Fast Reliability Search in Uncertain Graphs

Arijit Khan*, Francesco Bonchi†, Aristides Gionis‡, Francesco Gullo†

*Systems Group, ETH Zurich †Yahoo Labs, Spain ‡Aalto University, Finland

ABSTRACT

Uncertain, or probabilistic, graphs have been increasingly used to represent noisy linked data in many emerging application scenarios, and have recently attracted the attention of the database research community. A fundamental problem on uncertain graphs is *reliability*, which deals with the probability of nodes being reachable one from another. Existing literature has exclusively focused on *reliability detection*, which asks to compute the probability that two given nodes are connected.

In this paper we study *reliability search* on uncertain graphs, which we define as the problem of computing all nodes reachable from a set of query nodes with probability no less than a given threshold. Existing reliability-detection approaches are not well-suited to efficiently handle the reliability-search problem. We propose RQ-tree, a novel index which is based on a hierarchical clustering of the nodes in the graph, and further optimized using a balanced-minimum-cut criterion. Based on RQ-tree, we define a fast filtering-and-verification online query-evaluation strategy that relies on a maximum-flow-based candidate-generation phase, followed by a verification phase consisting of either a lower-bounding method or a sampling technique. The first verification method returns no incorrect nodes, thus guaranteeing perfect precision, completely avoids sampling, and is more efficient. The second verification method ensures instead better recall.

Extensive experiments on real-world uncertain graphs show that our methods are very efficient—over state-of-the-art reliability-detection methods, we obtain a speed-up up to five orders of magnitude; as well as accurate—our techniques achieve precision > 0.95 and recall usually higher than 0.75.

1. INTRODUCTION

Graphs are a ubiquitous model to represent objects and their relations. In many applications, uncertainty is inherent in the data due to a variety of reasons, such as noisy measurements [2], inference and prediction models [1, 26], or explicit manipulation, e.g., for privacy purposes [8]. In these cases, data is represented as an *uncertain graph*, also called *probabilistic graph*, i.e., a graph whose arcs are accompanied with a probability of existence.

A fundamental problem in uncertain graphs is the so-called *reliability* problem, which asks to estimate the probability that two given (sets of) nodes are reachable. Reliability has been well-studied in the context of device networks (e.g., telecommunication networks): networks whose nodes are electronic devices and the (physical) links between such devices have a probability of failure [3]. More recently, the attention has been shifted to other kind of networks that can naturally be represented as uncertain graphs, such as social networks or biological networks [20, 28, 34].

The reliability problems studied so far in the literature, including all works on device-network reliability, fall into the general class of *reliability detection*. Specific problem formulations in this class ask to measure the probability that a certain reliability event occurs, e.g., what is the probability that two given nodes are connected (*two-terminal* reliability [3]), all nodes in the network are pairwise connected (*all-terminal reliability* [31]), or all nodes in a given subset are pairwise connected (*k-terminal reliability* [18]).

In this work we depart from the existing literature and focus on the problem of *reliability search*, which, to the best of our knowledge, has never been considered so far: given a probability threshold $\eta \in (0, 1)$ and a set of source nodes S, find all nodes that are reachable from S with probability no less than η.

Applications. Reliability search naturally arises in a variety of scenarios. In the problem known as *influence maximization*, whose main application is *viral marketing* [23], the probability of an arc (u, v) represents the influence that u exerts on v, i.e., the likelihood that some action of u will be adopted by v, or the likelihood that information propagates from u to v. An important, as well as the most computationally expensive step common to state-of-the-art methods, is to determine all nodes that can be influenced by a given *set* of nodes, whose computation is based on the iterative execution of reliability-search queries, as shown in Section 7.7.

In protein-interaction networks [5] nodes represent proteins and arcs represent *interactions* among them. Interactions are established for a limited number of proteins, through noisy and error-prone experiments. Thus, each arc is typically associated with a probability accounting for the existence of the interaction. In this context, predicting co-complex memberships [5, 25], and new interactions [28, 30] require to compute all proteins that are evidently (i.e., with high probability) reachable from a core (source) set of proteins: this operation exactly corresponds to running a reliability-search query using the core proteins as source nodes.

