DistR: A Distributed Method for the Reachability Query over Large Uncertain Graphs

Yurong Cheng, *Student Member, IEEE,* Ye Yuan, *Member, IEEE,* Lei Chen, *Member, IEEE,* Guoren Wang, Christophe Giraud-Carrier, and Yongjiao Sun

Abstract—Among uncertain graph queries, reachability, i.e., the probability that one vertex is reachable from another, is likely the most fundamental one. Although this problem has been studied within the field of network reliability, solutions are implemented on a single computer and can only handle small graphs. However, as the size of graph applications continually increases, the corresponding graph data can no longer fit within a single computer's memory and must therefore be distributed across several machines. Furthermore, the computation of probabilistic reachability queries is #P-complete making it very expensive even on small graphs. In this paper, we develop an efficient distributed strategy, called *DistR*, to solve the problem of reachability query over large uncertain graphs. Specifically, we perform the task in two steps: *distributed graph reduction* and *distributed consolidation*. In the *distributed graph reduction* step, we find all of the maximal subgraphs of the original graph, whose reachability probabilities can be calculated in polynomial time, compute them and reduce the graph accordingly. After this step, only a small graph remains. In the *distributed consolidation* step, we transform the problem into a relational join process and provide an approximate answer to the #P-complete reachability query. Extensive experimental studies show that our distributed approach is efficient in terms of both computational and communication costs, and has high accuracy.

Index Terms—Reachability, Distributed Computing, Uncertain Graphs.

1 Introduction

Graphs have become increasingly popular data models in today's big data processing environments, with applications in areas such as social network analysis and pattern discovery in biological networks. In many practical applications, however, there is inherent noise, incompleteness, or delay during data collection [1] [2], so that the corresponding graphs are uncertain, i.e., the presence of edges is probabilistic. Over the past decade, researchers have developed a number of techniques to query and mine uncertain graphs (e.g., see [3] [4] [5]).

One of the most fundamental queries over graphs is reachability, that is, given a start vertex s and a terminal vertex t, can t be reached from s? In uncertain graphs, the reachability query cannot assert whether t can be reached from s; rather, it computes the probability that t can be reached from s. The reachability query over uncertain graphs has important applications in estimating the similarity and influence between vertices. In addition, several other important queries over uncertain graphs, such as *shortest path* [4], *k-nearest neighbor* [6], and *pattern match* [7], are based on the reachability query.

As an example of the use of the reachability query over uncertain graphs, consider the uncertain graph depicted in Fig. 1. Assume that the graph represents a simple social network, such that each vertex is an individual and there is an edge between two vertices if the corresponding individuals are friends. If the probability of each edge is defined as the influence one individual has over the other (e.g., when a_1 recommends a product to b_1, the probability that b_1 also likes the product is 0.7), as in [8], then the reachability query can be used to estimate the influence among individuals in the social network. Alternatively, if the probability of each edge captures the interest similarity between individuals (e.g., the probability that a_1 and b_1 like the same movie is 0.7), as in [9], then the reachability query can be used to estimate the interest similarity among individuals in the social network.

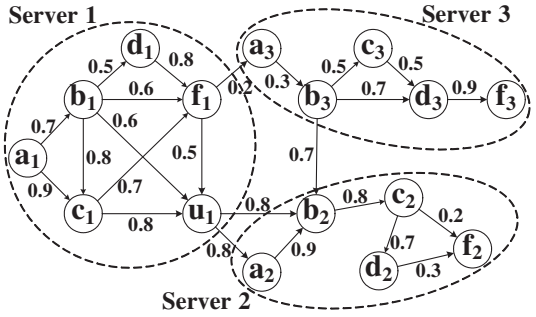


Fig. 1. An Example of a Distributed Uncertain Graph

In addition to exhibiting inherent uncertainty, many interesting graph applications are much too large to be stored on a single computer. For example, popular social networking platforms, such as Twitter, Facebook, WeChat and Sina Weibo, serve huge numbers of users (more than 1 billion in some cases), and their corresponding graph data must be distributed over a number of servers or data centers [10], as illustrated in Fig. 1. Several

- *Yurong Cheng, Ye Yuan, Guoren Wang and Yongjiao Sun are with the College of Information Science and Engineering, Northeastern University, China.*
  E-mail: {yrcheng@stumail, yuanye@ise, wanggr@mail, sunyongjiao@ise}.neu.edu.cn
- *Lei Chen is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology.*
  E-mail:leichen@cse.ust.hk
- *Christophe Giraud-Carrier is with the Department of Computer Science, Brigham Young University, USA.*
  E-mail:cgc@cs.byu.edu

distributed systems, such as Pregel [11] and GraphLab [12], have been designed explicitly to deal with large graphs in distributed environments.

Although the problem of reachability over uncertain graphs has been widely studied in the field of network reliability [13] [5], existing methods assume that the graph is stored on a single server. Moreover, as the reachability probability problem is #P-complete [13] [5], computational costs can be prohibitive even on small graphs. For large graphs, these methods are impractical both in terms of storage requirement and computational cost. Thus, it is desirable to develop efficient and effective solutions to the reachability probability problem for distributed graphs. We propose one such solution here.

Our approach is based on the following key insight: even though computing the reachability probability over the complete graph is #P-complete, computing the reachability of a large portion of the graph can be done in polynomial time. We proceed by decomposing the larger problem into tractable subproblems and combining their partial solutions to produce an estimate of the final solution. In so doing, we make a number of contributions.

1) We develop a *distributed graph reduction* method that finds all subgraphs of the original graph whose reachability probability can be computed in polynomial time.
2) We show that these polynomial-time-computable subgraphs form a special class of graphs known as *maximal series-parallel subgraphs* of the original graph.
3) We design an efficient algorithm, known as *Dist-2hop*, that reduces each maximal series-parallel subgraph to a single edge labeled by its reachability probability, in a distributed environment.
4) We show that the final reachability probability can be obtained via a simple *table join process* based on the reduced graph.

This paper is a significant extension of one of our recent short papers [14], where we provided only a brief introduction, problem definition and basic algorithmic framework. Here, we add a number of examples to the background and motivation, describe the theoretical foundation for and implementation details of our algorithm, and report the results of extensive experiments.

The remainder of the paper is organized as follows. We begin with a formal definition of the reachability probability problem in Section 2. We then describe the details of our *distributed graph reduction and consolidation* framework in Section 3. Section 4 presents a thorough evaluation of the performance of our algorithm over a range of graph applications. Finally, Section 5 reviews some of the most relevant related work, and Section 6 concludes the paper.

2 Problem Statement

Since it has been shown previously that any directed graph can be converted into a directed acyclic graph (DAG) by merging the graph's strongly connected components into single vertices [15], we assume, without loss of generality, that the graphs we operate on here are (simple) DAGs.

Recall that a DAG is a pair $G = (V, E)$, where V is a set of vertices and $E \subseteq V \times V$ is a set of edges among the vertices of V, such that G contains no cycles. We denote by $|V|$ and $|E|$ the number of vertices, respectively edges, of G.

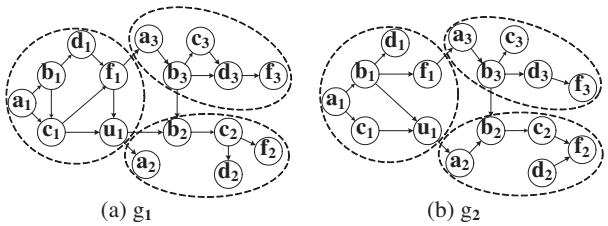
Definition 2.1 (Uncertain Graph). An uncertain graph with respect to a DAG $g_c = (V, E)$ is a pair $\mathcal{G} = (g_c, Pr)$, where Pr is a probability mass function on E. All edges are assumed to be independent of each other.

Definition 2.2 (Possible World Graph). A possible world graph $g = (V', E')$ of an uncertain graph $\mathcal{G} = (g_c, Pr)$ is a subgraph of g_c, such that $V' = V$, $E' \subseteq E$, and the edges in E' are sampled from E according to Pr. The existence probability $pr(g)$ of g, is thus given by:

$$pr(g) = \prod_{e \in E'} pr(e) \prod_{e \in E \setminus E'} (1 - pr(e)) \quad (1)$$

where $pr(e)$ is the existence probability of edge e, as per E's probability mass function Pr, and $E \setminus E'$ is the relative complement of E' in E, i.e., the set of edges from E that do not appear in g.

Note that Definition 2.2 is a generalization of the traditional notion of subgraph. When the graph $\mathcal{G}$ is certain, or deterministic, each edge in E has probability 1, and all other possible edges are missing or have probability 0. In this case, one can view all subgraphs of $\mathcal{G}$ as being constructed by randomly selecting edges from E to produce E', i.e., each edge in E has probability $\frac{1}{2}$ of appearing in E'. Hence, assuming that edges are selected independently of each other, the probability of any subgraph being thus generated, or its existence probability, is $(\frac{1}{2})^{|E|}$, which is identical to Equation 1, since $E = E' \cup (E \setminus E')$. There are $2^{|E|}$ subgraphs of $\mathcal{G}$, and as expected, the sum of the existence probabilities of all of these subgraphs is $(\frac{1}{2})^{|E|} \times 2^{|E|} = 1$. When graphs are uncertain, each edge has its own existence probability, as given by the probability mass function Pr. It follows that the space of subgraphs, here referred to as possible world graphs, no longer follows a uniform distribution. Equation 1 specifies the probability of each possible world graph in that space, based on the probability of each edge being selected (i.e., appearing in E') or not (i.e., belonging to $E \setminus E'$). As with deterministic graphs, the total existence probability of possible world graphs of an uncertain graph sums to 1.



