Soc. Netw. Anal. Min. (2013) 3:953–968
DOI 10.1007/s13278-013-0140-x

ORIGINAL ARTICLE

Pairwise trust inference by subgraph extraction

Yuan Yao · Hanghang Tong · Feng Xu · Jian Lu

Received: 27 October 2012 / Revised: 10 September 2013 / Accepted: 23 September 2013 / Published online: 6 October 2013

Abstract Inferring pairwise trustworthiness is a core building block behind many real applications, e.g., e-commence, p2p networks, mobile ad hoc network, etc. Most of the existing inference algorithms suffer from the scalability and usability issues due to the large scale of the underlying social networks. In this paper, we propose subgraph extraction to address these challenges. The core of the proposed method consists of two stages: *path selection* and *component induction*. The path selection stage is flexible and it admits many of existing top-k path extraction algorithms. We propose two evolutionary algorithms for component induction stage. Our method has two main advantages. First, the outputs of both stages can be used as an intermediate step to speed up a variety of existing trust inference algorithms. Second, it improves the usability of the trust inference result by presenting an intuitive subgraph that concisely summarizes how the trustworthiness score is calculated. The extensive experimental evaluations on real datasets demonstrate the effectiveness and efficiency of the proposed method.

Y. Yao (✉) · F. Xu · J. Lu
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
e-mail: yyao@smail.nju.edu.cn

F. Xu
e-mail: xf@nju.edu.cn

J. Lu
e-mail: lj@nju.edu.cn

Y. Yao · F. Xu · J. Lu
Department of Computer Science and Technology, Nanjing University, Nanjing, China

H. Tong
City College, CUNY, New York, NY, USA
e-mail: tong@cs.ccny.cuny.edu

Keywords Trust inference · Pairwise trustworthiness · Social network · Subgraph extraction · Path selection · Component induction

1 Introduction

Trust is a fundamental concept in computer science, especially in online environment (Paul and Richard 2002; Jøsang et al. 2007; Tang et al. 2012; Yao et al. 2013). In online environment, users usually need to interact with counterparts with whom they have no previous experiences (e.g., sellers in eBay, or reviewers in Epinions); additionally, due to the open nature of such environment, these counterparts could be dishonest or even malicious (Fang et al. 2013). Therefore, it is essential to evaluate the trustworthiness of these counterparts. For example, users in eBay need to decide the trustworthiness of the sellers to guide their purchase; in product review sites such as Epinions, trust could help the users to find high-quality reviews.

The key insight of existing trust evaluation methods is to exploit indirect experiences, i.e., the direct experiences from others. For example, consider the case when *Alice* wants to decide the trustworthiness of a stranger *Carol* who *Alice* has no direct experiences with. If *Alice's* friend *Bob* has direct experiences with *Carol*, then *Alice* may decide her trustworthiness on *Carol* based on the experiences of *Bob*. We call such trust evaluation methods as trust inference, which aims to infer a pairwise trustworthiness score from the trustor to the trustee based on the existing trust relationships in the underlying social network. Typically, trust inference takes the existing trust ratings in the social network as input, and outputs the personalized trustworthiness score to indicate to what degree the trustor should believe in the trustee. In the past decade, trust inference is studied and applied in many real-world applications including e-commerce (Xiong and Liu 2004), peer-to-peer networks (Kamvar et al. 2003), mobile ad hoc networks (Buchegger and Le Boudec 2004), recommender systems (Anand and Bharadwaj 2013), emotion diffusion (Wang et al. 2011), etc.

To date, many trust inference algorithms have been proposed, which can be categorized into two main classes (see Sect. 3 for a review): (a) path-based inference (Mui et al. 2002; Wang and Singh 2006; Liu et al. 2009; Hang et al. 2009; Wang and Wu 2011) and (b) component-based inference (Guha et al. 2004; Massa and Avesani 2005; Ziegler and Lausen 2005; Zhou and Hwang 2007). The basic idea behind the path-based methods is to propagate trust along a set of connected paths from the trustor to the trustee. On the other hand, the component-based methods focus on applying graph theories on a connected component from the trustor to the trustee. The major difference between these two classes is whether there exists explicit concept of paths in the trust inference algorithms.

Despite their own success, most of the existing inference algorithms have two limitations. The first challenge lies in *scalability*—many existing trust inference algorithms become very time-consuming, or even computationally infeasible for the graphs with more than thousands of nodes. Therefore, it is necessary to make these algorithms more efficient because the underlying trust networks are usually of large scale. Additionally, some trust inference algorithms assume the existence of a subgraph while how to construct such a subgraph remains an open issue (Wang and Wu 2011). The second challenge is the *usability* of the inference results—we need to ensure that the inference result is interpretation-friendly and the inference process itself is trustworthy. However, most, if not all, of the existing inference algorithms only output an abstract numerical trustworthiness score. This gives a quantitative measure of *to what extent* the trustor should trust the trustee, but gives few cues on *how* the trustworthiness score is inferred. This usability/interpretation issue becomes more evident when the size of the underlying graph increases, since we cannot even display the entire graph to the end-users (see Fig. 1a for an example).

In this paper, we propose subgraph extraction to address these challenges. Notice that we are not going to propose a new trust inference algorithm, but to extract subgraphs in order to speed up existing trust inference algorithms. The core of our subgraph extraction method consists of two stages: *path selection* and *component induction*. In the first (path selection) stage, we extract a few, important paths from the trustor to the trustee (see Fig. 1b for an example).

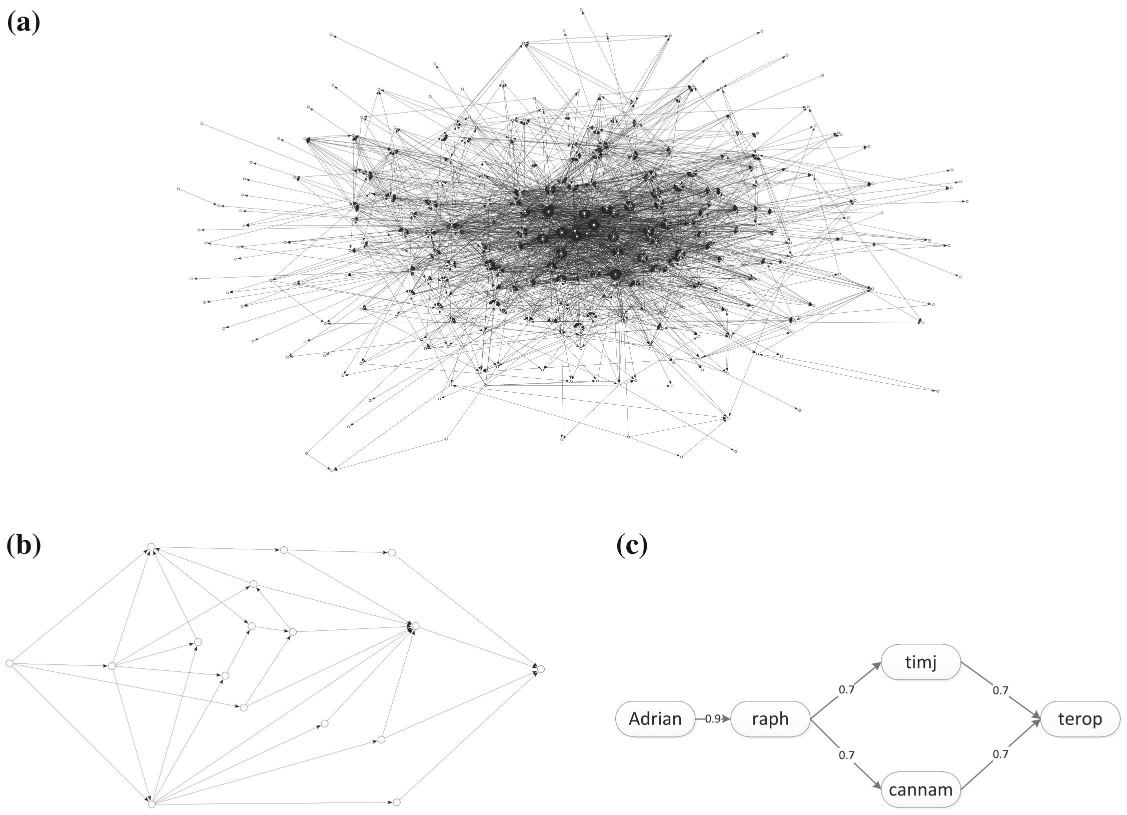

Fig. 1 The motivation example. **a** The original whole graph. **b** The output subgraph of the path selection stage. The paths are from 'Adrian' (the *leftmost node*) to 'terop' (*rightmost node*)

This stage is flexible and it admits many of the existing top-k path extraction algorithms. In the second (component induction) stage, we propose two novel evolutionary algorithms to generate a small subgraph based on the extracted paths (see Fig. 1c for an example). The outputs of these two stages are then used as an intermediate step to speed up the path-based inference and component-based inference algorithms, respectively. Our experimental evaluations on real graphs show that the proposed method can significantly accelerate existing trust inference algorithms (up to 2,400× speed-up), while maintaining high accuracy (P-error is less than 0.04). In addition, the extracted subgraph provides an intuitive way to interpret the resulting trustworthiness score by presenting a concise summarization on the relationship from the trustor to the trustee. We believe that our work can improve most of the existing trust inference algorithms by: (1) scaling up as well as (2) delivering more usable (i.e., interpretation-friendly) inference results to the end-users.

