Fast Discovery of Reliable Subnetworks

Petteri Hintsanen
Department of Computer Science and HIIT
University of Helsinki
petteri.hintsanen@cs.helsinki.fi

Hannu Toivonen
Department of Computer Science and HIIT
University of Helsinki
hannu.toivonen@cs.helsinki.fi

Petteri Sevon
Biocomputing Platforms Ltd.
petteri.sevon@bcplatforms.com

***Abstract*—We present a novel and efficient algorithm, PATH COVERING, for solving the most reliable subgraph problem. A reliable subgraph gives a concise summary of the connectivity between two given individuals in a social network. Formally, the given network is seen as a Bernoulli random graph $\mathcal{G}$, and the objective is to find a subgraph $\mathcal{H} \subset \mathcal{G}$ with at most B edges such that the probability that a path exists in $\mathcal{H}$ between the given two individuals is maximized. The algorithm is based on an efficient stochastic search of candidate paths, and the use of Monte-Carlo simulation to cast the problem as a set cover problem. Experimental evaluation on real graphs derived from DBLP bibliography database indicates superior performance of the proposed algorithm.**

I. INTRODUCTION

Link discovery in social networks is a research area with numerous applications. We address the problem of identifying a subnetwork (or community) that connects two given persons of a large network. This task can also be seen as information retrieval: given two individuals, return other persons and relations that are maximally relevant to connecting the given pair of individuals. Fig. 1 gives an example of a collaboration subnetwork between two researchers, extracted from DBLP computer science authorship network. Applications of subnetworks include analysis and description of potential collaborations, discovery of hidden relationships for instance in criminology, and description of possible viral effects between given individuals.

We propose a novel application of Bernoulli random graphs to the subnetwork extraction problem on social networks. Our main contribution is a new, effective and efficient method for the problem. We illustrate and experimentally evaluate the proposed method on the DBLP network. The results indicate superior performance over previous methods on the same task.

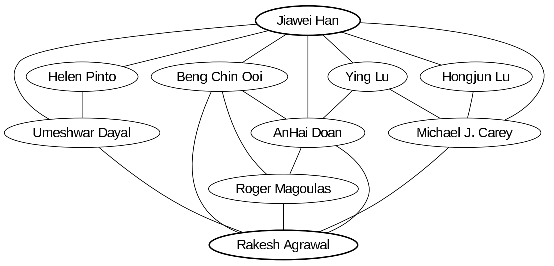


Fig. 1. An excerpt from a reliable connection subgraph between Rakesh Agrawal and Jiawei Han.

Specifically, we consider the *most reliable subgraph problem* [1]. Let $\mathcal{G}$ be a weighted random graph where edges have mutually independent probabilities of being true. Given two nodes of $\mathcal{G}$ and a budget B, the task is to extract a subgraph in which the probability of the specified nodes being connected is maximized, subject to the number of edges being limited to B.

Obviously the problem is domain independent. Other application areas with large, graph-structured data collections include communication networks, the web, and biological networks such as protein interaction graphs. Search and retrieval of relevant information from such graphs has been researched a lot, but the majority of work has been on identifying important nodes (such as web search results) rather than relevant subgraphs. We will briefly review related work in the next section.

II. Problem definition and related work

We view a given weighted social network as a Bernoulli random graph. In other words, we assume a network of individuals and their relations are given, and that relations have weights between 0 and 1. In typical applications, the weights reflect the strength of a friendship, the relative frequency of interaction, or the probability of influence between individuals. The subgraph extraction task only looks at the structure of the network, including the edge weights, and ignores any other attributes of the individuals or the relations.

Formally, we define the problem of finding the most reliable subgraph following conventions and notations from previous work [2]. Let $\mathcal{G} = (V, E)$ be a graph where V is the set of nodes (e.g. persons) and E the set of edges (relations between persons). Further, each edge $e \in E$ has an associated probability $p(e)$: we say that relation e exists (or is true or successful) with probability $p(e)$, and conversely e does not exist (or is not true or fails) with probability $1 - p(e)$. Edge states are assumed to be mutually independent, and nodes are static. Consequently, the probability of a path P being true is $\Pr(P) = \prod_{e \in P} p(e)$, that is, the probability that all of its edges are true. Finally, given two *terminal* nodes $s, t \in V$, the *two-terminal network reliability* $R(\mathcal{G})$ of $\mathcal{G}$ is defined as the probability that s and t are connected by a path in $\mathcal{G}$ after all edges in E have been randomly decided according to their probabilities [3].