In mobile ad-hoc networks the connectivity between nodes is estimated using noisy measurements, thus leading to links naturally associated with a probability of existence. In these networks the notion of "delivery probability" is usually exploited to determine the nodes for which the probability of receiving a packet by another node in the network is adequately high [15]. Once again, the

Table 1: *Time complexity of reliability-search queries: the proposed RQ-tree-based methods vs. existing two-terminal reliability-detection methods when used for reliability search. n and m are the number of nodes and arcs in the input uncertain graph $\mathcal{G}$, d is the diameter of $\mathcal{G}$, K is the number of deterministic graphs sampled from $\mathcal{G}$, S is the set of query source nodes. $\tilde{n}$ and $\tilde{m}$ ($\tilde{n} \ll n$, $\tilde{m} \ll m$) are the number of nodes and arcs of the subgraph of $\mathcal{G}$ that the proposed RQ-tree-based methods need to visit.*

	MC-Sampling [13]	*RHT-Sampling* [20]	RQ-tree-LB (this work)	RQ-tree-MC (this work)
single-source queries	$\mathcal{O}(K(m+n))$	$\mathcal{O}(n^2 d)$	$\tilde{\mathcal{O}}(\tilde{n}\tilde{m})$	$\tilde{\mathcal{O}}(\tilde{n}\tilde{m} + K(\tilde{m}+\tilde{n}))$
multiple-source queries	$\mathcal{O}(K(m+n))$	$\mathcal{O}(n^2 d)$	$\tilde{\mathcal{O}}(\lvert S\rvert\tilde{n}\tilde{m})$	$\tilde{\mathcal{O}}(\lvert S\rvert\tilde{n}\tilde{m} + K(\tilde{m}+\tilde{n}))$

solution to this problem can be determined by reliability search.

Road networks can be modeled as uncertain graphs because of unexpected traffic jams [19]. Due to the presence of arc probabilities in such types of networks, reachability from a set of alternative source locations to a set of affordable target locations should be interpreted in a probabilistic way, thus naturally leading to queries formulated as reliability-search queries: "What are all the locations among the possible alternative ones given in input that are reachable from the source location(s) with high probability?".

Challenges. Even the simplest reliability-detection problem, i.e., two-terminal reliability, is a $\#\mathbf{P}$-complete problem [6, 32]. Thus, although exact reliability detection has received attention in the past [3], the focus nowadays, due to the large size of networks, has mainly been on approximate solutions. Most work in this regard has resorted to Monte-Carlo sampling methods [13, 23, 28], as well as other sampling techniques improving upon the efficiency of classic Monte Carlo methods (*RHT-sampling* [20]). Such approximate reliability-detection strategies can in principle be adapted to handle the novel reliability-search queries we study in this work, but, as discussed next, they are not really appropriate.

The classic Monte-Carlo approach would simply consider a set of K deterministic graph instances sampled from the input uncertain graph according to its edge probabilities, and determine all nodes reachable from the query source nodes in each graph instance: all nodes reachable in a fraction of graph instances $\geq \eta K$ are returned as answer to the query.

The RHT-sampling technique [20] can also be easily adapted to handle reliability-search queries. The idea is to make a number of $O(n)$ distinct reliability-detection queries (where n is the number of nodes in the input graph) in order to determine the probability that each node in the graph is reachable from the source nodes; the answer to the reliability-search query will be then given by all those nodes whose reliability is no less than the threshold η.

In all applications such as those listed above, however, the required rate of reliability-search queries is usually high. Thus, a fundamental requirement is to perform any single query very quickly. This makes the naïve adaptations of existing approximate reliability-detection methods not well-suited. Indeed, for an input graph of n nodes, m arcs, and diameter d, the time complexity of such adaptations is either $\mathcal{O}(K(n+m))$ (MC-sampling) or $\mathcal{O}(n^2 d)$ (RHT-sampling), which is clearly unaffordable for online computations on large-sized graphs that are commonly encountered nowadays (Table 1). This makes the problem of fast estimating reliability-search queries very challenging.