The rest of the paper is organized as follows. Section 2 presents the detailed definition of the subgraph extraction problem for trust inference. Section 3 reviews the related work of subgraph extraction and trust inference. Sections 4 and 5 describe the algorithms for the path selection stage and component induction stage, respectively. Section 6 presents our experimental setup and results. Section 7 concludes the paper.

2 Problem definition

In this section, we formally define our subgraph extraction problem, which consists of two subproblems of path selection and component induction, for pairwise trust inference in social networks.

Following the standard notations in the existing trust inference algorithms, we model the trust relationships in social networks as a weighted directed graph (Barbian 2011; Yao et al. 2011). The nodes of the graph represent the participants in the network, the edges represent trust relationships, and the weight on each edge indicates the local trust value derived from the historical interactions.

We then categorize the existing trust inference algorithms into two major classes: *path-based trust inference* and *component-based trust inference*. Path-based trust inference includes the approaches, which are started by the trustor, to evaluating the trustworthiness of the trustee, through a set of paths from the trustor to the trustee in the network. On the other hand, component-based trust inference includes the approaches, which are started by the trustor, to evaluating the trustworthiness of the trustee, through a connected component from the trustor to the trustee in the network.

Let us first explain the differences between these two classes. In path-based trust inference, trust is propagated along a path, and the propagated trust from multiple paths is combined to form a final trustworthiness score. In contrast, there is no explicit concept of paths in component-based trust inference. Instead, it takes the initial graph as input and treats trust as, for example, random walks on a Markov chain (Richardson et al. 2003).

As to the similarity, both classes belong to the subjective/pairwise trust metrics (Ziegler and Lausen 2005), where different trustors can form different opinions on the same trustee. Accordingly, path-based trust inference such as Mui et al. (2002), Wang and Singh (2006), Liu et al. (2009), Hang et al. (2009), Wang and Wu (2011) and component-based inference such as Guha et al. (2004), Massa and Avesani (2005), Ziegler and Lausen (2005), Zhou and Hwang (2007) all belong to trust inference algorithms. Although the main focus of this paper is on the subjective metrics, our proposed subgraph extraction can also be applied to the objective trust metrics.

Despite the success of most existing inference algorithms, they share the scalability and usability limitations. To address these issues, we propose subgraph extraction for trust inference. The core of our subgraph extraction consists of two stages. The first stage, which serves for path-based trust inference, selects a set of paths from the trustor to the trustee. The second stage aims to produce a connected component between the trustor and the trustee for component-based trust inference. In addition, the second stage of our subgraph extraction produces a relatively small subgraph which can be clearly displayed and help the end-user better understand the inference result.

We now formally define the subgraph extraction problem for trust inference. In accordance with the corresponding two stages, the problem is divided into two subproblems: *path selection problem* and *component induction problem.*

Definition 1 Path selection problem.

Given: a weighted directed graph $G(V, E)$, two nodes $s, t \in V$, and an integer K;

Find: a set C with K paths from s to t that minimizes the error function $f(C)$.

Definition 2 Component induction problem.

Given: a set C of paths from s to t, and an integer N;

Find: an induced component $H(V', E')$ with at most N edges that minimizes the error function $g(H)$, where $V' \subseteq \{v|(u,v) \in P$ or $(v,u) \in P, P \in C\}$ and $E' \subseteq \{e|e \in P, P \in C\}$.

We next discuss the error function in the definitions. Basically, the error function is used as a measure to indicate the subgraph quality. For example, the error function

$f(C)$ in Definition 1 indicates the goodness of the extracted paths, and $f(C)$ reaches its minimum value when C contains all the possible paths from s to t. Similarly, the error function $g(H)$ in Definition 2 reaches its minimum value if $H = G$. In this paper, we use P-error, which is defined as follows, as the error function for both subproblems, i.e., $f = g = P$-error.

Definition 3 *P*-error.

For a given trustor–trustee pair, the error function P-error is defined as

$$P\text{-error} = |p_{\text{sub}} - p_{\text{whole}}|,$$

where p_{sub} is the trustworthiness score inferred from the subgraph, and p_{whole} which serves as a ground truth, is the trustworthiness score inferred from the whole graph.

3 Related work

To provide a better understanding of the proposed subgraph extraction problem, we review the related work in this section, which can be categorized into three parts: trust, trust inference algorithms, and subgraph extraction.

3.1 Trust

Here, we distinguish several similar concepts including trust, reputation, and similarity. We first compare trust with reputation. We can obtain the basic intuition from the following two sentences: "I trust you because of your good reputation", and "I trust you despite your bad reputation" (Jøsang et al. 2007). Actually, trust is known to be subjective, while reputation is objective and usually based on a collective of opinions from the community (Fazeen et al. 2011). This is why we focus on the pairwise trust inference as the unique trustworthiness score is much closer to reputation. Second, the relationship between trust and similarity is also studied: on one hand, there is empirical evidence showing that trust and similarity are positively correlated (Golbeck 2009); on the other hand, similarity can also be used as the input to infer trust (Xiang et al. 2010; Li et al. 2010). Actually, the trust derived from similarity is referred to as social selection, and the similarity based on trust is known as social influence (Tang et al. 2009).

3.2 Trust inference

We roughly categorize existing trust inference algorithms into two main classes: path-based trust inference and component-based trust inference.

In the first class of path-based inference, trust is propagated along a path from the trustor to the trustee, and the propagated trust from multiple paths can be combined to form a final trustworthiness score. For example, Wang and Singh (2006, 2007) as well as Hang et al. (2009, 2013) propose operators to concatenate trust along a path and aggregate trust from multiple paths. They further prove some desiring algebra properties based on their operators. Liu et al. (2009, 2010) argue that not only trust values but social relationships and recommendation role are important for trust inference. They also propose algorithms to combine these dimensions together for more advanced trust inference. However, all these algorithms are only suitable for small networks due to their complexity. Some other path-based trust inference algorithms, such as Mui et al. (2002), Wang and Wu (2011), assume the existence of an extracted subgraph while how to construct such a subgraph remains an open issue (Wang and Wu 2011).

In the second class of component-based inference, EigenTrust (Kamvar et al. 2003) tries to compute an objective trustworthiness score for each node in the graph. In contrast to EigenTrust, our main focus is to provide support for subjective trust metrics where different trustors can form different opinions on the same trustee. In contrast to path-based trust inference algorithms, there is no explicit concept of paths in component-based trust inference. Instead, existing subjective trust algorithms, including Guha et al. (2004), Massa and Avesani (2005), Ziegler and Lausen (2005), Kuter and Golbeck (2007), Nordheimer et al. (2010), take the initial graph as input and treat trust as random walks on a Markov chain or on a graph (Richardson et al. 2003). For example, in MoleTrust (Massa and Avesani 2005) and Appleseed (Ziegler and Lausen 2005), trust propagates along the edges according to the trust values on the edges. Sunny (Kuter and Golbeck 2007) adopts a variant of backward breadth-first search from the trustee to the trustor, and generates a Bayesian Network before inferring trust between them. Nordheimer et al. (2010) propose to use Monte Carlo simulation method to evaluate trust on large graphs. Our subgraph extraction method not only can speed up many of these algorithms, but also can provide interpretive result which is not considered by the existing algorithms.

Overall, our subgraph extraction is motivated to address the two common challenges (i.e., scalability and usability) shared by most of these existing trust inference algorithms.

3.3 Subgraph extraction

Several end-to-end subgraph extraction algorithms are developed to solve different problems.

In the field of graph mining, Faloutsos et al. (2004) refer to the idea of electrical current where trust relationships are modeled as resistors, and try to find a connection subgraph that maximizes the current flowing from source to target. Later, Tong et al. (2007) generalize the connection subgraph to directed graphs and use the subgraph to compute proximities between nodes. Similar to Tong et al., Koren et al. (2006) also try to induce a subgraph for proximity computation. In addition, Koren et al. search the k-shortest paths to provide a basis for measuring the proximity.

since it is proposed for existing trust inference algorithms most of which do not take social context into consideration. Jiang et al. (2012) also propose to extract a subgraph for existing trust inference algorithms. They adapt the breadth-first search method by selecting a subset of neighbors in each step. However, their method still requires context information such as the categories of products.