Our focus in this paper is on *the most reliable subgraph problem*. In this problem we are given, in addition to $\mathcal{G}$ and the terminal nodes, a positive integer B. The task is to find a subgraph $\mathcal{H} \subset \mathcal{G}$ such that $\mathcal{H}$ has at most B edges and a maximal reliability with respect to the terminals [1]. In other words, we are looking for

$$\mathcal{H} = \underset{\mathcal{H}' \subset \mathcal{G}, ||\mathcal{H}'|| \leq B}{\arg\max} R(\mathcal{H}'),$$

where $||\mathcal{H}'||$ denotes the number of edges in $\mathcal{H}'$. This problem, like reliability problems in general [4], is inherently difficult. Efficient solutions are available for restricted classes of graphs, but cases on general graphs are most likely intractable [1].

The simple Bernoulli random graph model is a strong tool for subgraph extraction. First, it allows a relatively simple but elegant definition of the strength of the overall connection ("reliability") between two given nodes. Second, the subgraph extraction task then has a natural objective: choose the subgraph so that the overall connection in the subgraph is maximized. This guarantees that the resulting subgraph is maximally relevant *as a whole*. In other words, the best subgraph is not the collection of best paths, much less a collection of best bridging individuals.

Network reliability has first been applied on communication networks. There, links between devices may fail, and a network optimization task is to find a network structure that provides maximal reliability between given devices. Hence the name *maximal reliability* [3], [5]–[8].

Extraction of subgraphs connecting user-given query nodes has recently gained some research interest in data mining and in various application fields [1], [2], [9]–[14]. Some of these methods address the most reliable subgraph problem. Kroese et al. proposed a solution based on the cross-entropy method [8]. It has guarantees of optimality and a more general model with individual edge costs. De Raedt et al. give another solution in the setting of theory compression for ProbLog, a probabilistic Prolog [13]. Unfortunately, both methods are computationally demanding. According to the original publications, the running time of the cross-entropy method on a graph with 51 links ranged from 299 seconds to 361 minutes, depending on the exact method and its parameters [8]. ProbLog theory compression took 3–40 minutes on graphs with 200–1,400 edges, depending on the tolerance allowed in reliability estimates [13]. With the method proposed in this paper, we can extract a reliable subgraph from a graph of 10,000 nodes and 23,000 edges in a matter of seconds.

We have previously proposed two practical algorithms, BPI and SPA, for the most reliable subgraph extraction problem [2]. BPI is based on the use of best paths as building blocks, and SPA works on the restricted class of series-parallel graphs. Other closely related work includes connection subgraphs [9], center-piece subgraphs [11], and proximity graphs [12]. Their general motivations and goals are similar to ours, but the underlying models and optimization problems are different. For connection subgraphs, the subgraph model is taken from conductance in electric circuits, and for center-piece and proximity subgraphs it is based on random walks. We feel that the maximization of the reliability is an elegant and well justified optimization criterion for the chosen random graph model.

Monte-Carlo sampling is an efficient approximation method for many computationally complex problems [15], including network reliability [16]. Since we use Monte-Carlo sampling extensively, we briefly review its basic principle here. In the crude Monte-Carlo method for estimating network reliability between terminals s and t, one draws N independent samples, or *realizations* G_i, $1 \leq i \leq N$, from the random graph $\mathcal{G}$. Each realization is generated by simulating the existence of each edge of $\mathcal{G}$ randomly and independently. To estimate reliability $R = R(\mathcal{G})$, one counts the number K of those realizations G_i which contain at least one s–t-path. Then $\tilde{R} = K/N$ is an unbiased estimator for R with variance $R(1-R)/N$.

III. The Path Covering algorithm

We now propose a novel algorithm for solving the most reliable subgraph problem. The algorithm, called Path Covering (PC), builds the result subgraph $\mathcal{H}$ by incrementally adding s–t-paths one by one to an initially empty subgraph. In this respect, the approach is similar to some previous proposals [2], [9], [12].