Our contributions and roadmap. In this work we study the problem of *fast online approximation of reliability-search queries on uncertain graphs*. Our solution relies on pre-computing offline information that can be exploited to speed-up online query processing. To this aim, we devise a novel index, called RQ-tree, which allows to process our queries very efficiently. Our offline indexing technique relies on a *hierarchical clustering* of the nodes in the input graph, where the hierarchical structure is based on the optimization of a principled balanced-minimum-cut criterion. Query evaluation consists of a maximum-flow-based candidate generation (filtering) step and a verification step that relies on either (a) an efficient lower bound based on the notion of most-likely path, or (b) a sampling technique applied to the candidate set only. The former verification method guarantees *perfect precision*, as it returns no incorrect (false positive) nodes (while false negatives can arise); it also avoids sampling at all, resulting in very high efficiency—the speed-up over reliability-detection baselines up to five orders of magnitude. On the other hand, sampling-based verification guarantees better accuracy (in terms of recall). The improved accuracy comes at a price of higher execution time, which, however, remains drastically less than the time required by sampling-based baselines—speed-up over reliability-detection baselines, in this case, is up to one order of magnitude.

As a further important feature, the proposed RQ-tree supports both single-source and *multiple-source reliability-search queries*. Particularly, multiple-source reliability search is a crucial generalization that is required in several real-world scenarios, such as influence maximization (see Section 7.7).

Our contributions can be summarized as follows:

- We define the fundamental problem of reliability search in uncertain graphs (Section 2).
- We devise an index, called RQ-tree, to support efficient yet effective approximate online answers to reliability-search queries (Section 3). The proposed index is based on a hierarchical clustering of the nodes in the graph. The hierarchical structure of RQ-tree derives from the optimization of a partitioning method based on the balanced-minimum-cut optimization criterion (Section 6).
- Based on RQ-tree, we develop a fast filtering-and-verification strategy (Sections 4–5). We exploit an upper bound on the probability of a set of nodes in a cluster to reach nodes outside the cluster, and a lower bound on the probability of reaching any other node. The first bound is used for candidate generation, while the latter is used for verification.
- We conduct a thorough experimental evaluation by involving several real-world uncertain graphs and comparing the proposed RQ-tree-based query-evaluation strategy with two baselines: a simple Monte-Carlo-sampling technique and the RHT-sampling method [20], both originally conceived for two-terminal reliability detection (Section 7). Results clearly attest high efficiency and accuracy of our proposal.
- We show how to apply RQ-tree in the well-known influence-maximization problem [23] (Section 7.7).

2. PROBLEM STATEMENT

An *uncertain graph* $\mathcal{G}$ is a triple (N, A, p), where N is a set of n nodes, $A \subseteq N \times N$ is a set of m directed arcs, and $p : A \to (0, 1]$ is a probability function that assigns a probability of existence to each arc in A.

The bulk of the literature on uncertain graphs and device-network reliability assumes the existence of the arcs in the graph independent from one another and interprets uncertain graphs according to the well-known *possible-world semantics* [10, 11, 18, 20, 28, 29, 31, 34]: an uncertain graph $\mathcal{G}$ with m arcs yields 2^m possible deterministic graphs, which are derived by sampling independently each arc $a \in A$ with probability $p(a)$. More precisely, a possible

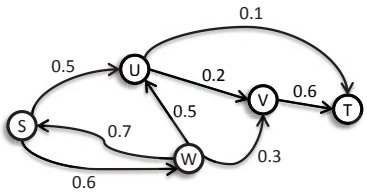

Figure 1: *Run-through example: an uncertain graph.*

graph $G \sqsubseteq \mathcal{G}$ is a pair (N, A_G), where $A_G \subseteq A$, and its sampling probability is:

$$\Pr(G) = \prod_{a \in A_G} p(a) \prod_{a \in A \setminus A_G} (1 - p(a)). \quad (1)$$

For a possible deterministic graph G, we define an indicator function $P_G(S,t)$ to be 1 if there is a path in G from a set of source nodes $S \subseteq A$ to a target node $t \in A$, and 0 otherwise. We say there is a path from the node set S to a node t if a path from *at least* one node $s \in S$ to t exists. The probability that t is reachable from S in the uncertain graph $\mathcal{G}$, denoted by $R(S,t)$, is computed as:

$$R(S,t) = \sum_{G \sqsubseteq \mathcal{G}} P_G(S,t) \Pr(G). \quad (2)$$

The number of possible worlds $G \sqsubseteq \mathcal{G}$ is exponential in the number of arcs, which makes the exact computation of $R(S,t)$ infeasible even for moderately-sized graphs.