Algorithm 1 KS algorithm.

```
Require: Weighted directed graph $G(V, E)$, two nodes $s, t \in V$, and a parameter $K$ of path number
Ensure: Set $C$ with $K$ paths from $s$ to $t$
 1: $X \leftarrow$ shortest path from $s$ to $t$
 2: $C \leftarrow$ shortest path from $s$ to $t$
 3: while $|C| < K$ and $X \neq \emptyset$ do
 4:     $P \leftarrow$ remove a shortest path in $X$
 5:     $d \leftarrow$ the deviation node of $P$
 6:     for each node $v$ between $d$ (inclusive) and trustee $t$ (exclusive) in $P$ do
 7:         $pre \leftarrow$ subpath from trustor $s$ to $v$ in $P$
 8:         $post \leftarrow$ the deviated shortest path from $v$ to $t$
 9:         combine $pre$ and $post$, and add it to $X$
10:     end for
11:     $C \leftarrow C$ + a shortest path in $X$
12: end while
13: return $C$
```

In the past few years, several algorithms are proposed for reliable subgraph extraction (Hintsanen 2007). Among them, Monte Carlo pruning (Hintsanen and Toivonen 2008) measures the relevance of each edge by Monte Carlo simulations, and tends to remove the edge of lowest relevance one by one. The most related work is perhaps the randomized Path Covering algorithm (Hintsanen et al. 2010) which also consists of two stages of path sampling and subgraph construction. However, both Monte Carlo pruning and Path Covering tend to find a subgraph with highest probability to be connected, while we aim to find a subgraph to address the scalability and usability issues in trust inference.

More recently, Jin et al. (2011) propose to mine all the highly reliable subgraphs from a probabilistic graph. However, their subgraphs are node-independent while we need to extract subgraphs for specific trustor and trustee. Liu et al. (2012) propose to extract a social context-aware trust network in which social relationships, recommendation role, and preference similarity can all be captured. In contrast, our subgraph extraction method is more general

4 Path selection

In this section, we put our focus on the first stage of path selection, and describe the algorithms used in this stage followed by some effectiveness and efficiency analysis.

4.1 Algorithm description

In the path selection stage, the goal is to extract a few paths from the trustor to the trustee as an intermediate step to speed up path-based trust inference algorithms. These extracted paths will also serve as the input for the component induction stage.

There are two preprocessing steps in our extraction method. First of all, trust is interpreted as the probability by which the trustor expects that the trustee will perform a given action. This interpretation of trust is adopted by many existing trust inference algorithms, and it allows trust to be multiplicatively propagated along a path (Liu et al. 2010). Second, we transform probability into weight by negative logarithm. Namely, the local trust value on the edge e is interpreted as probability $p(e)$, and the probability $p(e)$ is transformed to weight $w(e) = -log(p(e))$. Based on these two steps, the weight of a path P can be presented as

$$w(P) = \sum_{e \in P} -\log(p(e)) = -\log\left(\prod_{e \in P} p(e)\right)$$
$$= -\log(Pr(P)).$$

As a result, finding a path of high trustworthiness in the original network is equivalent to finding a short path in the transformed network. We will use this transformed weighted graph $G(V, E)$ as the input of our method.

Then, the path selection problem becomes to extract top-k short paths from the trustor to the trustee in the transformed graph $G(V, E)$. To this end, many existing algorithms can be plugged in. In this paper, we consider two representative algorithms from the literature: *Yen's k-shortest loopless paths (KS)* algorithm (Yen 1971) and *path sampling* (*PS*) algorithm (Hintsanen et al. 2010).

The first KS algorithm is shown in Algorithm 1. In the algorithm, we use Dijkstra's algorithm for finding a shortest path. All the computed paths are loopless by temporarily removing visited nodes. The key idea of the KS algorithm is *deviation*. The *deviation node d* of path P is the node that makes P deviate from existing paths in the candidate set C. For each node v between d (inclusive) and trustee t (exclusive) in P, the *deviated shortest path* from node v to t is computed by temporarily removing the edge starting at v in P. The computed deviated shortest path *post* and the subpath *pre* (the path from s to v in P) are combined to form a possible path candidate. For the nodes before d, possible shortest paths are already computed and included in X. Based on deviation, KS finds the K shortest paths from trustor s to trustee t one by one. Following Martins and Pascoal's implementation (Martins and Pascoal 2003), we compute the deviated shortest path from deviation node d to the trustee in a reverse order.

Algorithm 2 PS algorithm.

```
Require: Weighted directed graph $G(V, E)$, two nodes $s, t \in V$, and a parameter $K$ of path number
Ensure: Set $C$ with $K$ paths from $s$ to $t$
1: $C \leftarrow$ shortest path from $s$ to $t$
2: while $|C| < K$ do
3:     re-decide all the edges in $E$
4:     for each path $P$ in $C$ do
5:         if $P$ is decided as *true* then
6:             $F \leftarrow F + P$
7:         end if
8:     end for
9:     while $F \neq \emptyset$ do
10:        re-decide the most overlapped edge in $F$ as failed
11:        remove failed paths from $F$, if there are any
12:    end while
13:    $P \leftarrow$ shortest path among the non-failed edges from $s$ to $t$
14:    if $P \neq \emptyset$ then
15:        $C \leftarrow C + P$
16:    end if
17: end while
18: return $C$
```

The other algorithm is the randomized algorithm PS, which is proposed for the *most reliable subgraph problem* (Hintsanen and Toivonen 2008). While PS is proposed for undirected graphs, trust relationships in social networks should be directed as trust is asymmetric in nature (Golbeck and Hendler 2006). Therefore, we adapt PS (as shown in Algorithm 2) for a directed graph.

PS considers the input graph as a Bernoulli random graph (Robins et al. 2007), and the algorithm is based on the *edge decision* of this random graph. An edge is randomly decided as true with probability $p(e)$, and a path is decided as true if all the edges on the path are decided as true. At the beginning of each iteration, all the edges of the graph are re-decided, and these *graph decisions* provide opportunities for neutral or distrust information to be contained. Like KS, PS first adds a shortest path into candidate set C. PS then tries to find a graph decision based on which none of the paths in C is true. To avoid the situation when this graph decision is hardly found, PS stores the true paths in C to a temporary set F, and deliberately fails the most overlapping edges in F until none of the paths in F is true. Finally, based on the results of graph decision and edge failing, PS finds a shortest path P among the non-failed edges from trustor s to trustee t, and adds it to C. The algorithm ends until K paths are found.

4.2 Algorithm analysis

Based on our interpretation and transformation of trust, KS can be viewed as an optimistic algorithm because it always tries to find the most trustworthy paths from the trustor to the trustee and ignores the controversial opinions from others. Compared to KS, PS is more neutral since it allows some controversial opinions to be incorporated into the extracted subgraph, which could in turn lower the P-error based on our experiments. In addition, as indicated in our experiments, we found that both KS and PS perform well even if the multiplicative property of the interpretation does not hold.

However, the time complexity of PS is difficult to estimate, since the wall-clock time depends on the graph density. For example, if the graph is sparse which is the usual case in reality, finding a shortest path among the non-failed edges (Step 13 in Algorithm 2) could be very time-consuming because no path might exist after many edges have been failed in the previous steps. In addition, as shown in our experiments, the wall-clock time of PS is especially long when K becomes sufficiently large. As to KS, the worst-case time complexity is $O(K|V|(|E| + |V| \log |V|))$, which is known as the best result to ensure that k-shortest loopless paths can be found in a directed graph (Hershberger et al. 2007). The actual wall-clock time of KS on many real graphs is often much better than such worst-case scenario (Martins and Pascoal 2003). In fact, based on our experiments, we find that it empirically scales near linearly wrt the graph size $|V|$ in the chosen datasets.

5 Component induction

In this section, we first describe the two proposed algorithms for the component induction stage, and then analyze the effectiveness and efficiency of the proposed algorithms.

5.1 Algorithm description

In the component induction, we take the output of path selection stage (i.e., a set of K paths) as input, and output a small connected component from the trustor to the trustee. The output of the component induction stage not only acts as an intermediate step to speed up component-based trust inference algorithms, but also helps to improve the usability of trust inference by interpreting the inference results for the end-users.

Here, we propose two algorithms (i.e., *EVO* and *PathEVO*) which belong to the so-called evolutionary methods (Bäck 1996). As we can see in Algorithms 3 and 4, both EVO and PathEVO aim to minimize P-error under the constraint of edge number. There are two implicit parameters in both algorithms, i.e., the initial vector number m and iteration number *iter*.