This incremental approach has two fundamental sub-problems. First, the number of possible s–t-paths is exponential in the number of edges. Second, evaluating the reliability of even a single subgraph instance $\mathcal{H}$ could take an exponential time. The proposed method negotiates these obstacles in two phases, a *path sampling* phase and a *subgraph construction* phase.

In the path sampling phase, PC gathers a relatively small set $\mathcal{C}$ of *candidate paths* from the set of all s–t-paths in $\mathcal{G}$. Then, in the subgraph construction phase, PC aims to choose an optimal subset $\mathcal{P}$ of the candidate paths in $\mathcal{C}$, according to the edge budget B, and returns the subgraph $G(\mathcal{P})$ induced by $\mathcal{P}$. (We say that a set of paths $\mathcal{P}$ *induces* a graph $G(\mathcal{P}) = (V, E)$, where $V = \{u : \{u, v\} \in P, P \in \mathcal{P}\}$ and $E = \{e \in P : P \in \mathcal{P}\}$.)

Both phases address the same general problem: choose a subset $\mathcal{P}$ of available s–t-paths to induce a reliable subgraph. Furthermore, both phases use a similar strategy to achieve this goal by iteratively and greedily maximizing $\Pr(\mathcal{P}) = \Pr(\bigvee_{P \in \mathcal{P}} P)$, that is, the probability that at least one of the paths in $\mathcal{P}$ is true. The main difference is that the path sampling phase scales to large inputs with exponentially many paths, while the subgraph construction phase produces a better optimized subgraph $G(\mathcal{P})$ with a larger computational cost per path. We give detailed descriptions of the two phases below.

A. Phase 1: Path sampling

In the first phase of Path Covering, we use an iterative strategy to construct the set $\mathcal{C}$ of candidate paths efficiently. The most probable, or *best*, s–t-path is used as the initial candidate path. Then we augment $\mathcal{C}$ in each iteration with a path P such that $\Pr(\mathcal{C} \vee P)$ is approximately maximized. Let $\overline{\mathcal{C}}$ denote an event where none of the paths in $\mathcal{C}$ exists. Since

$$\Pr(\mathcal{C} \vee P) = \Pr(\mathcal{C} \vee (\overline{\mathcal{C}} \wedge P)) = \Pr(\mathcal{C}) + \Pr(\overline{\mathcal{C}} \wedge P),$$

we are looking for the most probable s–t-path P under the condition that all current paths in $\mathcal{C}$ fail. We implement this idea with Monte-Carlo simulation by randomly realizing edges in each iteration according to their probabilities: an edge e is decided to exist with probability $p(e)$ and to not exist otherwise. A cut $\overline{\mathcal{C}}$ is found if every candidate path has at least one edge that does not exist. Then we can add a new s–t-path to $\mathcal{C}$, if one exists.

The algorithm in Fig. 2 implements the path sampling phase. It is mostly looking for a cut event $\overline{\mathcal{C}}$ in each of its iterations (lines 3–15). It realizes edges during the process only when needed, and

Fig. 2. Path sampling algorithm

```
Input: Random graph $\mathcal{G} = (V, E)$, terminal nodes $s$ and $t$,
    number of candidate paths $N$
Output: Set $\mathcal{C}$ of $s$–$t$-paths
 1: $\mathcal{C} \leftarrow$ the best $s$–$t$-path from $\mathcal{G}$
 2: Set $w(e) = -\log(p(e))$ for all $e \in E$
 3: repeat
 4:     Reset all $e \in E$ as undecided
 5:     for all $P \in \mathcal{C}$ do
 6:         for all $e \in P$ do
 7:             if $e$ has not been decided then
 8:                 Decide $e$ as successful with probability $p(e)$,
                    failed otherwise
 9:             if $e$ has failed then
10:                 continue from line 5 with next $P$
11:         go to 4 {$P$ exists}
12:     Find the best $s$–$t$-path $P$ from $\mathcal{G}$ using edge weights $w$,
        deciding edges as necessary
13:     if $P \neq \emptyset$ then
14:         $\mathcal{C} \leftarrow \mathcal{C} \cup P$
15: until $|\mathcal{C}| = N$
16: return $\mathcal{C}$
```

applies the following two rules to avoid unnecessary work. (1) When checking if a path exists, the rest of the path can be ignored as soon as a failed edge is encountered (line 10). (2) When an existing path has been found, the remaining paths can be ignored (line 11). When a cut $\overline{\mathcal{C}}$ does take place, we find a new candidate path among the non-failed edges (line 12). Any shortest path algorithm can be applied with edge weights $w(e) = -\log(p(e))$. Namely, let P be the shortest path found. Then