The problem we address in this work is the following.

PROBLEM 1 (RELIABILITY SEARCH). *Given an uncertain graph $\mathcal{G} = (N, A, p)$, a probability threshold $\eta \in (0,1)$, and a set of source nodes $S \subseteq N$, find all nodes in N that are reachable from S with probability greater than or equal to η, that is, $RS(S,\eta) = \{t \in N \mid R(S,t) \geq \eta\}$.* □

EXAMPLE 1. *Consider the uncertain graph in Figure 1, and suppose one wants to compute $RS(\{s\}, 0.5)$, i.e., all nodes reachable from s with probability no less than 0.5. It is easy to see that w is part of the solution due to a direct arc from s with probability 0.6. Also, u can be reached directly, or via w; the probability that u is reachable from s is thus $1-(1-0.5)\times(1-0.6\times0.5) = 0.65$. Hence, also u belongs to the solution set. Following a similar reasoning, one may verify that the answer to the query is: $RS(\{s\}, 0.5) = \{s, u, w\}$.* □

Problem 1 is a generalization of the two-terminal reliability-detection problem, which asks to compute the probability that a target node t is reachable from a source node s. Indeed, a simple reduction from two-terminal reliability-detection to Problem 1 exists. The idea is to estimate the answer to a given instance of the former problem by performing a binary search on the threshold η. Two-terminal reliability detection is a prototypical $\#\mathbf{P}$-complete problem [6, 32]; as a consequence, Problem 1 is hard as well.

Due to its intrinsic hardness, we tackle the reliability-search problem from an approximation viewpoint. Particularly, our main goal is to develop index structures that can be exploited to speed-up online query estimation. As our focus is on approximate solutions, the answer to any reliability-search query inevitably contains errors in terms of false negatives and/or false positives. Ideally, the goal is to have answers that exhibit low false-negative and false-positive rates. However, which of these two rates is to favor really depends on the application. Some applications require high precision (i.e., low false-positive rate) such as packet-delivery probability in sensor networks [15]. In other applications we are rather more interested in high recall (i.e., low false-negative rate), such as predicting co-complex memberships by finding all proteins reachable from a core of proteins [5]. For this purpose, our proposal provides the user with a choice between two methods—the first method (Section 5.1) favors precision (guaranteeing *perfect* precision), while the second method (Section 5.2) focuses more on recall.

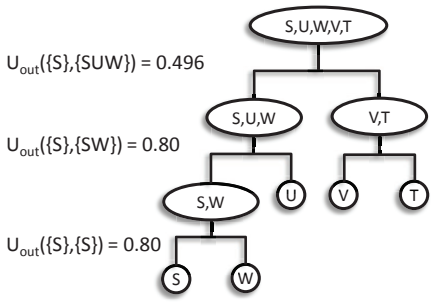

Figure 2: *An RQ-tree index for the uncertain graph in Figure 1. The upper bounds of outreach probability from* {s} *to outside various clusters are also shown (Example 2).*

3. THE RQ-TREE INDEX: OVERVIEW

The proposed index, called RQ-tree, is based on a *hierarchical clustering* of the nodes in the input uncertain graph. Specifically, the RQ-tree, hereinafter denoted by $\mathcal{T}$, is a tree, where the root contains the complete set of nodes N, and the leaves correspond to individual nodes of N. All clusters at any level i form a partition of N. A cluster at level i is partitioned into a number of children clusters at level $i+1$. As a result, there exists a unique path in $\mathcal{T}$ that connects each node $s \in N$ to the root. Such a path is composed of clusters that are all nested into each other. An example of RQ-tree index for the uncertain graph of Figure 1 is shown in Figure 2, together with some bounds that will be clarified in the next section.