We first explain EVO (as shown in Algorithm 3) in detail. There are three major steps in EVO:

1. The first step is to generate the directly induced component $G^c(V^c, E^c)$ from the set C of paths from trustor s to trustee t, where $V^c = \{v|(u,v) \in P$ or $(v,u) \in P, P \in C\}$ and $E^c = \{e|e \in P, P \in C\}$. We will refer to this step as *direct induction* in the following.
2. After direct induction, the second step of EVO is to establish a one-to-one correspondence between the edges in G^c and the elements in vector B. Each element of B is a 0/1 bit where 1 indicates that the corresponding edge exists and 0 indicates otherwise. The vector has exactly $|E^c|$ bits where $|E^c|$ is the edge number of G^c.
3. In the third step, the algorithm generates m vectors $B_1, B_2, \ldots, B_m$, and each of them has at most N 1-bits. In our implementation, we apply a constant-time search in C to find a subset of paths with minimized P-error.
4. Next, EVO adopts *mutation* on each of the vectors generated by the previous step and separately generates m new vectors $B_{m+1}, B_{m+2}, \ldots, B_{2m}$. In the mutation from B_i to B_{i+m}, each bit of B_i is changed with probability $1/|E^c|$. If the resulting vector has more than N 1-bits, the mutation operation is redone.

Algorithm 3 EVO algorithm.

```
Require: Set $C$ of $K$ paths from $s$ to $t$, as well as a constraint $N$ of the edge number
Ensure: Induced component $H(V', E')$ with at most $N$ edges
 1: directly induce the component $G^c(V^c, E^c)$ from the set $C$ of paths from $s$ to $t$
 2: define 0/1 vector $B$ of size $|E^c|$ where each element in $B$ stands for the existence of a corresponding
    edge in $G^c$
 3: initialize $m$ vectors $S \leftarrow \{B_1, B_2, ..., B_m\}$, with at most $N$ 1-bits for each vector
 4: while $iter > 0$ do
 5:     for each vector $B_i$ in $S$ do
 6:         repeat
 7:             $mutate$ $B_i$ to $B_{i+m}$ with mutation probability $1/|E^c|$
 8:         until the number of 1-bits in $B_{i+m} \leqslant N$
 9:     end for
10:     compute P-error results for the $2m$ vectors $\{B_1, B_2, ..., B_{2m}\}$
11:     $S \leftarrow$ the best $m$ vectors from the $2m$ ones
12:     $iter \leftarrow iter$ - 1
13: end while
14: $B_{final} \leftarrow$ the best vector in $S$
15: return the corresponding component $H(V', E')$ of $B_{final}$
```

Algorithm 4 PathEVO algorithm.

```
Require: Set $C$ of $K$ paths from $s$ to $t$, as well as a constraint $N$ of the edge number
Ensure: Induced component $H(V', E')$ with at most $N$ edges
 1: define 0/1 vector $B$ of size $K$ where each element in $B$ stands for the existence of a corresponding path
    in $C$
 2: initialize $m$ vectors $S \leftarrow \{B_1, B_2, ..., B_m\}$, each of which contains at most $N$ edges in the directly
    induced component
 3: while $iter > 0$ do
 4:     for each vector $B_i$ in $S$ do
 5:         repeat
 6:             $mutate$ $B_i$ to $B_{i+m}$ with mutation probability $1/K$
 7:         until the edge number of the directly induced component of $B_{i+m} \leqslant N$
 8:     end for
 9:     compute P-error results for the $2m$ vectors $\{B_1, B_2, ..., B_{2m}\}$
10:     $S \leftarrow$ the best $m$ vectors from the $2m$ ones
11:     $iter \leftarrow iter$ - 1
12: end while
13: $B_{final} \leftarrow$ the best vector in $S$
14: return the directly induced component $H(V', E')$ of $B_{final}$
```

5. Finally, the error function, which is P-error in our case, is computed on each of the $2m$ vectors from the previous two steps. We keep the m vectors with smallest P-error and then go back to the third step. For efficiency, the P-error computation on vector B herein means computing the P-error between $G^c(V^c, E^c)$ and the component corresponding to the vector B. Namely, we use the input component $G^c(V^c, E^c)$ as an approximation of the ground truth in this stage.

PathEVO (as shown in Algorithm 4) is similar to EVO except for the following points. First, PathEVO establishes the one-to-one correspondence between the paths in C and the elements in vector B, while EVO establishes a one-to-one correspondence between the edges in G^c and the elements in vector B. In other words, PathEVO evolves on the path level while EVO evolves on the edge level. Second, instead of inducing component G^c at the first step, PathEVO makes the direct induction when a subset of paths are chosen (e.g., in Step 7, Step 9, and Step 14 in Algorithm 4), and the edge number is counted on the directly induced component. Third, we change the mutation probability of PathEVO to $1/K$ since there are only K elements in vector B now. By doing so, we can keep the expected number of element change in B as 1. For other things, PathEVO is quite similar to EVO. For example, PathEVO also uses the component $G^c(V^c, E^c)$ as an approximation of the ground truth.

5.2 Algorithm analysis

As mentioned above, both EVO and PathEVO belong to the evolutionary method. One of the advantages of such methods is that they are general purpose algorithms, i.e., they can be applied to a large variety of problems. For a specific problem, these algorithms only need to know how to represent and evaluate the solution. This advantage exactly matches our situation, as in our problem we represent the solution as a component and evaluate it with P-error, and we know little about the complexity and structure of the problem. Therefore, EVO and PathEVO can converge to their local optima. In terms of applicability, EVO outperforms PathEVO because PathEVO can only be used when the input is a set of paths. Notice that although the proposed EVO algorithm could also be applied on the whole graph, we do not recommend it in practice for the following two reasons: (1) most trustworthy paths have already been captured by the path selection stage (i.e., KS, etc); and (2) applying EVO on the whole graph would cost more memory and time to achieve high accuracy. We will present more detailed experimental evaluations to validate this in the next section.

The time complexity of EVO and PathEVO is summarized in the following lemmas, which basically say that the expected time complexity of EVO and PathEVO scales linearly wrt both initial vector number m and iteration number *iter*.

Lemma 1 *The average-case time complexity of EVO is* $O(iter \cdot m(|E^c|/N + \theta))$, *where* θ *is the time complexity of the error function computation.*

Proof In the mutation step of EVO, with mutation probability $1/|E^c|$, the expected number of bit changes is 1. This step is expected to be redone only when the number of 1-bits is N and the bit change is from 0 to 1. Under this condition, the probability of bit change from 0 to 1 is $(|E^c| - N)/|E^c|$. Therefore, the expected iteration number of the mutation step in EVO is $|E^c|/N$. Therefore, the whole expected time complexity of EVO is $O(iter(m \cdot |E^c|/N + m\theta)) = O(iter \cdot m(|E^c|/N + \theta))$, which completes the proof. □

Lemma 2 *The average-case time complexity of PathEVO is* $O(iter \cdot m(K + \theta))$ *when there exists at least one solution for PathEVO, where* θ *is the time complexity of the error function computation.*

Proof In the mutation step of PathEVO, with mutation probability $1/K$, the expected number of bit changes in vector B is 1. This step is expected to be redone when the bit change is from 0 to 1 and the resulting directed component has more than N edges. Let us assume there are i paths in B when this condition happens. Consequently, the redone probability under this condition is at most $(K - i)/K$ which is also the probability of bit change from 0 to 1. There are two cases now. Case 1: if $i > 0$, the expected iteration number of the mutation step is K/i which is at most K; Case 2: if $i = 0$, the iteration number is also at most K since we assume there exists at least one solution for PathEVO. Therefore, the expected iteration number of the mutation step is at most K. The whole expected time complexity of PathEVO is $O(iter(mK + m\theta)) = O(iter \cdot m(K + \theta))$, which completes the proof. □

6 Experiments

In this section, we first describe the experimental setup, and then present experimental evaluations. All the experiments are designed to answer the following questions:

- *Effectiveness* How accurate are the existing trust inference algorithms if they are applied on our extracted subgraphs?
- *Efficiency* How much is the computational savings by applying the proposed subgraph extraction method? How does our method scale?
- *Interpretation* Is the extracted subgraph clear and interpretable?

Table 1 High-level statistics of Advogato datasets

Graph	Nodes	Edges	Avg. degree	Avg. clustering (Watts and Strogatz 1998)	Avg. diameter (Leskovec et al. 2005)	Date
Advogato-1	279	2,109	15.1	0.45	4.62	2000-02-05
Advogato-2	1,261	12,176	19.3	0.36	4.71	2000-07-18
Advogato-3	2,443	22,486	18.4	0.31	4.67	2001-03-06
Advogato-4	3,279	32,743	20.0	0.33	4.74	2002-01-14
Advogato-5	4,158	41,308	19.9	0.33	4.83	2003-03-04
Advogato-6	5,428	51,493	19.0	0.31	4.82	2011-06-23

6.1 Experimental setup

Before presenting the experimental results, we first describe the datasets and the representatives of path-based and component-based trust inference algorithms. All algorithms are implemented in Java, and have been run on a T400 ThinkPad with 1280m jvm heap space. Few other activities are done during the experiments.