$$\begin{aligned} w(P) &= \sum_{e \in P} -\log(p(e)) = -\log(\prod_{e \in P} p(e)) \\ &= -\log(\Pr(P)), \end{aligned}$$

and since the logarithm function is strictly increasing and $w(P)$ is minimized, $\Pr(P)$ is maximized. Edge decisions can be integrated into best path search, too, and carried out whenever an undecided edge is encountered. The choice of the number of sampled candidate paths N is nontrivial. We will return to the issue in Section IV.

A possible problem in the path sampling algorithm is that cut events $\overline{\mathcal{C}}$ are rarely found when $\Pr(\mathcal{C})$ is high. In such cases the algorithm could require unacceptably many iterations before stopping. A more effective approach would be to draw random realizations of $\mathcal{G}$ directly under the condition that no path in $\mathcal{C}$ is true. Unfortunately, given the potential dependencies between paths in $\mathcal{C}$, it is difficult to do this exactly.

Fig. 3. Cut sampler

```
1: $\mathcal{F} \leftarrow \{P \in \mathcal{C} : P \text{ is true}\}$
2: while $\mathcal{F} \neq \emptyset$ do
3:     Let $E(\mathcal{F}) = \{e \in P : P \in \mathcal{F}\}$
4:     Let $e^* = \arg\max_{e \in E(\mathcal{F})} |\{P \in \mathcal{F} : e \in P\}|$
5:     Re-decide $e^*$ as failed
6:     $\mathcal{F} \leftarrow \{P \in \mathcal{F} : e^* \notin P\}$
```

As an alternative algorithmic variant, we propose the following approximation where we deliberately "fail" edges of candidate paths until all candidate paths have been broken. First, edges are realized until all paths in $\mathcal{C}$ have been decided—even if some paths are found to exist. Then, if some paths exist, we iteratively and greedily fail the edge e which intersects the largest number of true paths in $\mathcal{C}$ until no true paths remain in $\mathcal{C}$. If there are more than one such edge, we choose the one with the smallest probability $p(e)$. This modification is implemented by removing line 11 of the path sampling algorithm (Fig. 2) and adding the cut sampler (Fig. 3) just before line 12.

B. Phase 2: Subgraph construction

In the second phase of PATH COVERING, we take the set $\mathcal{C}$ of candidate paths generated in the first phase, choose a subset $\mathcal{P} \subset \mathcal{C}$ having at most B unique edges in total, and return the subgraph $G(\mathcal{P}) \subset \mathcal{G}$ induced by them. The objective is to choose the set $\mathcal{P}$ of paths such that the reliability $R(G(\mathcal{P}))$ is maximized.

Exhaustive search and evaluation of all feasible subsets is intractable, even though the number of candidate paths $\mathcal{C}$ is assumed to be relatively small. We relax the problem by maximizing the probability $\Pr(\mathcal{P}) = \Pr(\bigvee_{P \in \mathcal{P}} P)$ instead. It is a lower bound of $R(G(\mathcal{P}))$ and is easier to evaluate—but still requires exponential time in the worst case. Therefore we resort to Monte-Carlo approximation of probabilities. This choice also allows us to cast the path selection task as a set cover problem.

In the path selection algorithm (Fig. 4), we first draw N random realizations G_i of the whole graph $G(\mathcal{C})$ (line 3). Let $C(P) = \{i : P \in G_i\}$ be the *cover set* of each path $P \in \mathcal{C}$, that is, the