(a) g_1 (b) g_2

Fig. 2. Two of the Possible World Graphs of Fig. 1

As an example, consider Fig. 2 that shows two of the 2^{24} possible world graphs of Fig. 1. The possible world graph g_1 shows the case in which $e_{a_1b_1}$ has been selected (probability $pr(e_{a_1b_1})$), $e_{a_1c_1}$ has been selected (probability $pr(e_{a_1c_1})$), ..., $e_{b_1f_1}$ has not been selected (probability $1 - pr(e_{b_1f_1})$), $e_{a_2b_2}$ has not been selected (probability $1 - pr(e_{a_2b_2})$), etc. As specified by Equation 1, the existence probability of g_1 is given by:

$$\begin{aligned} pr(g_1) &= pr(e_{a_1b_1}) \times pr(e_{a_1c_1}) \times \cdots \times (1 - pr(e_{b_1f_1})) \times \cdots \\ &= 0.7 \times 0.9 \times \cdots \times (1 - 0.6) \times \cdots \\ &= 3 \times 10^{-7} \end{aligned}$$

Definition 2.2, in turn, makes it clear how to define probability calculations over uncertain graphs, such as shortest path probability [4], k-nearest neighbor probability [6], pattern matching probability [7], or reachability probability (Definition 2.3 below), by summing over the existence probabilities of relevant possible world graphs.

Definition 2.3 (Reachability Query over Uncertain Graphs). Given an uncertain graph $\mathcal{G} = (g_c, Pr)$, a start vertex $s \in V$, and a terminal vertex $t \in V$, a reachability query $R(s,t)$ returns the probability that s can reach t in $\mathcal{G}$. This probability, denoted $RPr(s,t)$, is called the reachability probability, and is computed as follows.

$$RPr(s,t) = \sum_i pr(g_i) \tag{2}$$

where the g_i's are all of the possible world graphs of $\mathcal{G}$ in which s can reach t.

As an example, consider again the uncertain graph of Fig. 1. Assume that the start vertex is a_1 and terminal vertex is f_2. Then, the reachability probability $RPr(a_1, f_2)$ is the sum of the existence probabilities of all of the possible world graphs in which a_1 can reach f_2, such as the graphs g_1 and g_2 of Fig. 2. That is,

$$RPr(a_1, f_2) = pr(g_1) + pr(g_2) + \cdots$$

Definition 2.4 (Distributed Uncertain Graphs). An uncertain graph is distributed over a number of servers using a master-slave framework [16]. The graph is partitioned by vertices. The slaves store the partitioned graph; the master is used for scheduling and light calculations. Edges whose vertices are on different servers are called *cross edges*, and edges whose vertices are on the same server are called *inner edges*. A cross edge is represented as $e((v_1, ser_1), (v_2, ser_2), pr_e)$, where ser_1 is the server holding v_1 and ser_2 is the server holding v_2, and stored in both servers.

Note that the partition method used to distribute uncertain graphs has direct influence on the efficiency and effectiveness of any designed algorithm. If the partition is not well balanced, i.e., one server contains a large part of the original graph while the others together contain a small part of the original graph, then the server containing the large part of the original graph would become a bottleneck for any distributed algorithm. On the other hand, if many of the high degree vertices and their neighbors are stored in different servers, then there would be many cross edges in the distributed environment, causing significant communication overhead.

Problem Statement: Given an uncertain graph $\mathcal{G} = (g_c, Pr)$ distributed over a master-slave system as per Definition 2.4, a start vertex $s \in V$, and a terminal vertex $t \in V$, efficiently compute $RPr(s,t)$.

Note that graph partitioning is not part of our problem statement. That is, we consider that the graph of interest has been distributed to different slaves prior to our computation. This is consistent with what can be expected in practice, since in the type of large applications we are considering, the graph data is already organized in a distributed environment. For example, a Facebook user's information is stored in the server located near the place where he/she registers [10].

3 THE *DistR* METHOD

As stated above, our approach is inspired by the realization that although computing the reachability probability over a complete graph is #P-complete, the reachability probability on some parts of the graph can be calculated in polynomial time. This is illustrated in Fig. 3.

Consider the uncertain graph g_c of Fig. 3(a) (edge existence probabilities are not shown for simplicity), and assume that the reachability query is $R(s,t)$. Clearly, g_c can be decomposed into two subgraphs: one containing the vertices s, a, b, c, and d, together with the corresponding edges, call it g_c^1 (depicted in Fig. 3(b)), and the other containing the vertices d, e, f, and t, together with the corresponding edges. Subsequently, $R(s,t)$ may be decomposed into

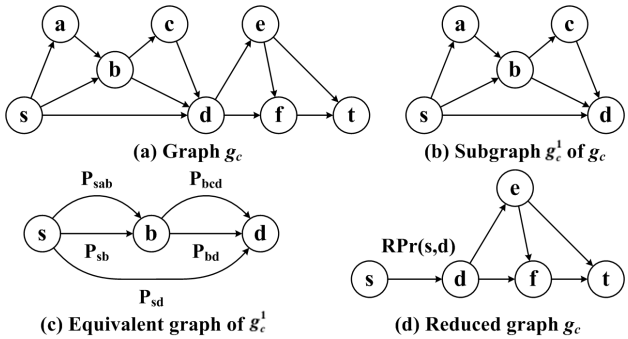


Fig. 3. An Uncertain Graph and its Equivalent Reduction

two reachability queries on each of these subgraphs, namely $R(s,d)$ and $R(d,t)$.

Let $P_{v_1 v_2 \ldots v_k}$ denote the path from vertex v_1 to vertex v_k through vertices $v_2, \ldots, v_{k-1}$. For example, P_{sab} in g_1^c corresponds to the path from s to b through a (i.e., the two edges e_{sa} and e_{ab}) in g_1^c. We can transform the graph g_1^c of Fig. 3(b) into the equivalent graph of Fig. 3(c), where some of the edges have been labeled with the corresponding paths in g_1^c (details of this transformation are in Section 3.1). It follows, and we will generalize this result shortly, that $R(s,d)$ can be solved in polynomial time, and the corresponding reachability probability is given straightforwardly by:

$$RPr(s,d) = pr[(P_{sab} \vee P_{sb}) \wedge (P_{bcd} \vee P_{bd}) \vee P_{sd}] \tag{3}$$

where:

$$\begin{aligned} pr(P_{v_1 v_2 \ldots v_k}) &= pr(e_{v_1 v_2} \wedge \ldots \wedge e_{v_{k-1} v_k}), \\ pr(P \wedge Q) &= pr(P) \times pr(Q), \text{and} \\ pr(P \vee Q) &= 1 - [1 - pr(P)] \times [1 - pr(Q)] \end{aligned}$$

so that,

$$\begin{aligned} RPr(s,d) = 1 - [1 - [1 - (1 - pr(e_{sa}) \times pr(e_{ab}) \times (1 - pr(e_{sb}))] \\ \times [1 - (1 - pr(e_{bc}) \times pr(e_{cd}) \times (1 - pr(e_{bd}))]] \\ \times [1 - pr(e_{sd})] \end{aligned}$$

Following this efficient computation, g_c^1 can be replaced by a single edge e_{sd} labeled with the value of $RPr(s,d)$, and the graph g_c subsequently reduced to the graph shown in Fig. 3(d).

While one may be tempted to apply the same idea to the other subgraph of g_c, such a reduction cannot be performed in polynomial time. We must enumerate all of the paths from d to t to calculate $RPr(d,t)$. As we will show in Section 3.1, the structures whose RPr's can be calculated in polynomial time are known as *series-parallel subgraphs*. If we can find all such maximal series-parallel subgraphs in a distributed way, compute their exact RPr's, and replace them by single edges, then we only need to approximate the RPr of the smaller reduced graph. We contend that this approach avoids unnecessary enumerations, speeds up the calculation, and improves the accuracy of the results.

The proposed DistR strategy thus consists of first reducing all maximal series-parallel subgraphs of the original graph in a distributed fashion, and then consolidating the results via a process of graph sampling and table join over the transformed version of the graph to get a final approximation of $RPr(s,t)$. These steps are detailed in the following subsections.

Note that DistR computes $RPr(s,t)$, the reachability probability between a start vertex s and a terminal vertex t. It is clear that no part of the graph $\mathcal{G}$ linked to from t has any relevance to $RPr(s,t)$. Hence, as is typical in related work on DAGs (e.g., see [17]), the part of the graph "following" t, which can be found efficiently through vertex ordering, is ignored by DistR.

3.1 Distributed Graph Reduction

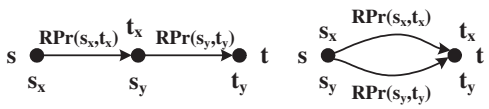
The first step of the DistR algorithm is the heart of the process, where all subgraphs whose reachability probabilities can be computed in polynomial time are reduced to single edges. To show how this is done, we make use of the notion of *series-parallel graph*, and prove that the subgraphs whose reachability probabilities can be calculated in polynomial time are the maximal series-parallel subgraphs of the original graph. Recall that we restrict our attention to DAGs.

Definition 3.1 (Series-Parallel Graph [18]). A series-parallel graph G is defined recursively as follows.

- *Basis*: The graph consisting of two vertices s and t, with an edge from s to t, is a series-parallel graph. The nodes s and t are known as *terminal* nodes.
- *Series Operation*: If G_x is a series-parallel graph with terminal nodes s_x and t_x, and G_y is a series-parallel graph with terminal nodes s_y and t_y, then the graph G formed by setting $s = s_x$, $t_x = s_y$ and $t = t_y$ is a series-parallel graph with terminals s and t.
- *Parallel Operation*: If G_x is a series-parallel graph with terminal nodes s_x and t_x, and G_y is a series-parallel graph with terminal nodes s_y and t_y, then the graph G formed by setting $s = s_x = s_y$ and $t = t_x = t_y$ is a series-parallel graph with terminals s and t.

Definition 3.2 (Maximal Series-Parallel Subgraph). A series-parallel subgraph is maximal if it is not a subgraph of any other series-parallel subgraph.

Fig. 4 provides a visual representation of the series and parallel operations used to construct series-parallel graphs.