6.1.1 Datasets description

We use the Advogato[1] datasets in our experiments. Advogato is one of the earliest social networking websites for open-source software developers. One interesting feature about Advogato is that the website provides a robust, attack-resistant trust metric. Namely, it provides multilevel trust assertions to allow users to certify each other in a kind of peer review process, and such assertions can be used to avoid the abuses that plague open community sites. There are four levels of trust assertions in the network, i.e., ‘Observer’, ‘Apprentice’, ‘Journeyer’, and ‘Master’. These assertions can be mapped into real numbers in [0, 1]. In our experiments, we map ‘Observer’, ‘Apprentice’, ‘Journeyer’, and ‘Master’ to 0.1, 0.4, 0.7, and 0.9, respectively. The statistics of the datasets is shown in Table 1.

Advogato is one of the earliest social networking websites for open-source software developers. It describes itself as “the free software developer’s advocate”. One interesting thing about Adovato is that the website provides a robust, attack-resistant trust metric. Users in the website can certify each other in a kind of peer review process and this information can be used to avoid the abuses that plague open community sites. In summary, it has been the basis of numerous research papers on trust metrics and social networking.

6.1.2 Trust inference representatives

To evaluate our subgraph extraction method, we need to apply trust inference algorithms on the whole graph and on our extracted subgraph to compare their effectiveness and efficiency. We chose *CertProp* (Hang et al. 2009) as the representative of path-based inference algorithms, and *Appleseed* (Ziegler and Lausen 2005) as the representative of component-based inference algorithms.

P-error computation in CertProp needs to first compute the ground truth p_{whole} by finding all paths from the trustor to the trustee in the whole graph. This computation, however, easily causes the overflow of the jvm heap space even on the Advogato-1 graph. Following the suggestions in the original CertProp (Hang et al. 2009), we apply the fixed search strategy and search all paths whose length is not longer than seven as an approximation of the ground truth. For CertProp, we define *collapsed samples* as the trustor–trustee pairs of which the *P*-error computation either exceeds the range of Java.lang.Double or runs out of the jvm heap space. We randomly select 100 node pairs out of 122 samples, where the rest 22 of them are collapsed samples. Our experimental results are all based on the average of these 100 samples. Notice that, as discussed in the path selection section, the multiplicative property of the probability interpretation does not hold in CertProp. As to Appleseed, we apply linear normalization on the outputs,

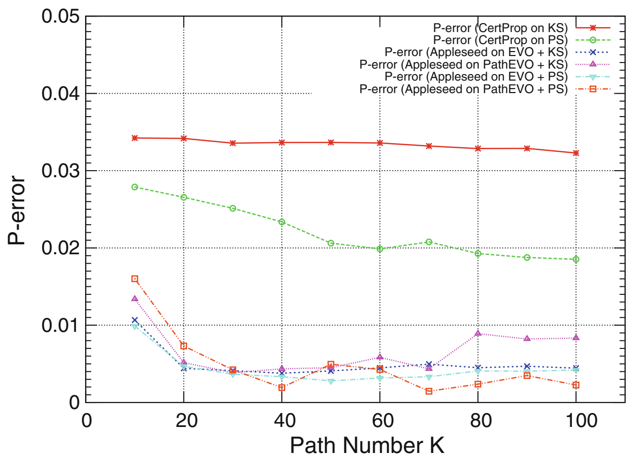

Fig. 2 Effectiveness of our subgraph extraction method with edge number constraint $N = K/2$. In all cases, the *P*-error is less than 0.04

[1] http://www.trustlet.org/wiki/Advogato_dataset.

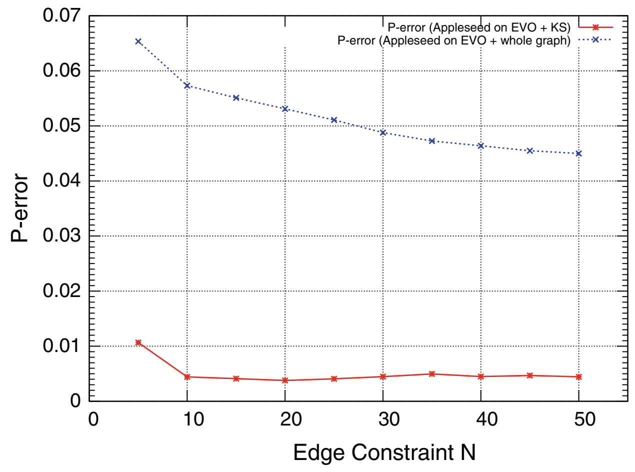

Fig. 3 Comparison of EVO on KS vs. EVO on the whole graph with edge number constraint $N = K/2$. EVO on KS outperforms EVO on the whole graph

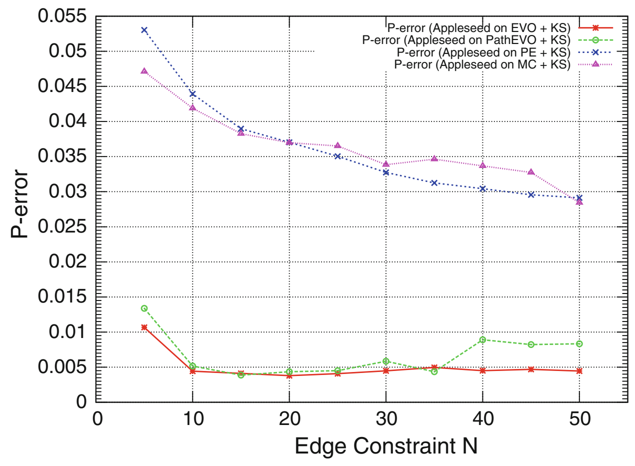

Fig. 4 Comparison of different component induction algorithms with edge number constraint $N = K/2$. EVO outperforms the existing component induction algorithms

since the algorithm can produce arbitrary trustworthiness scores.

6.2 Experimental results

We now present the experimental results of our subgraph extraction method. In our experiments, most of the effectiveness, efficiency comparisons, and interpretation results are all based on the Advogato-1 graph, as we found CertProp on the whole graph becomes computationally infeasible on all the other larger datasets. We evaluate the scalability of our method using all the datasets (i.e., Advogato-1 to Advogato-6). As for EVO, we set $m = 5$ and $iter = 10$ unless otherwise specified. The edge constraint N is set as $K/2$.

6.2.1 Effectiveness

For effectiveness, we first study how CertProp performs on KS and PS subgraph (the output of path selection stage), and how Appleseed performs on the EVO and PathEVO subgraph (the output of component induction stage), respectively. Considering we have two algorithms in the first stage and two algorithms in the second stage, there are four kinds of output in the second stage (i.e., 'EVO+KS', 'PathEVO+KS', 'EVO+PS', 'PathEVO+PS').

The results are shown in Fig. 2. We can first observe that all the *P*-error values of CertProp and Appleseed are less than 0.04, indicating that our extracted subgraphs, which are based on a small set of carefully selected paths and the evolutionary strategy, provide high accuracy for the trust inference algorithms. Second, we find that PS can provide better subgraphs wrt *P*-error for CertProp, which indicates that allowing some controversial information to be incorporated into the extracted subgraphs can increase the accuracy of trust inference. However, KS runs much faster than PS on average (as shown later in Fig. 5), and we therefore recommend KS in this stage. We conjecture that PS can be used in dense graphs where numerous paths exist between node pairs. Finally, the performance of Appleseed on the four output combinations in the second stage is close to each other. However, considering the fact that EVO is more general than PathEVO (e.g., PathEVO cannot be applied on the whole graph where there is no explicit set of paths), and the *P*-error results are comparable with each other, we will put our focus on EVO in the following.

Remember that the proposed EVO is always applied on the output of the path selection stage (referred to as 'EVO+KS'). Here, for comparison purpose, we also apply EVO on the entire graph (referred to as 'EVO+whole graph'). With the same parameter setting, the results are shown in Fig. 3. It can be seen that EVO on KS outperforms EVO on the whole graph. The reason is as follows. As an evolutionary algorithm, EVO (either on KS or on the entire graph) finds a local minimum. By restricting the search space to those highly trustworthy paths (i.e., the output of KS), it converges to a better local minimum in terms of *P*-error.