Fig. 4. Path selection algorithm

```
Input: Set $\mathcal{C}$ of $s$–$t$-paths, integer $B$
Output: A reliable subgraph $\mathcal{H} \subset G(\mathcal{C})$ with at most $B$ edges
 1: $\mathcal{P} \leftarrow \emptyset$
 2: Remove all paths with more than $B$ edges from $\mathcal{C}$
 3: Generate $N$ realizations $G_i$ of $G(\mathcal{C})$
 4: For each $P \in \mathcal{C}$, let $C(P) = \{i : P \in G_i\}$
 5: while $B > 0 \wedge \mathcal{C} \neq \emptyset$ do
 6:     Choose $P^* = \arg\max_{P \in \mathcal{C}} \tilde{s}(P)$ using (3)
 7:     if $\tilde{s}(P^*) = 0$ then
 8:         go to 4
 9:     $B \leftarrow B - w(P^*)$
10:     Add $P^*$ to $\mathcal{P}$ and remove it from $\mathcal{C}$
11:     Remove all paths $P$ from $\mathcal{C}$ s.t. $w(P) > B$
12:     Remove all paths $Q$ from $\mathcal{C}$ s.t. $P^* \succ Q$
13: return $\mathcal{H} = G(\mathcal{P})$
```

indexes of those Monte-Carlo realizations where P did exist (line 4). Given $\mathcal{P} = \{P_1, \ldots, P_k\}$, the cover sets can be used to estimate $\Pr(\mathcal{P}) \approx |C(P_1) \cup \cdots \cup C(P_k)|/N$ in $O(kN)$ time. This is a substantial improvement over $\Theta(2^k)$ time required for exact computation.

With cover sets, the path selection problem reduces to an instance of a specialized SET COVER problem (hence the name PATH COVERING), where the goal is to choose a set of paths $\mathcal{P}$ such that $|\bigcup_{P \in \mathcal{P}} C(P)|$ is maximized and $||G(\mathcal{P})|| \leq B$, where $||G(\mathcal{P})||$ denotes the number of edges in $G(\mathcal{P})$. This problem differs from the ordinary SET COVER in three ways: it does not require the entire universe (the set of all positive realizations) to be covered, it is weighted (via budget B), and the weights are dynamic (different choices of paths affect the cost of individual paths).

To solve the path selection problem, we use a greedy approach where we add one path at a time to an initially empty $\mathcal{P}$ (lines 5–12). We always choose the best possible addition from $\mathcal{C}$, until the budget B has been exhausted. Here, "best possible" means the one adding most Monte-Carlo realizations to the cover per edge added to $\mathcal{P}$. Formally, the *cost* $w(P)$ of a path P is the number of new edges added to the solution subgraph $G(\mathcal{P})$: $w(P) = ||P \setminus G(\mathcal{P})||$. The *score* $s(P)$ of a path is defined as the ratio of the improvement in probability over its cost:

$$s(P) = \frac{\Pr(\mathcal{P} \cup P) - \Pr(\mathcal{P})}{w(P)}. \qquad (1)$$

Note that both s and w vary in each iteration. With cover sets, score function (1) has an estimate

$$\tilde{s}(P) = \frac{|C(P) \setminus C(\mathcal{P})|}{w(P)} \qquad (2)$$

where $C(\mathcal{P}) = |\bigcup_{P \in \mathcal{P}} C(P)|$.

In an extreme case, the cost of a path becomes zero if all of its edges have already been included in the solution. As an additional optimization, we implement a look-ahead to take advantage of such situations: if the addition of path P would make another path $Q \in \mathcal{C}$ completely included in $G(\mathcal{P})$, then the cover set $C(P)$ is extended with the cover set $C(Q)$. More formally, we say that P *dominates* Q if an inclusion of P into $\mathcal{P}$ implies $Q \in G(\mathcal{P})$, and denote this relation by $P \succ Q$. Clearly, relation $\succ$ is reflexive and transitive. An improved estimate of (2) with dominating paths is thus

$$\tilde{s}(P) = \frac{|\bigcup_{Q \in \mathcal{C} : P \succ Q} C(Q) \setminus C(\mathcal{P})|}{w(P)}. \qquad (3)$$

At each iteration, we choose the path P with the maximum $\tilde{s}(P)$ of (3), and add P into $\mathcal{P}$ (line 6). In (3), we assume that after adding a path P to $\mathcal{P}$, all paths dominated by P are removed from $\mathcal{C}$ (line 12). During successive iterations, the enumerator in (3) approaches zero as the proportion of realizations covered by paths in $\mathcal{P}$ increases.

Eventually all realizations may become covered so that $\tilde{s}(P) = 0$, and the choice of the remaining paths becomes somewhat arbitrary. In these (rare) situations our implementation "restarts" by considering all realizations uncovered (line 8) and tries to recover them with additional paths from $\mathcal{C}$ as before. In every iteration, we also remove paths that are too expensive to be added to $\mathcal{P}$ (line 11).