Our query-processing strategy is based on two phases:

1. *Candidate generation*, where a *candidate* set of nodes is built based on the information stored into the pre-computed RQ-tree index. All nodes not belonging to the candidate set are discarded. A nice feature of this step is to guarantee that *no true positive* node is discarded from the candidate set.

2. *Verification*, where a screening is applied to the candidate set so to discard nodes that should not be part of the answer.

As the way we define the RQ-tree depends on the query processing strategy, for the sake of clarity we first present the query-processing strategy assuming an RQ-tree given (Sections 4–5), then we describe how to build the RQ-tree index (Section 6).

4. QUERY PROCESSING: CANDIDATE GENERATION

Here we describe the candidate-generation step of our online reliability-search strategy. We first present the main theoretical results (Section 4.1). Then, we discuss the case in which the source set is a singleton (Section 4.2). Finally, we focus on the general case where the source set has cardinality larger than one (Section 4.3).

4.1 Outreach probability

A key concept in our candidate-generation algorithm is the notion of *outreach probability*, which is the probability that a subset of nodes S within a cluster C in the RQ-tree index is connected to nodes outside C, i.e., within $\overline{C} = N \setminus C$.

DEFINITION 1 (OUTREACH PROBABILITY). *Given a set of nodes (cluster) $C \subseteq N$ and a subset $S \subseteq C$, the* outreach probability $R_{out}(S, C)$ *from S to outside C is defined as the probability that S reaches the nodes not belonging to C, i.e.,*

$$R_{out}(S, C) = \sum_{G \sqsubseteq \mathcal{G}} P_G(S, \overline{C}) \Pr(G) \quad (3)$$

where $P_G(S, \overline{C}) = 1$ if there exists at least a node $t \in \overline{C}$ such that $P_G(S, t) = 1$, $P_G(S, \overline{C}) = 0$ otherwise. □

Two interesting observations arise from the definition of outreach probability: if the outreach probability of S in C is smaller than η, then the probability of reaching *every* single node t outside C is also smaller than η (Observation 1), and the outreach probability values are non-decreasing for clusters that are nested into each other (Observation 2).

OBSERVATION 1. *For a cluster $C \subseteq N$ and its subset $S \subseteq C$ the following holds: if $R_{out}(S, C) < \eta$ then $R(S, t) < \eta$ for all $t \in \overline{C}$.* □

OBSERVATION 2. *Given any two clusters C_i, C_j such that $C_i \subseteq C_j$, and a set of source nodes $S \subseteq C_i$, it holds that $R_{out}(S, C_i) \geq R_{out}(S, C_j)$.* □

Observations 1 and 2 create the basis for retrieving a valid candidate set from an RQ-tree $\mathcal{T}$. Specifically, given a query $RS(S, \eta)$, consider all clusters C in $\mathcal{T}$, such that, $S \subseteq C$ and $R_{out}(S, C) < \eta$. Observation 1 guarantees that all nodes outside each of those clusters violate the reliability condition, therefore they can safely be discarded. Clearly, one wants to consider only the smallest among those clusters in order to maximize the number of pruned nodes. Observation 2 ensures that one only needs to focus on the cluster C having the largest value $R_{out}(S, C)$ that is smaller than η.

A candidate-generation strategy based on the above reasoning would require to compute outreach probabilities exactly, but such a computation is $\#\mathbf{P}$-complete. A possible solution is to approximate R_{out} values by sampling. Unfortunately, besides the well-known efficiency issues, this sampling-based solution would not guarantee that the results stated in Observations 1–2 carry over, as any sampling-based approximate R_{out} value can in general be either smaller or larger than the real R_{out} value. Instead, the validity of Observations 1–2 is still guaranteed if one would use an upper bound on R_{out}. For this reason, we next define an upper bound on R_{out} and use it in substitution for the actual R_{out} value.