Finally, to compare EVO with existing component induction algorithms, we implement the *Monte Carlo pruning (MC)* method (Hintsanen and Toivonen 2008) and the *proximity extraction (PE)* method (Koren et al. 2006), and plot the results in Fig. 4. Again, we can see that EVO outperforms both MC and PE wrt *P*-error. In fact, MC induces a component by successively deleting edges (edge-level component induction), while PE only selects a smaller set of paths (path-level component induction). Our EVO algorithm combines these two levels of component induction by searching a smaller set of paths in the initial

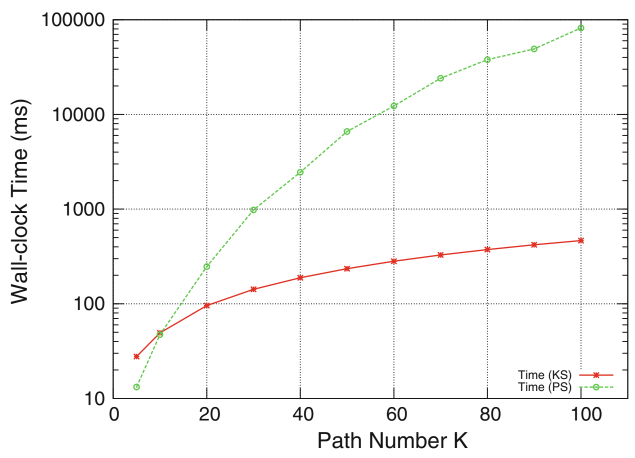

Fig. 5 The average wall-clock time of KS and PS. The average wall-clock time of KS is much faster than that of PS when K is greater than 30

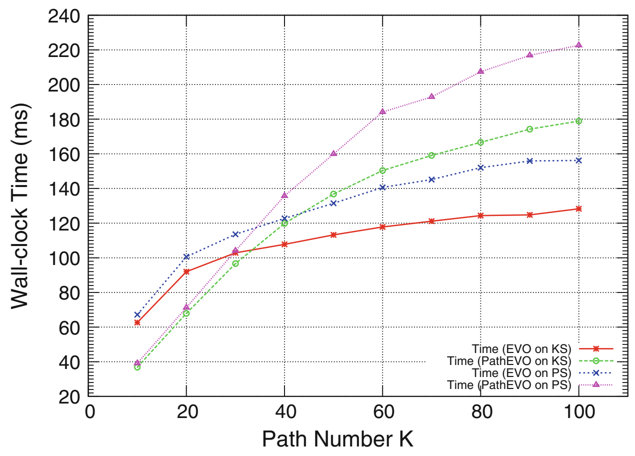

Fig. 6 The average wall-clock time of EVO and PathEVO. Both of them scale near linearly wrt K

step and then evolving the resulting component on the edge level. In the same figure, we also plot the result of applying PathEVO on KS (referred to as 'PathEVO+KS') for comparison. As we can see, the performance is very close, and 'EVO+KS' outperforms 'PathEVO+KS' a little on average.

6.2.2 *Efficiency*

For efficiency, we first compare the different algorithmic choices in the path selection stage. Namely, we compare the wall-clock time of KS with PS. The results are shown in Fig. 5. Note that the y-axis is of log scale. As we can see from the figure, although PS is slightly faster than KS when $K = 5$, the wall-clock time of PS is much longer than that of KS when K is greater than 30. For example, the wall-clock time of PS is more than $170\times$ longer than that of KS when $K = 100$. This is consistent with our algorithm analysis in the path selection section. Therefore, we recommend using KS for path selection.

Second, we compare the wall-clock time of the two algorithms in the component induction stage. Here, we only plot the wall-clock time of EVO and PathEVO (i.e., 'EVO on KS', 'PathEVO on KS', 'EVO on PS', 'PathEVO on PS'). The results are shown in Fig. 6. As we can see, both EVO and PathEVO scale near linearly wrt the path number K. In addition, the wall-clock time of EVO grows slower than PathEVO as K increases.

Next, we study the computational savings by applying the proposed subgraph extraction as the intermediate steps for the existing trust inference algorithms. To this end, we report the wall-clock time of CertProp on the output of the path selection stage (KS subgraph), and Appleseed on the output of the component induction stage ('EVO+KS' subgraph), respectively. As mentioned above, the CertProp method may run out of the jvm heap space even on the *Advogato-1* graph. Therefore, to apply CertProp on the *Advogato-6* graph, we shrink the searching depth to four hops, i.e., we only search the paths whose length is not

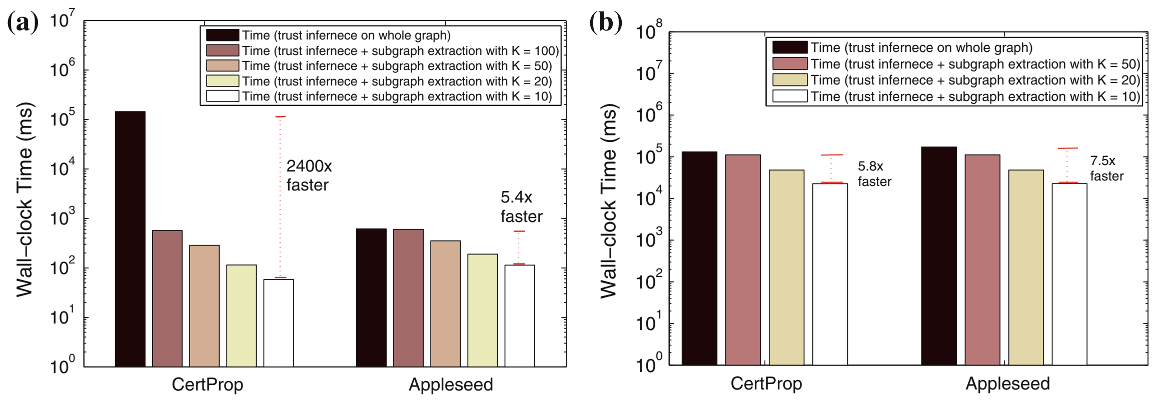

Fig. 7 The average wall-clock time of CertProp on KS and Appleseed on KS+EVO. We achieve up to 2,400× speed-up. **a** The first stage. **b** The second stage

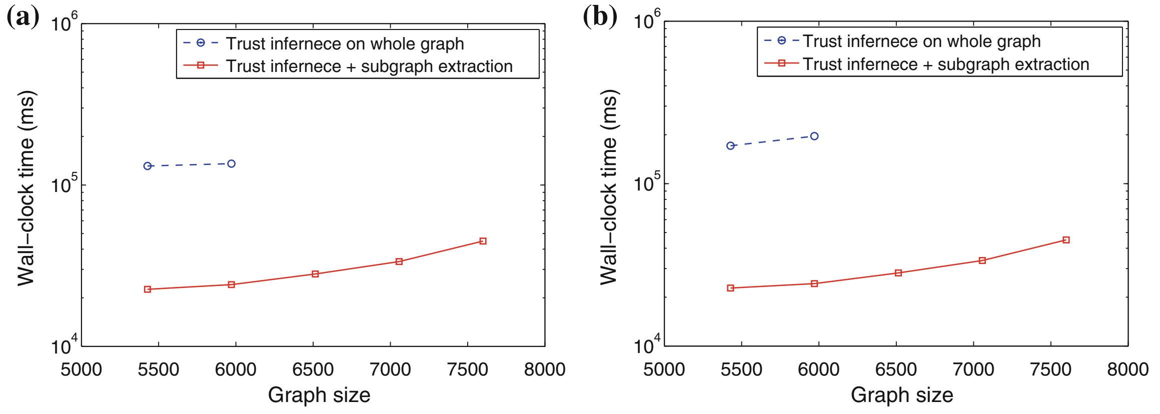

Fig. 8 The speed-up of applying our methods when we randomly added nodes/links on the *Advogato-6* graph. **a** The first stage. **b** The second stage

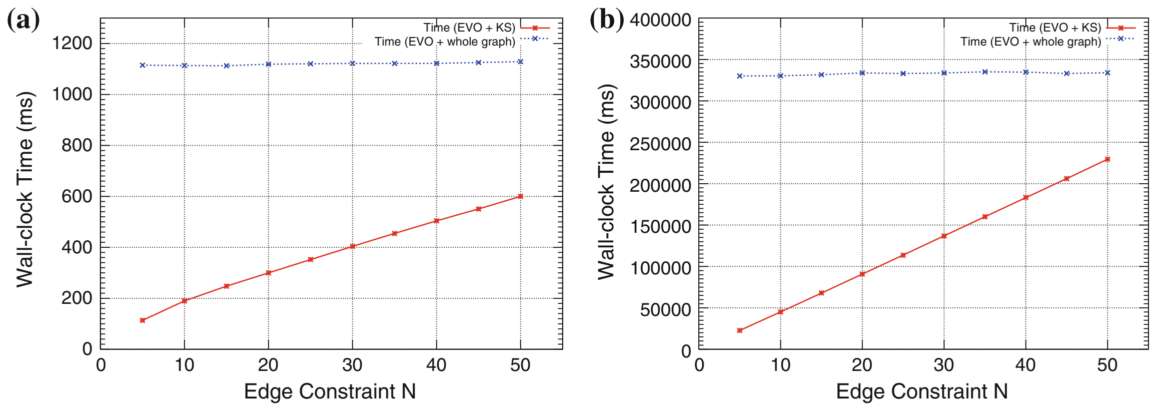

Fig. 9 The average wall-clock time of EVO on KS and EVO on the whole graph with edge number constraint $N = K/2$. EVO on KS is much faster. **a** Advogato-1. **b** Advogato-6

longer than four. The results on both *Advogato-1* and the largest *Advogato-6* are shown in Fig. 7 where the *y*-axis is of log scale. Notice that the reported time includes the wall-clock time of both subgraph extraction and trust inference. In the figures, we also plot the wall-clock time of CertProp and Appleseed on the entire graph for comparison. We can see from Fig. 7a that our subgraph extraction method saves the wall-clock time for both path-based trust inference and component-based trust inference on *Advogato-1*, especially for the former one. For example, when