(a) Series Operation (b) Parallel Operation

Fig. 4. Series and Parallel Operations

The reachability from s to t after a series operation follows naturally from the reachability of s_x to t_x *and* the reachability from s_y to t_y. Its probability is given by:

$$RPr(s,t) = RPr(s_x,t_x) \times RPr(s_y,t_y) \quad (4)$$

As with series operations, the reachability from s to t after a parallel operation follows naturally from the reachability of s_x to t_x *or* the reachability from s_y to t_y. Its probability is given by:

$$RPr(s,t) = 1 - (1 - RPr(s_x,t_x)) \times (1 - RPr(s_y,t_y)) \quad (5)$$

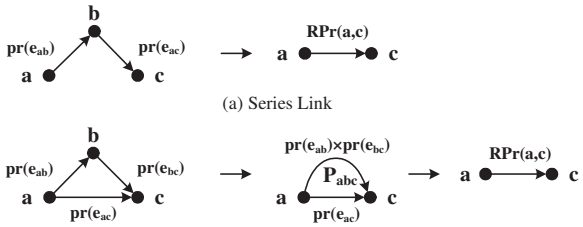
Clearly, Definition 3.1, together with Equations 4 and 5, also provide the basis for what we now define as the *reduction* of an uncertain series-parallel graph.

Definition 3.3 (Series Link). Edges $e_{v_iv_j}$ and $e_{v_jv_k}$ form a series link if both the in-degree and out-degree of v_j are 1.

Definition 3.4 (Triangle Link). Edges $e_{v_iv_j}$, $e_{v_jv_k}$ and $e_{v_iv_k}$ form a triangle link if both the in-degree and out-degree of v_j are 1.

Note that a triangle link can be thought of as consisting of a series link ($e_{v_iv_j}$ and $e_{v_jv_k}$) combined with a single edge ($e_{v_iv_k}$) via a parallel operation. It will be useful in our implementation, to consider triangle links, rather than parallel links, as minimum units of a series-parallel graph, along with series links. Simple examples of a series link and a triangle link are depicted on the left-hand side of Fig. 5(a) and Fig. 5(b), respectively.

The reduction of an uncertain series-parallel graph, depicted on the right-hand side of Fig. 5(a) and Fig. 5(b), can now be defined recursively as follows.



(a) Series Link

(b) Triangle Link

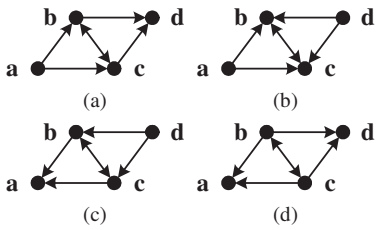
Fig. 5. Series and Triangle Links

Definition 3.5 (Series-parallel Reduction). A series-parallel graph G is reduced recursively as follows.

- *Basis*: A series link consisting of edges $e_{v_iv_j}$ and $e_{v_jv_k}$ is reduced to a single edge $e_{v_iv_k}$ with probability reachability given by Equation 4. A triangle link consisting of edges $e_{v_iv_j}$, $e_{v_jv_k}$ and $e_{v_iv_k}$ is reduced to a single edge $e_{v_iv_k}$ with probability reachability given by Equations 4 and 5.
- *Recursive step*: If G is a series-parallel graph composed of two subgraphs G_x and G_y connected by a series operation (respectively, a parallel operation), then G can be reduced to the reduction of G_x and the reduction of G_y connected by a series operation (respectively, a parallel operation) and its reachability probability is given by Equation 4 (respectively, Equation 5).

Note that although a graph may not be series-parallel, any of its series-parallel subgraphs can be reduced by the above procedure. If a graph is series-parallel, its reduction will yield a single edge with the associated reachability probability. If a graph is not series-parallel, the following lemma characterizes its partial reduction.

Lemma 1 (Contrapositive of Lemma 3.5 [19]). If a graph is not a series-parallel graph, then its partial reduction will yield one of the structures shown in Fig. 6. Note that in the context of DAGs, there are 8 such possible structures. They are captured by bidirectional edges between vertices b and c.



(a) (b) (c) (d)

Fig. 6. The Minimum Units of a Non-series-parallel Graph

The following theorem then provides the theoretical justification for the approach advocated by DistR.

Theorem 1. The subgraphs of an uncertain graph on which reachability queries can be computed in polynomial time are its maximal series-parallel subgraphs.

Proof: We first show that the computation of reachability probability over a series-parallel graph is polynomial. Note that Equation 4 consists of a single multiplication, and Equation 5 involves 3 subtractions and 1 multiplication, thus making each require only constant time. A series-parallel graph $G = (V, E)$ is formed by applying only series or parallel operations as per Definition 3.1. The number of such operations is at most $|E|$, which is polynomial (i.e., $|E| = O(|V|^2)$). Hence, computing reachability probability on a series-parallel graph requires only polynomial time.

Secondly, we show that the same computation on a non-series-parallel graph is not polynomial. We do so indirectly, using related work on nested paths [13] [18] [5].

Definition 3.6 (Nested Path [18]). A path P_1 from s_1 to t_1 is said to be **contained** in a path P_2 from s_2 to t_2 if the end points of P_1 (i.e., s_1 and t_1) appear as internal points in P_2. The path between s_1 and t_1 in P_2 is called the **projection** of P_1. A set $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ of paths is said to be nested if the following conditions hold.

1) For all $1 < i \leq n$, there exists $j < i$ such that P_i is contained in P_j.
2) If P_i and P_j are both contained in P_k, the projections of P_i and P_j are such that either one contains the other or they are disjoint from each other.

Fig. 7. Nested Paths

For example, in Fig. 7(a), the path P_i is contained in the path P_j, while in Fig. 7(b), the paths P_i and P_j are both contained in P_k, and the projections of P_i and P_j are disjoint from each other.

Lemma 2 (adapted from [13] [18] [5]). If a graph is not composed of nested paths, the computation of reachability probability over this graph cannot be performed in polynomial time.

The minimum unit of a non-series-parallel graph are the structures shown in Lemma 1. It is clear that none of these structures are composed of nested paths. Hence, it follows from Lemma 2 that the computation of reachability over non-series parallel graphs cannot be performed in polynomial time. □

We now turn our attention to the task of efficiently reducing all maximal series-parallel subgraphs. To do so, we rely on the concept of a 2-hop path defined as follows.

Definition 3.7 (2-hop Path). Let $G = (V, E)$ be a graph, with $v_0, v_1, v_2 \in V$, and $e_{v_0v_1}, e_{v_1v_2} \in E$. The path $P_{v_0v_1v_2}$ is a 2-hop path of v_0, and v_2 is in the 2-hop vertex set of v_0.

It is clear that the subgraph formed by a vertex and its 2-hop vertex set is series-parallel, and that its minimum units are either series links or triangle links. Hence, if $P_{v_0v_1v_2}$ is a 2-hop path of vertex v_0, and both the in-degree and out-degree of v_1 are 1, then $P_{v_0v_1v_2}$ can be reduced to a single edge $e_{v_0v_2}$ labelled by $RPr(v_0, v_2)$.

The following theorem then provides the computational motivation for our distributed algorithm.

Theorem 2. All maximal series-parallel subgraphs of an uncertain graph G are reduced after recursively detecting the 2-hop vertex set of each of its vertices.

Proof: From the foregoing presentation, given any vertex v, the edges that can be reduced are those linked to the neighbors of v whose in-degree and out-degree are both 1. The only cases where any such neighbor vertex has in-degree or out-degree larger than 1 is when there are either underlying parallel operations or the structures shown in Lemma 1. If there exist parallel operations, they are transformed into the formation of triangle links during the recursion, and can thus be reduced according to Definition 3.5. If not, then the subgraph is not series-parallel. □

From Theorem 2, we can see that the only information needed for graph reduction is the in-degree and out-degree of each vertex. In other words, if we keep track of the in-degree and out-degree of each vertex, we can find out the edges to be reduced without needing to know the structure of the whole graph. This means that each vertex can determine the edges to reduce in parallel, entirely based on the units formed by its 2-hop vertex sets. This locality property is well suited for a distributed computing environment, and is duly exploited as each slave node i executes Algorithm 1 on the portion $\mathcal{G}_i$ of the uncertain graph it receives.

Algorithm 1: The Dist-2hop Algorithm

```
   Input: G_i = (g_ci, Pr) with g_ci = (V_i, E_i), s, t
   Output: The reduced non-series-parallel graph G_i
1  V_nv = V_i
2  while V_nv is not empty do
3      Choose v_0 from V_nv
4      Lock v_0
5      Find the neighbor vertex set V_ne of v_0
6      Find the 2-hop vertex set V_2-hop of v_0
7      while V_2-hop is not empty do
8          Find a 2-hop path P_{v0 v1 v2} of v_0
9          if inDegree(v_1) == outDegree(v_1) == 1 then
10             if v_1 and v_2 are not locked then
11                 Lock v_1 and v_2
12                 Reduce v_1 and v_2
13                 if v_1 and v_2 are in the same slave with v_0 then
14                     V_2-hop = V_2-hop + children(v_2) − v_2,
                       V_i = V_i − v_1, V_nv = V_nv − v_1
15                 if v_1 is in the same slave with v_0 but v_2 is not then
16                     V_2-hop = V_2-hop − v_2, V_i = V_i − v_1,
                       V_nv = V_nv − v_1
17                 else
18                     V_2-hop = V_2-hop − v_2
19             V_ne = V_ne − v_1 + v_2
20             Unlock v_2
21         else
22             V_2-hop = V_2-hop − v_2
23             Unlock v_1 and v_2 ;
24     V_nv = V_nv − v_0 ;
25     Unlock v_0
26 Return (G_i)
```

The algorithm uses a set V_{nv} to record the vertices that are not visited in $\mathcal{G}_i$ and initializes it to the set V_i of vertices of $\mathcal{G}_i$ (Line 1). A vertex v_0 from V_{nv} is chosen, a lock is secured on v_0 so that it cannot be processed by other slaves, and v_0's neighbor vertex set V_{ne} and 2-hop vertex set $V_{2\text{-}hop}$ are computed (Lines 3-6). For each 2-hop path $P_{v_0v_1v_2}$ of v_0, if both the in-degree and out-degree of v_1 are equal to 1, and v_1 and v_2 are not locked, a lock is secured on v_1 and v_2, and the structure formed by v_0, v_1 and v_2 is reduced (Lines 8-18). Finally, the lock on v_2 is released (since v_1 is deleted, there is no need to unlock it) (Line 20). On the other hand, if either the in-degree or out-degree of v_1 is not 1, v_2 is removed from $V_{2\text{-}hop}$ (Lines 22), and both v_1 and v_2 are released (Line 23). This process is repeated until $V_{2\text{-}hop}$ is empty (Line 7). Once the process completes for v_0, the lock on v_0 is released (Line 25), and other vertices in V_{nv} are similarly handled until all vertices have been processed. Finally, the now reduced non-series-parallel graph $\mathcal{G}_i$ is returned (Line 26).