IV. EVALUATION

In this section, we experimentally evaluate the proposed algorithm. We study the reliability of the extracted subgraphs and the running times, we test different algorithmic variants and alternatives, and we compare PATH COVERING against it closest competitors, BPI and SPA [2].

Our data source is DBLP (http://www.informatik.uni-trier.de/~ley/db/), a computer science bibliography. We converted DBLP to a bipartite graph with

2,076,911 author and publication nodes, linked by 3,293,211 edges. Edge weights were assigned as in previous work [10] with parameters $rq = 0.8$ and $\alpha = 0.05$. This assignment gives larger weights to low-degree nodes. In other words, an article with few authors indicates a stronger relationship between its authors that a paper with many authors. In a similar way, an author with few publications indicates stronger similarity for the publications.

We evaluate PATH COVERING on three different cases from DBLP, each one consisting of a pair of well known scholars. Rakesh Agrawal (177 links to publications in our graph) and Jiawei Han (388 links) work on similar sub-fields of computer science. Heikki Mannila (171 links) and Mark de Berg (157 links) in turn work on different sub-fields. Finally, Donald E. Knuth (99 links) and Edsger W. Dijkstra (67 links) are prominent yet unrelated authors.

The test problems were characterized by the test case, the source graph size and the extracted subgraph size. The source graphs were subsets of DBLP, ranging from 500 to 10,000 nodes, and they were obtained specifically for each of the test cases. Unless otherwise mentioned, the results below are for the graphs of 1,000 nodes and roughly 2,000 edges. The extracted subgraph sizes ranged from $B = 20$ to 250 edges (default value is 80).

In the experiments, PATH COVERING was run with the following parameters. In the first phase, we produced $2 \cdot B$ candidate paths. In the second phase, 10,000 iterations were done. The cut sampling algorithm (Fig. 3) turned out to work well (see below) and was used in all tests. To control random variation, we report averages over 50 independent test runs. All reported reliability values are estimates calculated with crude Monte-Carlo method using 1,000,000 iterations. We report reliabilities and running times for all three cases; for brevity, other results we show are representative samples.

A. Subgraph reliability

From Fig. 5 (a) we can see that a subgraph with 100 edges almost surely connects Han and Agrawal. Interestingly, there are only 12 author nodes between them in the extracted subgraphs on average, and a total of 47 distinct in-between authors over all 50 independent runs. Fig. 1 is a simplified excerpt of one such subgraph with 15 author nodes, where nodes are connected if the corresponding authors have coauthored at least one article. On the contrary, Mannila and de Berg, and especially Knuth and Dijkstra, are less strongly connected (Fig. 5 (d) and (g)).

Fig. 5 (a), (d), and (g) show that PC fares consistently better than the previous algorithms BPI and SPA, while being on par in efficiency (Fig. 5 (b), (e), and (h)).