Upper bound on outreach probability. While various upper bounds for reliability exist in the literature [7, 14, 24, 29], none of them is really suitable for our problem. Indeed, the outreach probability can be viewed as a special notion of *source-to-any-terminal* reliability, where one is asked to compute the probability that some source nodes are connected to *at least one* node in a target set [21]. To our knowledge, no upper bounds have been defined for this particular type of reliability problem. One might adapt the upper bounds on two-terminal reliability by interpreting source-to-any-terminal reliability as a special case of two-terminal reliability where the sources and the terminals are sets of nodes instead of single nodes. However, the upper bounds on two-terminal reliability *require to consider the entire network*, which in our context would lead to lose the pruning benefits given by the RQ-tree structure. Instead, the upper bound we propose, denoted by U_{out}, is based on the min-cut/max-flow principle and it *requires only to consider the subgraph induced by the nodes of the currently being processed cluster*. We start by defining the notion of *most-likely cut* between two disjoint sets of nodes.

Algorithm 1 Compute U_{out}

Input: an uncertain graph $\mathcal{G} = (N, A, p)$; a cluster $C \subseteq N$; a set of source nodes $S \subseteq C$
Output: $U_{out}(S, C)$
1: $\overline{C}' \leftarrow \{v \in \overline{C} \mid \exists\, u \in C : (u, v) \in A\}$
2: $A' \leftarrow \{(u, v) \in A \mid \{u, v\} \subseteq C \cup \overline{C}'\}$
3: for all $a \in A'$, set $c(a) = -\log(1 - p(a))$
4: build $\widehat{G} = (C \cup \overline{C}', A', c)$
5: $f^* \leftarrow$ MaxFlow($\widehat{G}, S, \overline{C}'$)
6: $U_{out}(S, C) \leftarrow 1 - \exp(-f^*)$

DEFINITION 2 (MOST-LIKELY CUT). *Consider a deterministic graph $G = (N, A)$ and two disjoint sets of nodes $X, Y \subseteq N$. We define a cut $\mathcal{C}(X, Y)$ between the sets X and Y to be a set of arcs in A whose removal disconnects X and Y. Consider now an uncertain graph $\mathcal{G} = (N, A, p)$ and two disjoint sets of nodes $X, Y \subseteq N$. We define the most-likely cut $\mathcal{C}^*(X, Y)$ to be a set of arcs such that: (1) it is a cut between X and Y, as defined on the deterministic graph that contains all the arcs of $\mathcal{G}$; (2) among all cuts between X and Y, it is the one that maximizes the probability of having all its arcs non-present, i.e., $\mathcal{C}^*(X, Y) = \arg\max_{\mathcal{C}(X,Y)} \prod_{a \in \mathcal{C}(X,Y)} (1 - p(a))$.* □

As stated in the following theorem, the most-likely cut provides us a way to express the desired upper bound U_{out}.

THEOREM 1. *Given a cluster $C \subseteq N$ and a subset $S \subseteq C$, it holds that:*

$$R_{out}(S, C) \leq U_{out}(S, C) = 1 - \max_{\mathcal{C}(S,\overline{C})} \prod_{a \in \mathcal{C}(S,\overline{C})} (1 - p(a)).$$

PROOF. Consider any cut $\mathcal{C}(S, \overline{C})$. From the independence assumption, the probability that none of the arcs in $\mathcal{C}(S, \overline{C})$ exists is equal to $\prod_{a \in \mathcal{C}(S,\overline{C})} (1 - p(a))$. Now, consider the event that none of the nodes in S can reach any node outside C. The probability of such an event is equal to $1 - R_{out}(S, C)$, and is clearly lower-bounded by the probability that no arc in $\mathcal{C}(S, \overline{C})$ exists. Based on this reasoning, it holds that:

$$1 - R_{out}(S, C) \geq \prod_{a \in \mathcal{C}(S,\overline{C})} (1 - p(a)), \text{ for all } \mathcal{C}(S, \overline{C}),$$

or, equivalently, $R_{out}(S, C) \leq 1 - \max_{\mathcal{C}(S,\overline{C})} \prod_{a \in \mathcal{C}(S,\overline{C})} (1 - p(a))$. The theorem follows. □