As the reduction process takes place on Lines 8-18, graph structures, i.e., relevant sets of vertices, are updated accordingly. If v_0, v_1 and v_2 are in the same slave, the structure can be updated by that slave (Line 13). In this case, v_1 is removed from both the neighbor vertex set V_{ne} of v_0 and the vertex set V_i of $\mathcal{G}_i$, v_2 is added to the neighbor set of v_0, and v_2's children (i.e., terminal vertices of the outgoing edges from v_2) are added to v_0's 2-hop vertex set $V_{2\text{-}hop}$ (Lines 14 and 19). If v_0 and v_1 are in the same slave, but v_2 is in a different slave (Line 15), v_0's slave tells v_2's slave to update its corresponding cross edges synchronously with v_0's slave. As v_1 is essentially removed from the original graph, v_1 is also removed from both V_i and V_{nv}; v_2 is removed from $V_{2\text{-}hop}$, and V_{ne} is updated by removing v_1 and adding v_2 (Lines 16 and 19). If v_0, v_1 and v_2 are in three different slaves (Line 17), v_0's slave tells both v_1's and v_2's slaves to update their corresponding cross edges synchronously with v_0's slave. Then, v_2 is removed from $V_{2\text{-}hop}$, and V_{ne} is updated by removing v_1 and adding v_2 (Lines 18 and 19).

We illustrate Algorithm 1 with the uncertain graph of Fig. 1 distributed across 3 slaves, using a_1 as the start vertex and f_2 as the terminal vertex.

Example 1 (Dist-2hop Algorithm). The 3 slaves are started at the same time. Firstly, each slave chooses a vertex, secures a lock on it to prevent conflicts, and computes its 2-hop vertex set. For instance, Slave 1 chooses a_1 and computes $V_{2\text{-}hop}(a_1) = \{c_1, d_1, f_1, u_1\}$; Slave 2 chooses a_2 and computes $V_{2\text{-}hop}(a_2) = \{c_2\}$; and Slave 3 chooses a_3 and computes $V_{2\text{-}hop}(a_3) = \{c_3, d_3\}$. For vertex a_1, Slave 1 locks a_1's neighbor vertices b_1 and c_1, and 2-hop vertices c_1, d_1, f_1, and u_1. Because the degrees of a_1's neighbor vertices, b_1 and c_1, are not 1, no 2-hop path of a_1 is changed. Slave 1 unlocks all of a_1's neighbor vertices and 2-hop vertices, and goes on to choosing another vertex, say b_1. There, it computes the 2-hop path $P_{b_1 d_1 f_1}$, locks b_1, d_1 and f_1, and finds that both the in-degree and out-degree of its neighbor d_1 are equal to 1. Slave 1 reduces the structure formed by b_1, d_1 and f_1 to a single edge $e_{b_1 f_1}$, with probability $1-(1-0.5\times0.8)\times(1-0.6) = 0.76$. As d_1 has been removed, only f_1 needs to be unlocked. Similar local computations take place in Slave 2 and Slave 3, and the resulting graph is shown in Fig. 8(a).

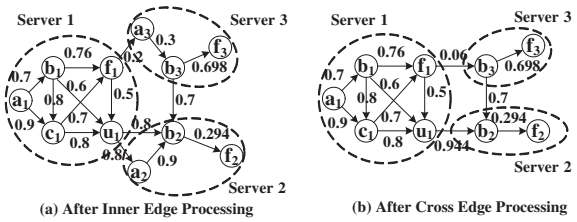


Fig. 8. Dist-2hop Processing of the Graph in Fig. 1

In addition to the inner edges that can be processed locally, information must be communicated among slaves so cross edges can also be handled. For example, when u_1 is processed, Slave 1 will find that the neighbor vertices and 2-hop vertices of u_1 are not in Slave 1. So it will send messages to Slave 2 to ask for the 2-hop vertex set of u_1 and the degree of u_1's neighbor vertices. Slave 2 returns a message that $V_{2\text{-}hop}(u_1) = \{b_2, f_2\}$. At the same time, u_1, b_2 and f_2 are locked in Slave 1 and Slave 2, respectively. From then on, the rest of the process is the same as for b_1, with the the addition that Slave 2 should update synchronously with Slave 1. The residue of the graph, after completion of the distributed execution of the Dist-2hop algorithm, is shown in Fig. 8(b).

The time complexity of Algorithm Dist-2hop is $O(|V_m|d^2)$, and the communication cost is $O(|F| \times |E_{cr}|)$, where $|V_m|$ is the maximum number of vertices in each slave, d is the average degree of the vertices in the original graph, $|F|$ is the fragment number in the distributed environment, and $|E_{cr}|$ is the number of cross edges.

3.2 Distributed Consolidation

Once the original distributed graph has been subjected to the reduction process, the final reachability probability must be computed. After running Dist-2hop, the output of each slave is of one of two forms.

- *The subgraph $\mathcal{G}_i$ received by Slave i is series-parallel*. In this case, the subgraph returned by Slave i consists of a single edge.
- *The subgraph $\mathcal{G}_i$ received by Slave i is not series-parallel*. In this case, according to Lemma 1, the subgraph returned by Slave i will be made up of single edges (from possible series-parallel subgraphs) and of the structures shown in Fig. 6.

For convenience, we refer to single edges and the structures of Fig. 6 as *P-irreducible structures*. Given the above, the residue of the original uncertain graph after the reduction step of DistR consists only of P-irreducible structures. Fig. 9 shows the five ways in which P-irreducible structures may appear combined in graphs following distributed reduction through the sharing of vertices and/or edges. Note that directions have been omitted for simplicity. Note also that each combination is representative of several as, for example, the shared node in Fig. 9(a) and (d) is irrelevant.

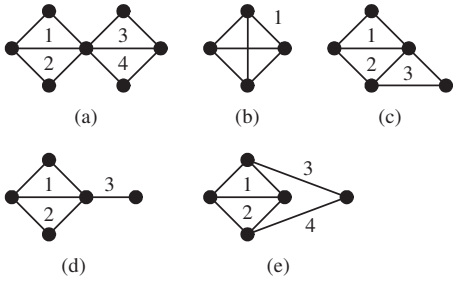


Fig. 9. P-irreducible Structure Combinations

It should be clear that adding any more edges to the structures of Fig. 9 would either result in reductions bringing them back to their original form, or yield more elaborate but similar structures. For example, if an edge sharing the right vertex of the edge labeled 3 is added to Fig. 9(d), then these two edges would from a series link (Definition 3.3) and be reduced, so that the structure would return to Fig. 9(d). On the other hand, For example, two of the structures in Fig. 9(b) could be connected by an edge, or a 5-clique could be obtained from 3 of the diamond-shaped P-irreducible structures, leading to more elaborate yet similar structures.

A quick inspection of Fig. 9 shows that all combinations of P-irreducible structures consist of k-cliques ($k \geq 2$), as highlighted by the numbers. For instance, Fig. 9(a) is made up of four 3-cliques, Fig. 9(b) is a single 4-clique, and Fig. 9(e) consists of two 3-cliques and two 2-cliques. Hence, the residue of the original uncertain graph after the distributed execution of the Dist-2hop algorithm can also be seen as consisting exclusively of k-cliques. Returning to our example, the residue of the graph of Fig. 1 shown in Fig. 8(b) consists of five 2-cliques ($\{b_2, f_2\}$, $\{b_3, f_3\}$, $\{b_2, b_3\}$, $\{f_1, b_3\}$, and $\{u_1, b_2\}$), one 3-clique ($\{a_1, b_1, c_1\}$), and one 4-clique ($\{b_1, c_1, f_1, u_1\}$).

Based on this observation, it is possible to recast the computation of reachability probability as one of a sequence of join operations over tables composed of partial results from the cliques of the residual graph. An illustration will suffice to show how the process works. Again, we use our running example from Fig. 8(b), with a_1 as the start node and f_2 as the terminal node. As mentioned above, the graph is made up of 7 cliques, which are shown separately at the top of Fig. 10. It is possible to calculate the needed RPr's locally and then to combine them. The process is as follows.

- *2-clique*. A 2-clique is a single edge, say e_{ab}. Only $RPr(a, b)$ must be computed and it is simply the weight of the corresponding edge. For example, the 2-clique $\{b_2, b_3\}$ returns $RPr(b_2, b_3) = 0.7$
- *3-clique*. A 3-clique has 3 vertices, say a, b, and c, and 3 edges. Since the graphs of interest are DAGs, exactly one vertex of the 3-clique has out-degree 2, say a, and exactly one vertex has both in-degree and out-degree equal to 1, say b. It is possible to use triangle reduction as per Definition 3.5 to compute $RPr(a, c)$. For example, in the 3-clique $\{a_1, b_1, c_1\}$, $RPr(a_1, c_1) = 1-(1-RPr(a_1, c_1))\times(1-RPr(a_1, b_1)\times RPr(b_1, c_1)) = 1-(1-0.9)\times(1-0.7\times0.8) \approx 0.96$. However, a 3-clique, since it comes from a P-irreducible structure, cannot be reduced to that single edge. The local computations here must be aware of the context of the overall graph. In particular, it is always the case that $_{bc}$ is the shared edge; if not the clique would have been reduced during the reduction phase of DistR. Hence, the 3-clique must also return $RPr(a, b)$. In our example, since e_{b1c_1} is the shared edge, the 3-clique $\{a_1, b_1, c_1\}$ must return both $RPr(a_1, c_1)$ and $RPr(a_1, b_1)$.
- *k-clique ($k \geq 3$)*. In these cases, possible world graphs have to be enumerated so as to calculate the RPr's according to

Definition 2.3. As stated earlier, this is #P-complete. Thus, we apply *Monte Carlo* sampling to compute RPr's. For each separate part, we sample its edges with their existence probability independently in different servers. For example, in the 4-clique $\{a_1, c_1, f_1, u_1\}$, the edge $e_{b_1 f_1}$ is sampled with existence probability 0.76, and using the same method, we sample $e_{b_1 c_1}$, $e_{c_1 u_1}$ and $e_{f_1 u_1}$. We apply locks to ensure that all of the shared edges, such as $e_{b_1 c_1}$ and $e_{f_1 u_1}$, are sampled only once. Then, we test whether each pair of vertices are reachable. After sampling N times in each server, we count the number of times where each pair of vertices are reachable, say n_r. Then, the RPr's are estimated by n_r/N.