B. Scalability to large input graphs

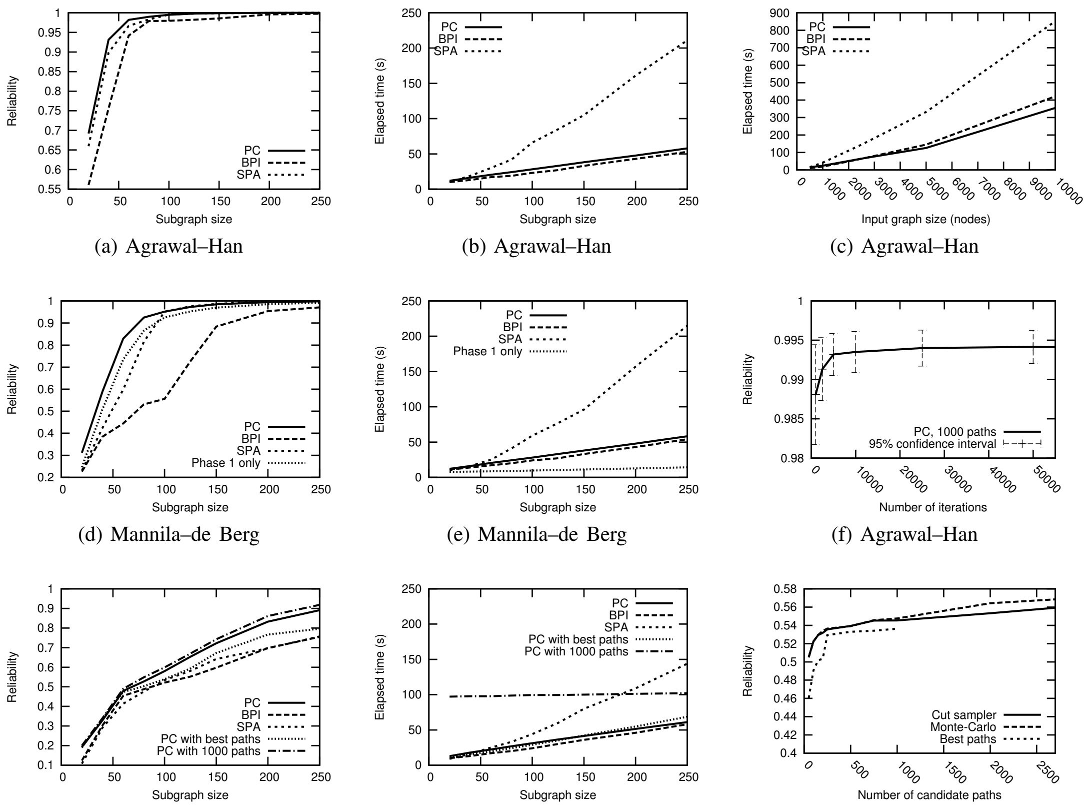
PC scales well to large source graphs and is as efficient as the fastest existing methods (Fig. 5 (c)). Scalability is close to linear, which is expected: the running time of the algorithm is dominated by Monte-Carlo simulation, whose complexity grows linearly with respect to the input graph size and the number of iterations. We emphasize that the reported running times are, for the sake of comparability, from unoptimized implementations. With our optimized implementation, input graphs up to 10,000 nodes can be handled in less than 10 seconds (results not shown).

On very large graphs, however, it is likely that the shortest path algorithm used in the path sampling algorithm (Fig. 2) becomes the dominating factor. Setting a minimum acceptable probability or a maximum path length for the shortest path might be useful in such cases.

C. Algorithmic variants and other alternatives

The cut sampling algorithm (Fig. 3) approximates the basic Monte-Carlo method (Fig. 2). It is efficient under various conditions (results not shown) and the approximation does not have a significant adverse effect on the reliability of the result subgraph (Fig. 5 (i)).

Both phases of PC try to choose an optimal set of paths, but the first phase is constrained by the very large number of possible paths. The additional benefit of using a second phase to fine tune the result is consistent but not huge (Fig. 5 (d)), with a clear



(a) Agrawal–Han (b) Agrawal–Han (c) Agrawal–Han

(d) Mannila–de Berg (e) Mannila–de Berg (f) Agrawal–Han

(g) Knuth–Dijkstra (h) Knuth–Dijkstra (i) Knuth–Dijkstra

Fig. 5. Representative results from the experiments. Corresponding source graphs are given in the figure labels.

additional cost in computation time (Fig. 5 (e)). Apparently, the first phase can be used alone as an approximate but extremely efficient algorithm.

An alternative to the first phase would be to find the k best s–t-paths instead of k sampled paths. Indeed, in some cases using k best paths works equally well (results not shown), but often is inferior (see Fig. 5 (g) and (h) for examples). Additionally, finding a large number of best paths can become intractable, as was the case in Fig. 5 (i), where our implementation ran out of memory after 1,000 paths.

D. Number of candidate paths in phase 1

Both phases of PC have a parameter with which to tune the reliability/time trade-off: for the first phase, the number of paths produced; for the second phase, the number of Monte-Carlo realizations produced. According to our results (Fig. 5 (g) and (h)), producing $2 \cdot B$ paths seems a reasonable compromise between efficiency and quality. Using 1,000 paths instead of the 40–500 paths for subgraphs of 20–250 edges produces only marginally better subgraphs (Fig. 5 (g)) with a significant increase in running time (Fig. 5 (h)). As a function of the number of paths, the reliability levels off quite soon but does not completely stop growing (Fig. 5 (g)).

For the second phase of PC, in our experiments the default number of iterations (10,000) seems to be large enough to produce accurate estimates for path score calculations (Fig. 5 (f)). Smaller numbers of iterations have both smaller reliabilities and larger variances. For larger numbers of iterations, the differences are small.