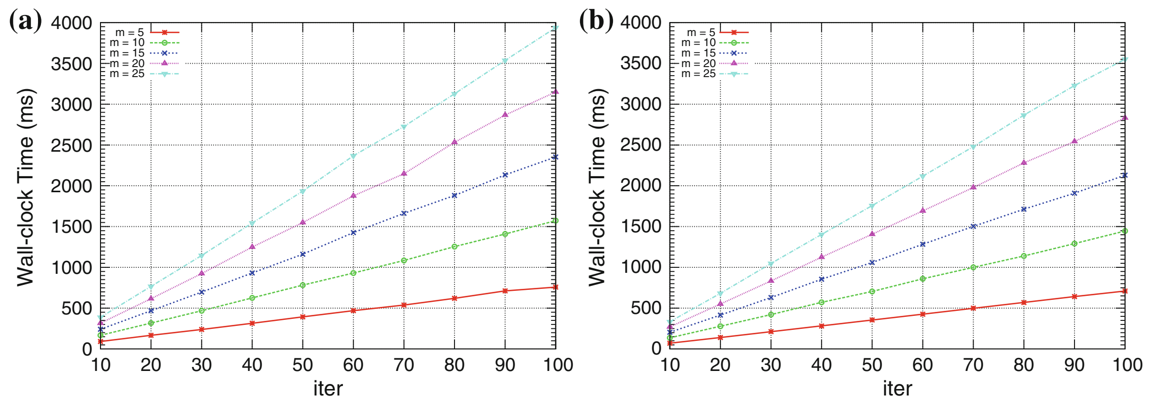

Fig. 10 The average wall-clock time of EVO and PathEVO with $K = 20$ and $N = 10$. Both EVO and PathEVO scale linearly wrt *iter* for the fixed *m*. **a** EVO, **b** PathEVO

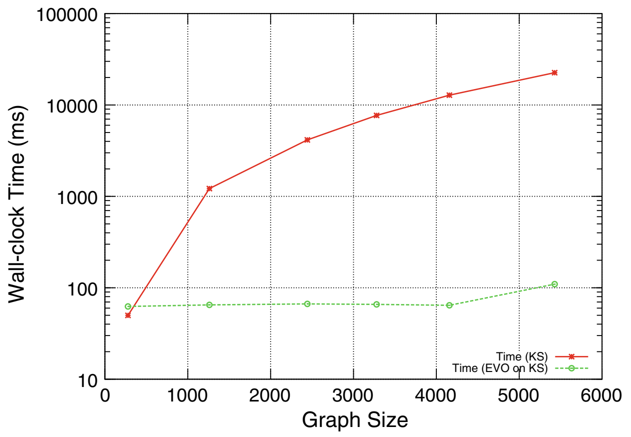

Fig. 11 The scalability of our subgraph extraction method. KS scales near linearly wrt the graph size, while the wall-clock time of EVO stays almost constant

$K = 10$, our subgraph extraction method achieves up to 2,400× and 5.4× speed-up for CertProp and Appleseed, respectively. Even when K grows to more than 50, our method can still achieve 200–400× speed-up for CertProp. As for the largest *Advogato-6* graph, our method can still achieve up to 5.8× and 7.5× speed-up for CertProp and Appleseed, respectively.

To further show the efficiency of our methods on larger graphs, we randomly added nodes into the *Advogato-6* graph. The links are also randomly added according to the density of the graph. We still shrink the searching depth of CertProp to five hops, and show the computational savings of applying the proposed subgraph extraction methods in Fig. 8. Both CertProp and Appleseed will run out of the memory when there are more than 6,000 nodes. Overall, our subgraph extraction method saves the wall-clock time for both path-based trust inference and component-based trust inference.

Next, we compare the efficiency between applying EVO on KS and applying EVO on the whole graph. With $N = K/2$, the results are shown in Fig. 9. As we can see from the figures, on both the *Advogato-1* graph and the *Advogato-6* graph, the wall-clock time of EVO on KS (which includes the wall-clock time of both EVO and KS) is much faster than EVO on the whole graph. Together with the effectiveness results (Fig. 3), we recommend running EVO on the KS subgraph in practice.

Finally, we evaluate how the parameters m and *iter* in EVO and PathEVO affect the wall-clock time. In this experiment, we fix $K = 20$ and $N = 10$, and the results are shown in Fig. 10. We can observe that the wall-clock time of both EVO and pathEVO scales linearly wrt *iter* for any fixed m, which is consistent with the time complexity analysis shown before.

6.2.3 *Scalability*

We now evaluate the scalability of our method (KS and EVO) on datasets Advogato-1 to Advogato-6. Figure 11 shows the results, where the y-axis is of log scale. In this experiment, we fix $K = 10$ and $N = 5$.

We can observe from the figure that even on the largest graph of 5,428 nodes and 51,293 edges, KS can help to infer the trustworthiness score within 25 s. In addition, KS

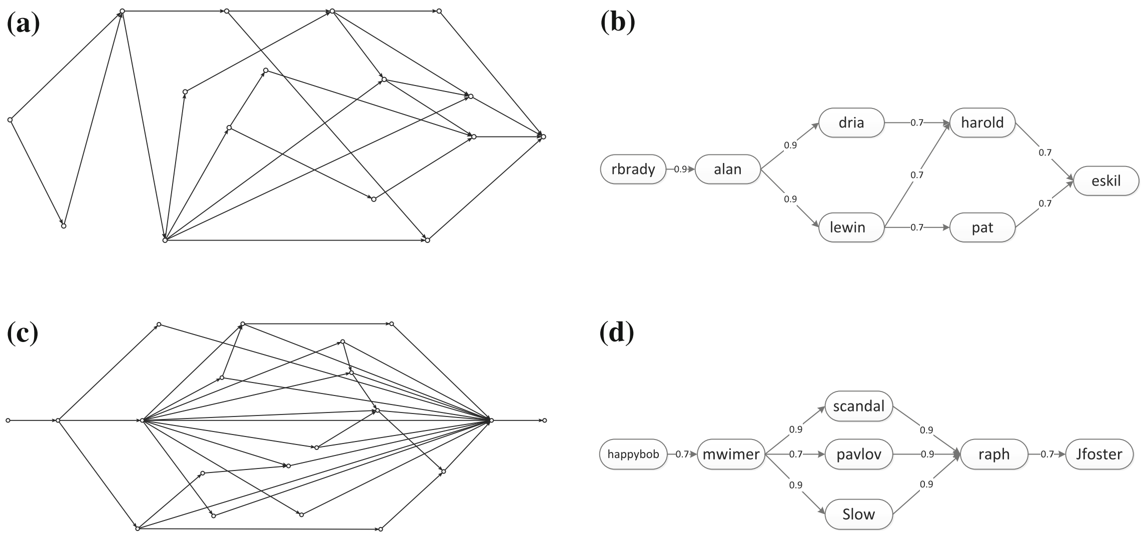

Fig. 12 The interpretation example of the whole graph, KS-20, and EVO-10 on KS-20. **a** KS subgraph with $K = 20$. The paths are from ‘rbrady’ (the *leftmost* node) to ‘eskil’ (the *rightmost* node). **b** EVO subgraph with $N = 10$ on KS-20. The component is from ‘rbrady’ to ‘eskil’. **c** KS subgraph with $K = 20$. The paths are from ‘happybob’ (the *leftmost* node) to ‘Jfoster’ (the *rightmost* node). **d** EVO subgraph with $N = 10$ on KS-20. The component is from ‘happybob’ to ‘Jfoster’

scales near linearly wrt the underlying graph size. As to EVO, the wall-clock time stays stable in spite of the growth of the graph size. The reason is that $|E^c|$ scales near linearly to K due to many overlapping edges, and N is set to $K/2$. Consequently, $|E^c|/N$ is close to a constant, and the time complexity of EVO can be approximated to $O(iter \cdot m \cdot \theta)$.

6.2.4 *Usability/interpretation*

Another important goal of the proposed EVO is to improve the usability in trust inference by interpreting the inferred trustworthiness score for end-users. Two illustrative examples are shown in Fig. 12. The whole graph can be seen in Fig. 1a, and the induced KS subgraph by the path selection stage are also plotted for comparison.