The results from each clique are stored in tables, that can then easily be joined as depicted in Fig. 10. For example, joining the tables of he 3-clique $\{a_1, b_1, c_1\}$ and the 4-clique $\{a_1, c_1, f_1, u_1\}$ produces a new table with $RPr(a_1, f_1)$ and $RPr(a_1, u_1)$, where $RPr(a_1, f_1) = (RPr(a_1, b_1) \wedge RPr(b_1, f_1)) \vee (RPr(a_1, c_1) \wedge RPr(c_1, f_1)) = (0.7 \times 0.76) \vee (0.96 \times 0.7) = 0.532 \vee 0.672 = 1-(1-0.532)\times(1-0.672) \approx 0.85$. The other tables are obtained in a similar fashion until the final $RPr(a_1, f_2) = 0.25$ is obtained.

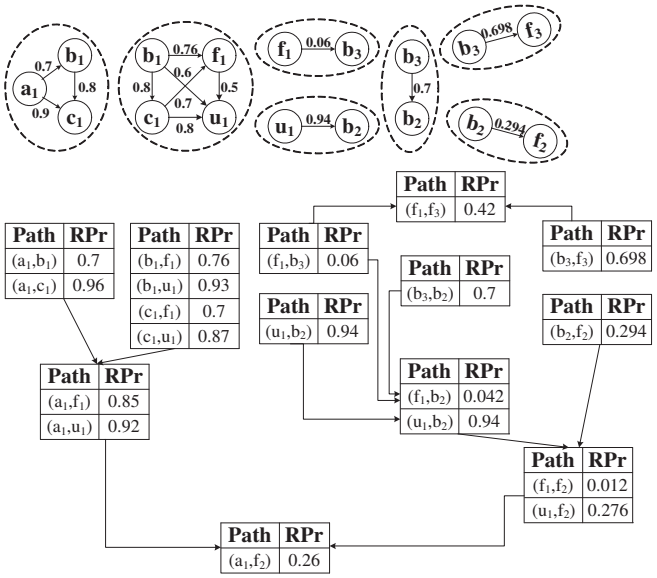


Fig. 10. Verification Method

Since each clique shares at most one edge with any other clique, and since any path going through a clique would therefore go through at most two such edges, the maximum size of each reachability probability recording table is 4. Thus, the computational cost of the consolidation step is $O(N(|V'_m| + |E'_m|) + \log n_{clique}))$, where $|V'_m|$ and $|E'_m|$ are the maximum number of remaining vertices and edges, respectively, in a slave, and n_{clique} is the number of cliques. The communication cost is $O(|V'_c + E'_c| + n_{clique})$, where $|V'_c|$ is the number of remaining vertices with cross edges, and $|E'_c|$ is the number of remaining cross edges.

As far as the accuracy of the reachability probability estimates computed by DistR, the following theorem, adapted from [20], allows us to provide a bound on the error of each clique's estimate. Note that in Monte Carlo sampling theory, usually $\xi = \eta = 0.1$.

Theorem 3. For any $0 \leq \xi \leq 1$ and $\eta > 0$, if $N \geq (4 \ln \frac{2}{\xi})/\eta^2$, then the reachability probabilities in each clique is bounded by

$$pr(|n_r/N - E(RPr))| < \xi E(RPr)) \geq 1 - \eta$$

where $E(RPr)$ is the expected value of RPr.

Finally, before turning to an experimental evaluation of DistR, we show how it can be extended with a simple graph pre-pruning technique. While not strictly necessary to the function of DistR, such a priori pruning reduces the size of the graph and may thus improve the efficiency of DistR.

3.3 Pre-pruning Strategy

Given that reachability has to do with paths from s to t, all vertices and edges that are not on any of the paths from s to t in any possible world graph can be safely pruned from the graph. For example, given the query $R(a_1, f_2)$ on the graph of Fig. 1, the vertices c_3, d_3 and f_3 (and associated edges) contribute nothing to $RPr(a_1, f_2)$ and can thus be removed from the graph. We begin with a few definitions.

Definition 3.8 (Bi-connected Component). A bi-connected component of a graph is an induced subgraph which remains connected after removing any one vertex from it.

For example, Fig. 11 shows all 5 bi-connected components of Fig. 1 (marked with the labels 1 through 5).

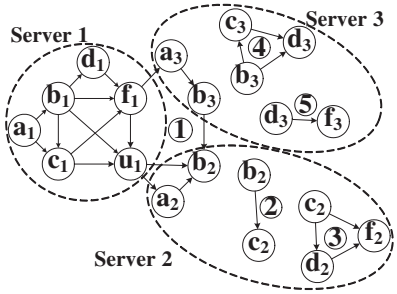


Fig. 11. Bi-connected Components of Fig. 1

The intersection of the vertex sets of two bi-connected component is a cut vertex [21], which is defined as follows.

Definition 3.9 (Cut Vertex). A vertex in a graph is a cut vertex if its removal causes the graph to become disconnected.

It follows from Definition 3.9 that any path from a vertex in a bi-connected component to a vertex in another bi-connected component must go through their cut vertex. For example, in Fig. 1, all paths from a_1 to f_2 must go through b_2 and c_2. Thus, if we treat a bi-connected component of a graph as a super vertex, this graph can be equivalently transformed into a tree [21]. We call this tree a bi-connected component tree, and define it as follows.

Definition 3.10 (Bi-connected Component Tree). A Bi-connected Component Tree is a pair $T = (VT, ET)$. Tree nodes, i.e., elements of VT, correspond to the bi-connected components of the original graph. For every pair of nodes vt_1 and vt_2 that share a cut vertex v_c, there is a directed tree edge $et_{vt_1 vt_2} \in ET$ labeled by v_c.

As an illustration, Fig. 12(a) shows the bi-connected component tree of Fig. 1, and Fig. 12(b) shows how the tree is distributed over different servers. The algorithm to build bi-connected component trees in distributed environments can be found in [21].

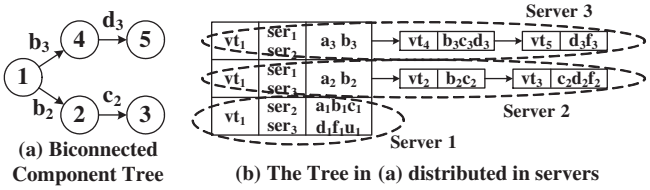


Fig. 12. The Biconnected Component Tree of Fig. 1

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2535444, IEEE Transactions on Parallel and Distributed Systems

Theorem 4 (adapted from [22]). There is a path from a start vertex s to a terminal vertex t in a DAG g_c if there is a tree path from the bi-connected component containing s to the bi-connected component containing t in the bi-connected component tree T of g_c.

It follows from Theorem 4 that if we find the bi-connected components containing the start vertex and terminal vertex respectively, and if we find a tree path between them in the distributed environment, then the vertices of the original graph found in the tree nodes not on this tree path can be pruned. The resulting pruning strategy is shown in Algorithm 2.

The algorithm first hashes s and t to the tree nodes vt_i and vt_j (Line 2) in the bi-connected component tree. A tree path PT from vt_i to vt_j is then found on the distributed bi-connected component tree (Line 3), using the method described in [21]. Note that if no such path is found, then s cannot reach t, and the DistR algorithm can terminate immediately with $RPs(s,t) = 0$. If PT exists, then only the tree nodes (bi-connected components) on PT are useful to the query; the graph vertices and edges in all other tree nodes do not contribute to the computation and can be pruned. Finally, the algorithm returns the remaining subgraph composed of the bi-connected components of PT. Following on from our earlier example, Fig. 13 shows the graph of Fig. 1 after pruning.

Algorithm 2: Bi-connected Component Pruning Algorithm

Input: distributed g_c, distributed T, s, t
Output: distributed subgraph of g_c after pruning
1 Master sends s and t to slaves
2 Hash s and t into $vt_i \in VT$ and $vt_j \in VT$
3 Slaves find a tree path PT from vt_i to vt_j
4 Return subgraph composed of vertices and edges in PT

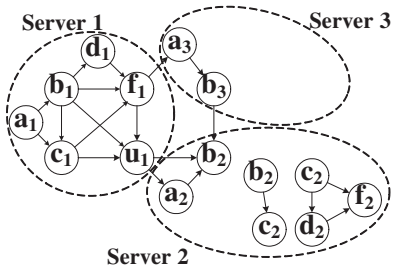


Fig. 13. The graph after Pruning

The following theorem provides the basis for computing the global reachability probability from the reachability probabilities of individual bi-connected components.

Theorem 5. Given an uncertain graph, a start vertex s and a terminal vertex t, if s and t are in different bi-connected components, say $s \in BC_1$ and $t \in BC_2$, and v_1^c, $v_2^c \ldots v_n^c$ are the cut vertices in the tree path from BC_1 to BC_2, then

$$RPr(s,t) = RPr(s,v_1^c) \times RPr(v_1^c, v_2^c) \times \ldots \times RPr(v_n^c, t) \quad (6)$$

Proof: According to Definition 3.9, any path from s to t across bi-connected components must go through the cut vertices $v_1^c \ldots v_n^c$. Thus, the occurrence of the event that s can reach t in the uncertain graph is equivalent to the simultaneous occurrences of the events that s can reach v_1^c, v_1^c can reach v_2^c, ..., and v_n^c can reach t. And it follows immediately that $RPr(s,t) = RPr(s,v_1^c) \times RPr(v_1^c, v_2^c) \times \ldots \times RPr(v_n^c, t)$. □

We note that the value of pre-pruning is highly data-dependent. Indeed, if the graph $\mathcal{G}$ has a significant number of bi-connected components and the queries issued against $\mathcal{G}$ are such that s and t are often in the same bi-connected component, then pre-pruning will offer significant advantages as large portions of $\mathcal{G}$ can be ignored. On the other hand, if $\mathcal{G}$ is such that it consists of a rather small number of bi-connected components or the queries issued against $\mathcal{G}$ are such that s and t often in different bi-connected components, then pre-pruning will have little impact since most of the graph will need to be retained.

