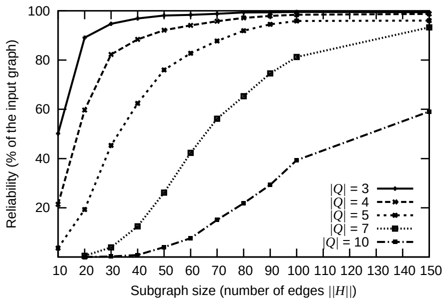

Fig. 1. Relative reliability of a subgraph as a function of its size, for various numbers of query nodes.

The scalability of the new algorithm to large source graphs (Fig. 2, left) is clearly superior to previous methods (see Section 1). Source graphs of thousands of edges are handled within a second or two. Scalability is close to linear, which is expected: the running time of the algorithm is dominated by Monte-Carlo simulation, whose complexity grows linearly with respect to the input graph size and the number of iterations. Running times for the two additional large source graphs (51,448 edges and 130,761 edges) are not shown in the figure, but the average running times over ten independent runs are approximately 16 (standard deviation 0.57) and 53 seconds (standard deviation 2.8), respectively. Limiting the length of tree branches might shorten running times in some cases.

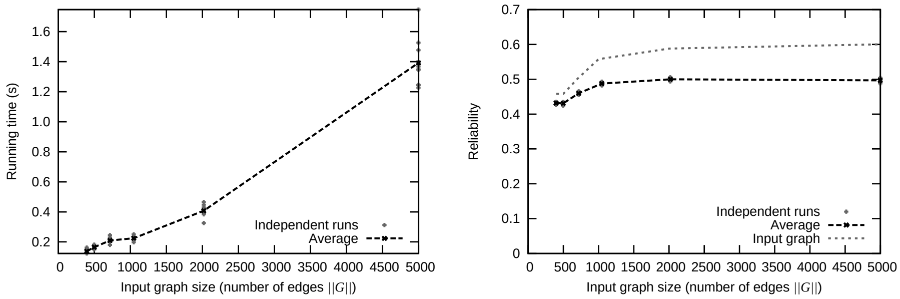

Fig. 2. Running time and reliability as functions of the size of the input graph.

The right panel of Fig. 2 indicates the original reliability of the growing source graph, as well as the reliability of the extracted subgraph (of a fixed size of 60 edges). The relative difference in reliability is less than 20% in all cases, emphasizing the ability of the algorithm to preserve strong connectivity between the query nodes. These results suggest that the algorithm is competitive for interactive visualization.

Finally, we compare the new algorithm to the Biomine Crawler (Fig. 3), as it is the only available method for comparison on this scale. The comparison is not completely fair, as the Crawler does not aim to optimize the k-terminal reliability, but the general goal of extracting a subgraph connecting the query nodes is the same. In the experiments, the proposed algorithm produces significantly more reliable subgraphs, especially when the extracted subgraphs are small. Both methods converge towards the reliability of the source graph. However, where the new method reaches 80% of the original reliability with only 30 edges, the Crawler needs 60 edges for the same.

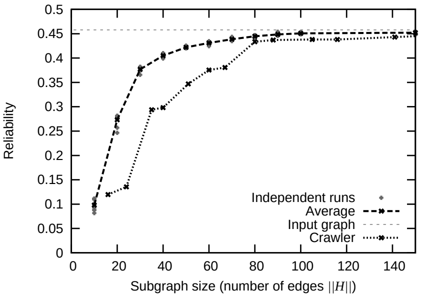

Fig. 3. Comparison with Crawler using 4 query nodes.

5 Conclusions

We gave an efficient algorithm for solving the most reliable subgraph extraction problem for more than two query nodes. This is a significant improvement over state-of-the-art that could efficiently only handle exactly two query nodes.

Experimental results with real biological data indicate that the problem and the proposed method have some very useful properties. First, reliable k-terminal subgraphs of fairly small sizes seem to capture essential connectivity well. Second, the proposed method extracts a reliable subgraph in a matter of seconds, even from a source graph of thousands of edges; the time complexity seems to be linear with respect to the size of the original graph.

There are many possible variants of the approaches described in this paper that could be explored to find better solutions. For instance, how to choose which partial tree to expand and how to expand it, or how to efficiently use partial trees also in the second phase? Another interesting approach could be using (approximated) Steiner trees as spanning trees.

Future experiments include systematic tests to find out robust sets of parameters that perform reliably over wide range of input graphs and query nodes, and more extensive comparisons with related methods. Possible extension of the proposed algorithm for directed variant of the problem is an open question.

Acknowledgements. We would like to thank the Biomine team and especially Lauri Eronen for providing the Biomine Crawler. This work has been supported by the Algorithmic Data Analysis (Algodan) Centre of Excellence of the Academy of Finland (Grant 118653).

References

1. Colbourn, C.J.: The Combinatorics of Network Reliability. Oxford University Press (1987)
2. Faloutsos, C., McCurley, K.S., Tomkins, A.: Fast discovery of connection subgraphs. In: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 118–127 (2004)
3. Hintsanen, P.: The most reliable subgraph problem. In: Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases. pp. 471–478 (2007)
4. Hintsanen, P., Toivonen, H.: Finding reliable subgraphs from large probabilistic graphs. Data Mining and Knowledge Discovery 17, 3–23 (2008)
5. Hintsanen, P., Toivonen, H., Sevon, P.: Fast discovery of reliable subnetworks (2010), submitted
6. Koren, Y., North, S.C., Volinsky, C.: Measuring and extracting proximity graphs in networks. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 245–255 (2006)
7. Kroese, D.P., Hui, K.P., Nariai, S.: Network reliability optimization via the cross-entropy method. IEEE Transactions on Reliability 56, 275–287 (2007)
8. Raedt, L.D., Kersting, K., Kimmig, A., Revoredo, K., Toivonen, H.: Compressing probabilistic Prolog programs. Machine Learning 70, 151–168 (2008)
9. Raedt, L.D., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic prolog and its application in link discovery. In: Proceedings of the 12th International Joint Conference on Artificial Intelligence. pp. 2468–2473 (2007)
10. Sevon, P., Eronen, L., Hintsanen, P., Kulovesi, K., Toivonen, H.: Link discovery in graphs derived from biological databases. In: Proceedings of Data Integration in the Life Sciences, Third International Workshop. pp. 35–49 (2006)
11. Tong, H., Faloutsos, C.: Center-piece subgraphs: problem definition and fast solutions. In: Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 404–413 (2006)
12. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM Journal on Computing 8, 410–421 (1979)