From Fig. 1a we can see that the whole graph is hard for interpretation. As to the KS subgraphs in Fig. 12, although the number of edges has significantly decreased compared with the original whole graph, there are still some redundant edges which might diverge end-users' attention. On the other hand, the EVO subgraphs only present the most important participants and their trust opinions, providing a much clearer explanation on how the trustworthiness score is inferred.

7 Conclusions

In this paper, we have proposed subgraph extraction to address the scalability and usability challenges of existing trust inference algorithms. The core of our subgraph extraction has two stages, and the outputs of both stages can be used as an intermediate step to speed up a variety of existing trust inference algorithms. Our experimental evaluations on real graphs show that the proposed method can significantly accelerate existing trust inference algorithms (up to 2,400× speed-up), while maintaining high accuracy (*P*-error is less than 0.04). In addition, the extracted subgraph provides an intuitive way to interpret the inferred trustworthiness score for end-users. Future work includes incorporating distrust in the subgraph extraction.

Acknowledgments We would like to thank Chao Qian for his useful comments. This work is supported by the National Natural Science Foundation of China (Nos. 61021062, 61073030), and the National 973 Program of China (No. 2009CB320702). The second author was partly sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053.

References

Anand D, Bharadwaj KK (2013) Pruning trust–distrust network via reliability and risk estimates for quality recommendations. Soc Netw Anal Min 3(1):65–84

Bäck T (1996) Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press, USA

Barbian G (2011) Assessing trust by disclosure in online social networks. In: Proceedings of the international conference on advances in social networks analysis and mining, pp 163–170

Buchegger S, Le Boudec JY (2004) A robust reputation system for mobile ad-hoc networks. Tech. rep., KTH Royal Institute of Technology, Theoretical Computer Science Group

Faloutsos C, McCurley KS, Tomkins (2004) A fast discovery of connection subgraphs. In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining, pp 118–127

Fang H, Zhang J, Thalmann NM (2013) A trust model stemmed from the diffusion theory for opinion evaluation. In: Proceedings of the 2013 international conference on autonomous agents and multi-agent systems, pp 805–812

Fazeen M, Dantu R, Guturu P (2011) Identification of leaders, lurkers, associates and spammers in a social network: context-dependent and context-independent approaches. Soc Netw Anal Min 1(3):241–254

Golbeck J (2009) Trust and nuanced profile similarity in online social networks. ACM Trans Web 3(4):12

Golbeck J, Hendler J (2006) Inferring binary trust relationships in Web-based social networks. ACM Trans Internet Technol 6:497–529

Guha R, Kumar R, Raghavan P, Tomkins A (2004) Propagation of trust and distrust. In: Proceedings of the 13th international conference on World Wide Web, pp 403–412

Hang CW, Wang Y, Singh MP. (2009) Operators for propagating trust and their evaluation in social networks. In: Proceedings of The 8th international conference on autonomous agents and multi-agent systems, vol 2, pp 1025–1032

Hang C, Zhang Z, Singh M (2013) Generalized trust propagation with limited evidence. IEEE Comput 46(3):78–85

Hershberger J, Maxel M, Suri S (2007) Finding the k shortest simple paths: a new algorithm and its implementation. ACM Trans Algorithms 3(4):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

Hintsanen P, Toivonen H (2008) Finding reliable subgraphs from large probabilistic graphs. Data Min Knowl Disc 17(1):3–23

Hintsanen P, Toivonen H, Sevon P (2010) Fast discovery of reliable subnetworks. In: Proceedings of the international conference on advances in social networks analysis and mining, pp 104–111

Jiang W, Wang G, Wu J (2012) Generating trusted graphs for trust evaluation in online social networks. Future Gener Comput Syst (online version)

Jin R, Liu L, Aggarwal C (2011) Discovering highly reliable subgraphs in uncertain graphs. In: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining, pp 992–1000

Jøsang A, Ismail R, Boyd C (2007) A survey of trust and reputation systems for online service provision. Decis Support Syst 43(2):618–644

Kamvar SD, Schlosser MT, Garcia-Molina H (2003) The Eigentrust algorithm for reputation management in p2p networks. In: Proceedings of the 12th international conference on World Wide Web, pp 640–651

Koren Y, North S, Volinsky C (2006) Measuring and extracting proximity in networks. In: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining, pp 245–255

Kuter U, Golbeck J (2007) Sunny: a new algorithm for trust inference in social networks using probabilistic confidence models. In:

Proceedings of the AAAI conference on artificial intelligence, pp 1377–1382

Leskovec J, Kleinberg J, Faloutsos C (2005) Graphs over time: densification laws, shrinking diameters and possible explanations. In: Proceedings of the 11th ACM SIGKDD international conference on knowledge discovery and data mining, pp 177–187

Li J, Zhang Z, Zhang W (2010) Mobitrust: trust management system in mobile social computing. In: Proceedings of the IEEE 10th international conference on computer and information technology, pp 954–959

Liu G, Wang Y, Orgun M (2009) Trust inference in complex trust-oriented social networks. In: Proceedings of the international conference on computational science and engineering, pp 996–1001

Liu G, Wang Y, Orgun M (2010) Optimal social trust path selection in complex social networks. In: Proceedings of the twenty-fourth AAAI conference on artificial intelligence, pp 1391–1398

Liu G, Wang Y, Orgun M (2012) Social context-aware trust network discovery in complex contextual social networks. In: Proceedings of the twenty-sixth AAAI conference on artificial intelligence, pp 101–107

Martins E, Pascoal M (2003) A new implementation of Yens ranking loopless paths algorithm.. 4OR Q J Oper Res 1(2):121–133

Massa P, Avesani P (2005) Controversial users demand local trust metrics: an experimental study on epinions.com community. In: Proceedings of the AAAI conference on artificial intelligence, pp 121–126

Mui L, Mohtashemi M, Halberstadt A (2002) A computational model of trust and reputation. In: Proceedings of the 35th annual Hawaii international conference on system sciences, pp 2431–2439

Nordheimer K, Schulze T, Veit D (2010) Trustworthiness in networks: a simulation approach for approximating local trust and distrust values. In: Proceedings of the 4th IFIP WG 11.11 international conference on trust management, pp 157–171

Paul R, Richard Z (2002) Trust among strangers in Internet transactions: empirical analysis of eBay's reputation system.. In: The economics of the Internet and E-Commerce. Advances in Applied Microeconomics: a Research Annual, vol 11, pp 127–157

Richardson M, Agrawal R, Domingos P (2003) Trust management for the semantic Web. In: The Semantic Web, vol 2870, pp 351–368

Robins G, Pattison P, Kalish Y, Lusher D (2007) An introduction to exponential random graph (p*) models for social networks. Soc Netw 29(2):173–191

Tang J, Sun J, Wang C, Yang Z (2009) Social influence analysis in large-scale networks. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining, pp 807–816

Tang J, Gao H, Liu H, Das Sarma A (2012) etrust: understanding trust evolution in an online world. In: Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining, pp 253–261

Tong H, Faloutsos C, Koren Y (2007) Fast direction-aware proximity for graph mining. In: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining, pp 747–756

Wang Y, Singh MP (2006) Trust representation and aggregation in a distributed agent system. In: Proceedings of the AAAI conference on artificial intelligence, pp 1425–1430

Wang Y, Singh MP (2007) Formal trust model for multiagent systems. In: Proceedings of the 20th international joint conference on artificial intelligence, pp 1551–1556

Wang G, Wu J (2011) Multi-dimensional evidence-based trust management with multi-trusted paths. Future Gener Comput Syst 27(5):529–538

Wang D, Sutcliffe A, Zeng XJ (2011) A trust-based multi-ego social network model to investigate emotion diffusion. Soc Netw Anal Min 1(4):287–299

Watts D, Strogatz S (1998) Collective dynamics of 'small-world' networks. Nature 393(6684):440–442

Xiang R, Neville J, Rogati M (2010) Modeling relationship strength in online social networks. In: Proceedings of the 19th international conference on World Wide Web, pp 981–990

Xiong L, Liu L (2004) Peertrust: supporting reputation-based trust for peer-to-peer electronic communities. IEEE Trans Knowl Data Eng 16(7):843–857

Yao Y, Zhou J, Han L, Xu F, Lü J (2011) Comparing linkage graph and activity graph of online social networks. In: Proceedings of the 3rd international conference on social informatics, vol 6984, pp 84–97

Yao Y, Tong H, Yan X, Xu F, Lu J (2013) MATRI: a multi-aspect and transitive trust inference model. In: Proceedings of the 22nd international conference on World Wide Web, pp 1467–1476

Yen J (1971) Finding the k shortest loopless paths in a network. Manag Sci 17(11):712–716

Zhou R, Hwang K (2007) Powertrust: a robust and scalable reputation system for trusted peer-to-peer computing. IEEE Trans Parallel Distrib Syst 18(4):460–473

Ziegler C, Lausen G (2005) Propagation models for trust and distrust in social networks. Inf Syst Front 7(4):337–358