4 Performance Evaluation

In this section, we evaluate the performance of DistR and compare it against that of relevant baseline algorithms.

4.1 Baseline Algorithms

Existing centralized (i.e., single server) solutions to the reachability problem use approximate sampling algorithms, such as random walk, to calculate the reachability probability. We straightforwardly extend these centralized solutions into their distributed counterparts to serve as baseline solutions for comparison with our proposed distributed approach.

4.1.1 Xfer-To-One Algorithm

The Xfer-To-One solution consists in first transferring the whole distributed uncertain graph to one server (provided sufficient memory is available), and then applying the following procedure.

1) Sample each edge of E according to its existence probability to produce a possible world graph g.
2) Determine whether s can reach t in g.

The process is repeated N times, and, if r is the number of possible world graphs in which s could reach t, then $RPr(s,t)$ is approximated by r/N. The computational cost of Xfer-To-One is $O(N(|V| + |E|))$, and its communication cost is $O(|V| + |E|)$.

We use the implementation of Xfer-To-One, and corresponding best estimator $\widehat{R_{RHT}}$, from [5].

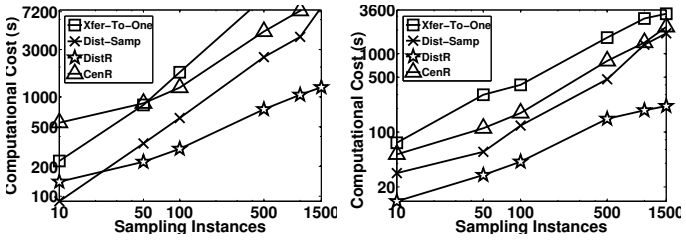
4.1.2 Dist-Samp

The Dist-Samp solution works on a distributed uncertain graph as follows.

1) Sample each edge of E according to its existence probability. In each server, first sample the inner edges. Then, whenever a server wants to sample a cross edge, it finds the other server holding that edge and sends the sampling result (whether the edge exists) to that server. After sampling all the edges of the distributed uncertain graph, the resulting graph is a distributed possible world graph g.
2) Using an algorithm for reachability query over deterministic graphs in distributed environments, such as the one proposed in [10], determine whether s can reach t in g.

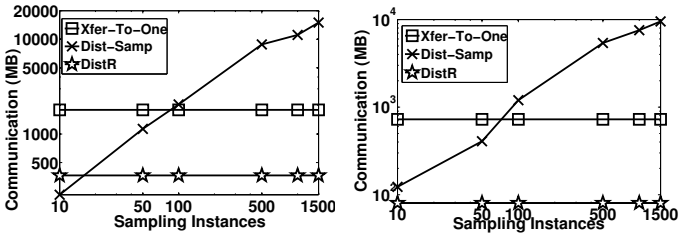
As with Xfer-To-One, the process is repeated N times, and, if r is the number of possible world graphs in which s could reach t, then $RPr(s,t)$ is approximated by r/N. The computational cost of Dist-Samp is $O(N|V_f|(|V_m| + |E_m|))$, and its communication cost is $O(N((|V_f|^2 + |E_c|)))$, where$|V_f|$ is the number of vertices involved in cross edges, $|V_m|$ (respectively, $|E_m|$) is the maximum number of vertices (respectively, edges) in each server, and $|E_c|$ is the number of cross edges.

In step (2) of Dist-Samp, we apply the algorithm proposed in [10]. Note that, the Dist-Samp method is actually an extended version of the Xfer-To-One method from centralized to distributed environments, using its $\widehat{R_B}$ estimator. The other estimators in [5] are hard to extend to distributed environment because their sampling algorithms are serial, i.e., the sampling of one edge depends on the sampling of its previous edges.



(a) LiveJournal (b) WikiTalk

Fig. 14. Computational Cost Comparison



(a) LiveJournal (b) WikiTalk

Fig. 15. Communication Cost Comparison

TABLE 1
Real Datasets

Datasets	$\|V\|$	$\|E\|$	Size (MB)	Avg. Degree
LiveJournal	4,847,571	68,993,773	1,800	28.5
WikiTalk	2,394,385	5,021,410	726	4.2
DBLP	317,080	1,049,866	115	6.6
Twitter7	17,069,982	476,553,560	24,576	55.8

4.2 Datasets

To facilitate comparison and ensure fairness, we evaluate DistR on the same three real datasets that were used for Xfer-To-One and Dist-Samp in [5] [10], namely, LiveJournal, WikiTalk, and DBLP. We also evaluate DistR on a significantly larger real dataset, Twitter7, which is too large for the centralized algorithm Xfer-To-One to execute. Details about each dataset are given in Table 1[1].

Note that these four graphs are deterministic. To add the needed uncertainty to the graphs, we generate a probability mass function using a normal distribution $N(\mu, \sigma)$, as in [5]. The default values of the distribution parameters are $\mu = 0.5$ and $\sigma = 0.1$. In practice, many graph datasets are inherently uncertain. For example, on a large social network such as Twitter or Sina Weibo, the level of interaction (measured by mentions, retweets, or direct messaging) varies significantly among users and offers a natural way of quantifying uncertainty. Unfortunately, that information is not currently stored in the social graph and, to the best of our knowledge, there are no such real-world uncertain graph benchmarks available in the literature.

In addition to the aforementioned three real datasets, we also use two groups of synthetic graphs, generated by *GraphStream*[2] to further test the scalability of the algorithms. Table 2 summarizes the characteristics of our synthetic datasets; default values are shown in bold. In the first group of graphs, the number of vertices in each graph is varied from 8M to 128M, with the average degree fixed at 10. In the second group, the number of vertices in each graph is fixed at 8M, with the average degree varying from 5 to 20.

TABLE 2
Synthetic Datasets

Number of Fragments $\|F\|$	**5**, 10, 15, 20
Number of Vertices $\|V\|$	**8M**, 16M, 32M, 64M, 128M
Average Degree of Vertices $\hat{d}$	5, **10**, 15, 20
μ	0.1, 0.3, **0.5**, 0.7, 0.9
σ	**0.1**, 0.3, 0.5, 0.7, 0.9

As stated in Section 2, we use the graph partition method in [23] to ensure the best possible balance. By default, the fragment number of the graph data is 5, denoted as $|F| = 5$. Recall that we assume that graph partitioning is performed as a pre-processing step, that is, we consider that the graphs have been distributed to different slaves prior to our experiments. We use the method in [23] to make the partition as balanced and reasonable as possible. Our cluster is composed of 1 master server and 10 slave servers. Each server has a 16GB Memory, 8TB hard disk, and 2 6-core Intel(R) Xeon(R) CPU E5-2620 @2.00GHz.

1. All real datasets are from http://snap.stanford.edu/data/.
2. http://graphstream-project.org

4.3 Experimental Results and Analysis

In all of our experiments, the results reported for computational and communication costs are the averages obtained over a set of 1,000 randomly selected pairs of query vertices on each graph dataset.

4.3.1 Performance of DistR

We begin with an analysis of the performance of the distributed graph reduction and distributed consolidation steps.

TABLE 3
Performance of Dist-2hop

	LiveJournal	WikiTalk	DBLP	Twitter7
Computational Cost (s)	111.21	8.19	1.74	1,423.28
Communication Cost (MB)	316.58	53.67	25.46	2,754.25

Table 3 shows the computational (in seconds) and communication (in MB) costs of the Dist-2hop algorithm over our four real datasets. As expected, both costs are directly related to the size of the datasets, with lower costs for smaller graphs and higher costs for larger graphs.

Table 4 shows the computational and communication costs of the distributed consolidation phase of DistR over the same four real datasets. As the number of sampling instances increases, the computational cost increases, while the communication cost stays nearly the same. This is because the sampling is processed independently in different servers to calculate the reachability probability between the cuts. An increase in sampling instances only makes the calculation in each server increase. The number of tables (probability) that need to be joined stays the same. Thus, the computational cost increases, but the communication cost does not change. This result verifies our statement in Section 3.2.

TABLE 4
Performance of the Distributed Consolidation Step

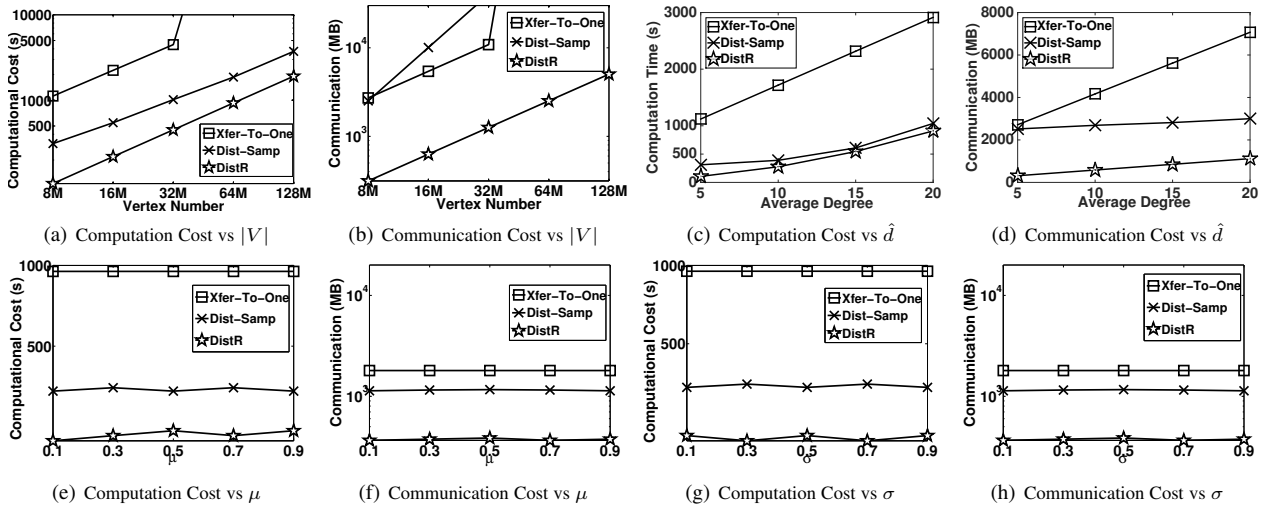
Sampling Instances	Datasets			
	LiveJournal	WikiTalk	DBLP	Twitter7
10	26.39	4.22	0.06	190.08
50	110.84	19.43	0.22	764.80
100	144.09	33.03	0.36	936.58
500	590.77	138.73	1.43	4,194.48
1000	886.15	180.35	2.78	5,494.13
1500	1,054.52	205.60	5.14	6,221.67
Communication Cost (MB)	49.73	26.79	10.14	353.08

4.3.2 Comparative Study

We now turn to a comparison of DistR with the two baseline algorithms outlined above.