V. Conclusions

Discovery of indirect links between individuals is a central task in social network analysis. We addressed the problem of extracting a small subgraph that connects two given individuals as strongly as possible. By viewing a social network as a Bernoulli random graph, the problem can be formulated as the most reliable subgraph extraction problem. We proposed a novel method, Path Covering (PC) for solving this problem.

Experiments with bibliography data indicate that PC improves over state of the art in the quality of the results while being on par in scalability to large graphs. For extreme efficiency, the first phase of PC can be used alone to obtain good results very fast. Future experiments include systematic tests to find out robust parameters that perform reliably over wide range of input graphs and query nodes, and extensive comparisons with related algorithms.

There are many possible variants of the approaches described in this paper that could be explored to find better solutions. For instance, the greedy algorithm in the second phase could be replaced with a more elaborate algorithm, such as the branch-and-bound approach used by Koren et al [12]. We have extended the Monte-Carlo framework proposed here to handle multiple query nodes, and the necessary modifications have been sketched by Kasari et al [14].

Acknowledgments

We thank the Biomine team for providing the DBLP data set and associated tools. This work has been supported by the Algorithmic Data Analysis (Algodan) Centre of Excellence of the Academy of Finland (Grant 118653). We also thank the European Commission under the 7th Framework Programme FP7-ICT-2007-C FET-Open, contract BISON-211898.

References

[1] P. Hintsanen, "The most reliable subgraph problem," in *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2007, pp. 471–478.

[2] P. Hintsanen and H. Toivonen, "Finding reliable subgraphs from large probabilistic graphs," *Data Mining and Knowledge Discovery*, vol. 17, pp. 3–23, 2008.

[3] C. J. Colbourn, *The Combinatorics of Network Reliability*. Oxford University Press, 1987.

[4] L. G. Valiant, "The complexity of enumeration and reliability problems," *SIAM Journal on Computing*, vol. 8, pp. 410–421, 1979.

[5] F. T. Boesch, A. Satyanarayana, and C. L. Suffel, "A survey of some network reliability analysis and synthesis results," *Networks*, vol. 54, pp. 99–107, 2009.

[6] S. Kiu and D. F. McAllister, "Reliability optimization of computer-communication networks," *IEEE Transactions on Reliability*, vol. 37, pp. 475–483, 1988.

[7] R. Jan, "Design on reliable networks," *Computers & Operations Research*, vol. 20, pp. 25–34, 1993.

[8] D. P. Kroese, K.-P. Hui, and S. Nariai, "Network reliability optimization via the cross-entropy method," *IEEE Transactions on Reliability*, vol. 56, pp. 275–287, 2007.

[9] C. Faloutsos, K. S. McCurley, and A. Tomkins, "Fast discovery of connection subgraphs," in *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pp. 118–127.

[10] P. Sevon, L. Eronen, P. Hintsanen, K. Kulovesi, and H. Toivonen, "Link discovery in graphs derived from biological databases," in *Proceedings of Data Integration in the Life Sciences, Third International Workshop*, 2006, pp. 35–49.

[11] H. Tong and C. Faloutsos, "Center-piece subgraphs: problem definition and fast solutions," in *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, pp. 404–413.

[12] Y. Koren, S. C. North, and C. Volinsky, "Measuring and extracting proximity graphs in networks," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, pp. 245–255.

[13] L. D. Raedt, K. Kersting, A. Kimmig, K. Revoredo, and H. Toivonen, "Compressing probabilistic Prolog programs," *Machine Learning*, vol. 70, pp. 151–168, 2008.

[14] M. Kasari, H. Toivonen, and P. Hintsanen, "Fast discovery of reliable k-terminal subgraphs," in *Proceedings of the 14th Pacific-Asia conference on knowledge discovery and data mining*, 2010, pp. 168–177.

[15] R. M. Karp, M. Luby, and N. Madras, "Monte-Carlo approximation algorithms for enumeration problems," *Journal of Algorithms*, vol. 10, pp. 429–449, 1989.

[16] G. S. Fishman, "A comparison of four Monte Carlo methods for estimating the probability of s-t connectedness," *IEEE Transactions on Reliability*, vol. R-35, pp. 145–155, 1986.