TABLE 5
Relative Error of Different Algorithms on Real Datasets (%)

Sample Size	LiveJournal			WikiTalk			DBLP		
	DistR	**Dist-Samp**	**Xfer-To-One**	**DistR**	**Dist-Samp**	**Xfer-To-One**	**DistR**	**Dist-Samp**	**Xfer-To-One**
10	0.01	10.27	9.28	1.06	16.14	18.23	2.35	15.73	15.25
50	0.06	4.13	4.41	0.82	10.38	11.65	1.42	11.23	12.37
100	0.04	3.26	3.37	0.36	4.47	6.37	0.53	6.14	5.96
500	0.01	1.97	1.86	0.11	1.82	1.76	0.16	2.37	2.57
1000	0.01	1.33	1.24	0.04	1.03	0.84	0.07	1.97	1.65
1500	0.00	0.27	0.29	0.02	0.46	0.36	0.03	0.86	0.77



(a) Computation Cost vs $|V|$ (b) Communication Cost vs $|V|$ (c) Computation Cost vs $\hat{d}$ (d) Communication Cost vs $\hat{d}$

(e) Computation Cost vs μ (f) Communication Cost vs μ (g) Computation Cost vs σ (h) Communication Cost vs σ

Fig. 16. Scalability Comparison

Computational Cost: Fig. 14 graphs how computational cost varies with the number of sampling instances for LiveJournal and WikiTalk. Note that DistR has on offline process, which could also be used to improve the centralized solution. Hence, to make clear the distinction between the improvement due to our novel algorithm and the improvement inherent in the parallelization process, we also implement a centralized version of our algorithm, denoted as CenR.

Again, as expected, the computational cost of all of the algorithms increases with the increase in sampling size. However, the rates of increase for Xfer-To-One and Dist-samp are much larger than that of DistR. This is due to the fact that in DistR, only the consolidation step is influenced by sampling sizes, and the reduced graph used during consolidation is much smaller than the original graph. Moreover, when the number of sampling instances is small, CenR is slower than Xfer-To-One, and DistR is slower than Dist-Samp. As the number of sampling instances becomes larger, our algorithms are faster. This is expected, since when sampling fewer instances, our distributed graph reduction step contributes relatively more to the total computation cost. Yet the computational cost of the distributed graph reduction step remains constant over the same dataset after all, and when sampling more instances, the main contribution to the computational cost is made by the consolidation step.

Communication Cost: Fig. 15 graphs how communication cost varies with the number of sampling instances for LiveJournal and WikiTalk. As clearly shown on the graphs, the communication cost of Dist-Samp increases with the number sampling instances, while it remains nearly constant for both Xfer-To-One and DistR. The communication cost of Xfer-To-One is just the size of the graph that must be moved to the centralized server. For Dist-Samp, however, when the number of sampling instances is small, the communication cost is also small. But that cost increases rapidly with the increase in sampling instances.

As we will demonstrate shortly, if users want a more accurate answer, the number of sampling instances must be large, and the communication cost of Dist-Samp will thus be very large. On the other hand, the communication cost of DistR is independent of the number of sampling instances. Furthermore that cost is much lower (almost one order of magnitude) than the cost of Dist-Samp.

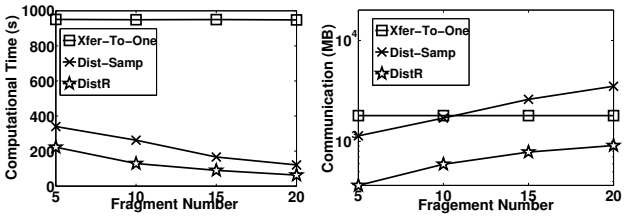
Accuracy: Table 5 summarizes the relative error of each algorithm in estimating true reachability probability. Relative error is defined here, as [5], by $\epsilon = \frac{|\widehat{RPr} - RPr|}{RPr}$.

Overall, all three algorithms have fairly low errors. Since Dist-Samp is an extension of Xfer-To-One with the $\widehat{R}_B$ estimator, its relative error is larger than that of Xfer-To-One. Since the algorithms in DistR's distributed graph reduction step are exact algorithms, and inaccuracy can only be caused by the distributed consolidation step, the relative error of DistR is the smallest, reaching almost 0 when the number of sample instances exceeds 1,500.

Scalability: Fig. 17 shows how computational and communication costs scale with respect to fragment size ($|F|$), graph size ($|V|$), and uncertainty parameters (μ and σ). The LiveJournal dataset is used to test $|F|$, μ, and σ, and the synthetic datasets are used to test $|V|$. By default, $N = 50$, $|F| = 5$, $|V| = 8M$ (synthetic data), $\mu = 0.5$, and $\sigma = 0.1$.

Scalability with respect to $|F|$ is shown in Fig. 17(a) and Fig. 17(b). As $|F|$, the computational cost of both Dist-Samp and DistR decreases, but that of Xfer-To-One remains constant at a much higher value. Across all values of $|F|$, the computational cost of DistR stays below that of Dist-Samp. Unsurprisingly, the communication costs of both Dist-Samp and DistR increase with increasing values of $|F|$, while the cost of Xfer-To-One remains constant. As with computational cost, across all values of $|F|$, the communication cost of DistR stays below that of the other algorithms.

Scalability with respect to $|V|$ is shown in Fig. 16(a) and Fig. 16(b). The computational costs of all three of the algorithms seems to grow linearly with $|V|$. When $|V|$ reaches 32M, Xfer-To-One can no longer run due to memory limitations. Across all values of $|V|$, the computational cost of DistR stays below that of Xfer-To-One and Dist-Samp. Similar results are observed for communication



(a) Computation Cost vs $|F|$ (b) Communication Cost vs $|F|$

Fig. 17. Scalability Comparison w.r.t. $|F|$

costs, with a more significant difference between the cost of DistR and the cost of Dist-Samp.

Scalability with respect to $\hat{d}$ is shown in Fig. 16(c) and Fig. 16(d). The computational costs and communication cost of all three of the algorithms seems to grow linearly with $\hat{d}$, but the increase speed of Xfer-To-One is larger than Dist-Samp and our algorithm. Among all cases, the computational cost of our algorithm is the smallest.

Scalability with respect to uncertainty (μ and σ) is shown in Fig. 16(e) to Fig. 16(h). In all four graphs, the costs remain nearly constant across the ranges of values of μ and σ. As expected from the design of all three algorithms, including DistR, computational and communication costs are independent of the graph's underlying probability mass function.

4.3.3 Summary

It is clear from the above experiments that the Xfer-To-One solution is impractical for larger graphs, and even when the graphs can fit in memory, its computational cost may be prohibitive when high accuracy (or low error) is expected. As shown above, when sampling 1,000 instances on a graph of 4.8 million vertices and 68 million edges with an error less than 1%, Xfer-To-One takes more than five hours to execute.

Superior to the Xfer-To-One solution, the Dist-Samp approach can be applied to large graphs. However, its computational cost, as well as its communication cost, can also become prohibitive with the increase of the number of sampling instances. As shown above, when sampling 1000 instances on the same graph of 4.8 million vertices and 68 million edges with an error less than 1%, Dist-Samp still needs about an hour to get the answer, and its communication cost is even higher than that of Xfer-To-One.

In contrast, DistR scales much better than both Xfer-To-One and Dist-Samp, and can produce highly accurate answers on large graphs in a fraction of the time. For example, over the same situation as above, DistR computes an answer in a little over 16 minutes, and the gap widens as the number of sampling instances, and hence accuracy level, increase.

5 Related Work

In this section, we review relevant related works along three lines of research: reachability query on deterministic graphs, distributed graph methods, and reachability query on uncertain graphs.

The reachability query on deterministic graphs can be computed easily by the depth-first search (DFS) or breadth-first search (BFS) algorithms, whose computational complexity is $O(|V|+|E|)$. To further speed up the algorithms, researchers in database area design different indexes. One famous index *2-hop labelling* [24], whose computation cost is linear to the size of labels. However, the construction of the optimal 2-hop labelling needs a large amount of time and memory. R.Jin et. al proposed a 3-hop labelling index [25] and a path tree index [26] to increase the efficiency of query and index construction. Besides the basic reachability query, there are extensive reachability queries, such as label-constraint reachability query [27] [7]. The above algorithms cannot be extended into our problem because they are all based on deterministic graph models, which is different from our uncertain graph model.

Distributed query processing has been studied on relational data [28] and XML data [29] for many years. There has also been recent work on distributed graph processing to manage large-scale graphs [30] [31] [11] [10]. In this work, we assume that data graphs are already partitioned. Indeed, how data graphs are partitioned may have a significant impact on the evaluation of our algorithm. Graph partitioning is a traditional problem that has been extensively studied since the 1970s [32] [33]. It is to find a set of non-overlapping fragments of a given graph such that (a) all fragments have a roughly equal number of nodes, and (b) the number of edges connecting nodes in different fragments is minimized. Although graph partitioning is an NP-hard problem [34], large-scale graph partitioning tools are available such as the well-known METIS [35]. A refined partition of data graphs could certainly benefit the computation of our algorithm. Prior work in this area is complementary but essentially orthogonal to ours.

Many kinds of queries are done over uncertain environment [36] [37] [38] [39] [40] [41] [42] [43]. The reachability query over uncertain graphs is widely studied using basic sampling methods in the field of network reliability [13]. Moreover, R.Jin et. al [5] proposed unequal probability sampling method to solve the uncertain reachability problem. Their proposed methods are centralized ones. If extending them directly, they turn out to be the two baseline algorithms, Xfer-To-One and Dist-Samp, which are detailed in Section 4.1. In our experiments, we have shown that our proposed method has lower computational cost and communication cost compared to the two baseline algorithms.

6 Conclusion

In this paper, we study reachability query over distributed uncertain graphs, which is to return a reachability probability from two given query vertices. As this problem is #P-complete, we use a distributed graph reduction and verification strategy to efficiently calculate the results. In the distributed graph reduction step, we reduce all the subgraphs whose reachability probability can be calculated in polynomial time to a single edge. In the distributed verification step, we transform the problem into an efficient table join operation to approximate the final answer. Our experiments show that our method has low computational cost, low communication cost and high accuracy.

Acknowledgments

Yurong Cheng and Guoren Wang is supported by the Natural Science Foundation Of China (NSFC) (Grant No. 61332006, 61332014, 61328202 and U1401256). Ye Yuan is supported by the NSFC (Grant No. 61572119 and 61173029) and the Fundamental Research Funds for the Central Universities (Grant No. N130504006). Lei Chen is supported by the NSFC (Grant No. 61328202). Yongjiao Sun is supported by the NSFC (Grant No. 61202807 and 61572121).

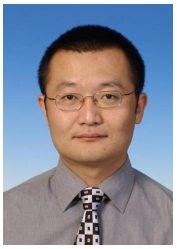
References

[1] E. Adar and C. Ré, "Managing uncertainty in social networks," *IEEE Data Eng. Bull*, vol. 30, no. 2, pp. 15–22, 2007.
[2] X. Lian and L. Chen, "Efficient query answering in probabilistic rdf graphs," in *SIGMOD*, 2011, pp. 157–168.
[3] S. Asthana, O. D. King, F. D. Gibbons, and F. P. Roth, "Predicting protein complex membership using probabilistic network reliability," *Genome Research*, vol. 14, no. 6, pp. 1170–1175, 2004.
[4] M. Hua and J. Pei, "Probabilistic path queries in road networks: traffic uncertainty aware path selection," in *EDBT*, 2010, pp. 347–358.
[5] R. Jin, L. Liu, B. Ding, and H. Wang, "Distance-constraint reachability computation in uncertain graphs," *PVLDB*, vol. 4, no. 9, pp. 551–562, 2011.
[6] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios, "K-nearest neighbors in uncertain graphs," *PVLDB*, vol. 3, no. 1-2, pp. 997–1008, 2010.
[7] L. Zou, L. Chen, and M. T. Özsu, "Distance-join: Pattern match query in a large graph database," *PVLDB*, vol. 2, no. 1, pp. 886–897, 2009.

[8] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 199–208.
[9] Y. Yuan and G. Wang, "Answering probabilistic reachability queries over uncertain graphs," *Chinese Journal of Computers*, vol. 33, no. 8, pp. 1378–1386, 2010.
[10] W. Fan, X. Wang, and Y. Wu, "Performance guarantees for distributed reachability queries," *PVLDB*, vol. 5, no. 11, pp. 1304–1316, 2012.
[11] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *SIGMOD*, 2010, pp. 135–146.
[12] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: a framework for machine learning and data mining in the cloud," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.
[13] L. G. Valiant, "The complexity of enumeration and reliability problems," *SIAM Journal on Computing*, vol. 8, no. 3, pp. 410–421, 1979.
[14] Y. Cheng, Y. Yuan, L. Chen, and G. Wang, "The reachability query over distributed uncertain graphs," in *ICDCS*, 2015, pp. 786–787.
[15] Z. Zhang, J. X. Yu, L. Qin, L. Chang, and X. Lin, "I/o efficient: computing sccs in massive graphs," in *SIGMOD*, 2013, pp. 181–192.
[16] H. Brian, T. Brunschwiler, H. Dill, H. Christ, B. Falsafi, M. Fischer, S. G. Grivas, C. Giovanoli, R. E. Gisi, R. Gutmann *et al.*, "Cloud computing," *Communications of the ACM*, vol. 51, no. 7, pp. 9–11, 2008.
[17] J. Nievergelt and K. H. Hinrichs, *Algorithms & data structures*. vdf Hochschulverl. an der ETH, 1999.
[18] D. Eppstein, "Parallel recognition of series-parallel graphs," *Information and Computation*, vol. 98, no. 1, pp. 41–55, 1992.
[19] H. L. Bodlaender and B. De Fluiter, *Parallel algorithms for series parallel graphs*. Springer, 1996.
[20] M. Mitzenmacher and E. Upfal, *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.
[21] W. Hohberg, "How to find biconnected components in distributed networks," *JPDC*, vol. 9, no. 4, pp. 374–386, 1990.
[22] F. Wei, "Tedi: efficient shortest path query answering on graphs," in *SIGMOD*, 2010, pp. 99–110.
[23] G. Karypis and V. Kumar, "Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0," 1995.
[24] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and distance queries via 2-hop labels," *SIAM Journal on Computing*, vol. 32, no. 5, pp. 1338–1355, 2003.
[25] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry, "3-hop: a high-compression indexing scheme for reachability query," in *SIGMOD*, 2009, pp. 813–826.
[26] R. Jin, Y. Xiang, N. Ruan, and H. Wang, "Efficiently answering reachability queries on very large directed graphs," in *SIGMOD*, 2008, pp. 595–608.
[27] R. Jin, H. Hong, H. Wang, N. Ruan, and Y. Xiang, "Computing label-constraint reachability in graph databases," in *SIGMOD*, 2010, pp. 123–134.
[28] D. Kossmann, "The state of the art in distributed query processing," *CSUR*, vol. 32, no. 4, pp. 422–469, 2000.
[29] D. Cavendish and K. S. Candan, "Distributed xml processing: Theory and applications," *JPDC*, vol. 68, no. 8, pp. 1054–1069, 2008.
[30] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[31] M. Giatsoglou, S. Papadopoulos, and A. Vakali, "Massive graph management for the web and web 2.0," in *New Directions in Web Data Management 1*, 2011, pp. 19–58.
[32] K. Brian W and L. S., "An efficient heuristic procedure for partitioning graphs," in *Bell System Technical Journal*, 1970, pp. 13–21.
[33] S. Yang, X. Yan, B. Zong, and A. Khan, "Towards effective partition management for large graphs," in *SIGMOD*, 2012, pp. 517–528.
[34] C. H. Papadimitriou, *Computational complexity*. John Wiley and Sons Ltd., 2003.
[35] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.
[36] M. Tang, F. Li, J. M. Phillips, and J. Jestes, "Efficient threshold monitoring for distributed probabilistic data," in *ICDE*, 2012, pp. 1120–1131.
[37] F. Li, K. Yi, and J. Jestes, "Ranking distributed probabilistic data," in *SIGMOD*, 2009, pp. 361–374.
[38] Y. Tong, L. Chen, and B. Ding, "Discovering threshold-based frequent closed itemsets over probabilistic data," in *ICDE*, 2012, pp. 270–281.
[39] Y. Tong, L. Chen, Y. Cheng, and P. S. Yu, "Mining frequent itemsets over uncertain databases," *PVLDB*, vol. 5, no. 11, pp. 1650–1661, 2012.
[40] Y. Tong, X. Zhang, and L. Chen, "Tracking frequent items over distributed probabilistic data," *World Wide Web*, pp. 1–26, 2015.
[41] Y. Tong, L. Chen, and J. She, "Mining frequent itemsets in correlated uncertain databases," *Journal of Computer Science and Technology*, vol. 30, no. 4, pp. 696–712, 2015.
[42] Y. Yuan, G. Wang, L. Chen, and H. Wang, "Efficient subgraph similarity search on large probabilistic graph databases," *PVLDB*, vol. 5, no. 9, pp. 800–811, 2012.
[43] Z. Zou, J. Li, H. Gao, and S. Zhang, "Mining frequent subgraph patterns from uncertain graph data," *TKDE*, vol. 22, no. 9, pp. 1203–1218, 2010.

Yurong Cheng received the BS degree in Computer Science from Northeastern University, China, in 2012. Currently, she is a PhD candidate in Computer Science at Northeastern University. Her main research interests include queries over queries and analysis over uncertain graphs, knowledge bases, and social networks.

Ye Yuan received the BS, MS and PhD degrees in Computer Science from Northeastern University, China, in 2004, 2007 and 2011, respectively. He is now a Professor in the College of Information Science and Engineering at Northeastern University. His research interests include graph databases, probabilistic databases, data privacy-preserving and cloud computing.



Lei Chen received the BS degree in Computer Science and Engineering from Tianjin University, China, in 1994, the MA degree from the Asian Institute of Technology, Bangkok, Thailand, in 1997, and the PhD degree in Computer Science from the University of Waterloo, Canada, in 2005. He is currently an Associate Professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His research interests include crowdsourcing over social media, social media analysis, probabilistic and uncertain databases, and privacy-preserved data publishing.

Guoren Wang received the BSc, MSc and PhD degrees in Computer Science from Northeastern University, China, in 1988, 1991 and 1996, respectively. Currently, he is a Professor in the Department of Computer Science at Northeastern University, China. His research interests include XML data management, query processing and optimization, bioinformatics, high-dimensional indexing, parallel database systems, and P2P data management.

Christophe Giraud-Carrier received the BS, MS and PhD degrees in Computer Science from Brigham Young University in 1991, 1993 and 1994, respectively. He is currently an Associate Professor in the Department of Computer Science at Brigham Young University. He has been a Senior Manager at ELCA, a Swiss IT services company, from 2001 to 2004, and a Senior Lecturer in the Department of Computer Science at the University of Bristol, from 1994 to 2001. His research interests include machine learning, data mining, computational health science and metalearning.

Yongjiao Sun received the B.S., M.S., and Ph.D., degrees in computer science from Northeastern University in 2006, 2008, and 2012, respectively. He is currently associate professor in the Department of Computer Science, Northeastern University, China. His research interests include probabilistic database, distributed data management, cloud database, and data privacy.