The upper bound U_{out} defined in Theorem 1 can be computed by running a max-flow algorithm on a capacitated graph appropriately derived from $\mathcal{G}$ (see Algorithm 1). Specifically, our algorithm works as follows. First, we construct a capacitated graph $\widehat{G}$, which has the same sets of nodes and arcs as $\mathcal{G}$. Each arc a in $\widehat{G}$ has a capacity $c(a)$ equal to $-\log(1 - p(a))$. Then, we compute the max-flow f^* from S to $\overline{C}$ on $\widehat{G}$.[1] As the following theorem states, the desired upper bound $U_{out}(S, C)$ can eventually be computed as $1 - \exp(-f^*)$.

THEOREM 2. *Given an uncertain graph $\mathcal{G} = (N, A, p)$, let $\widehat{G} = (N, A, c)$ be a capacitated graph derived from $\mathcal{G}$ by assigning a capacity $c(a) = -\log(1 - p(a))$ to each arc $a \in A$. Also, given a cluster $C \subseteq N$ and a set of source nodes $S \subseteq C$, let f^**

[1] To compute max-flow between a set of source nodes S and a set of sink nodes $\overline{C}$ we exploit the classic trick of asking for the max-flow between a dummy source s_0 and a dummy sink t_0, where the dummy source s_0 is connected to all nodes in S while all nodes in $\overline{C}'$ are connected to the dummy sink t_0, and all arcs outgoing from s_0 or incident to t_0 have infinite capacity.

denote the maximum flow from S to $\overline{C}$ on the graph $\widehat{G}$. It holds that $U_{out}(S,C) = 1 - \exp(-f^*)$.

PROOF. From the max-flow/min-cut equivalence, it follows that the value f^* of the max-flow is equal to the value c^* of the min-cut. We have

$$\begin{aligned} f^* = c^* = \\ &= \min_{\mathcal{C}(S,\overline{C})} \sum_{a \in \mathcal{C}(S,\overline{C})} c(a) = \min_{\mathcal{C}(S,\overline{C})} \sum_{a \in \mathcal{C}(S,\overline{C})} -\log(1-p(a)) = \\ &= \min_{\mathcal{C}(S,\overline{C})} -\log \prod_{a \in \mathcal{C}(S,\overline{C})} (1-p(a)) = \\ &= -\log \left(\max_{\mathcal{C}(S,\overline{C})} \prod_{a \in \mathcal{C}(S,\overline{C})} (1-p(a)) \right) = \\ &= -\log(1 - U_{out}(S,C)), \qquad \text{(from Theorem 1)} \end{aligned}$$

which proves the theorem. □

As stated above, the proposed upper bound U_{out} can be computed by considering only the subgraph induced by the nodes in the cluster C (and some close periphery), which leads to a significant speed-up. This is indeed possible thanks to following observation, which is exploited in Lines 1–2 of Algorithm 1.

OBSERVATION 3. *Given an uncertain graph $\mathcal{G} = (N, A, p)$, let $\widehat{G} = (N, A, c)$ be a capacitated graph derived from $\mathcal{G}$ by assigning a capacity $c(a) = -\log(1-p(a))$ to each arc $a \in A$. Given a cluster $C \subseteq N$ and a set of source nodes $S \subseteq C$, the maximum flow from S to $\overline{C}$ is equivalent to the maximum flow from S to the set $\overline{C}' \subseteq \overline{C}$ of all nodes in $\overline{C}$ having an incident arc outgoing from C, i.e., the set $\overline{C}' = \{v \in \overline{C} \mid \exists\, u \in C : (u,v) \in A\}$.* □

EXAMPLE 2. *Consider the running example in Figures 1–2. The upper bound on the outreach probability from $\{s\}$ to outside cluster $\{s,w\}$ is 0.80, due to the arcs$(s,w),(s,u)$. It means that the probability that $\{s\}$ reaches any node not belonging to $\{s,w\}$ is no greater than 0.80. Similarly, the upper bound on the outreach probability from $\{s\}$ to outside cluster $\{s,w,u\}$ is 0.496, due to the arcs $(u,t),(u,v),(w,v)$. As $\eta = 0.5$, all nodes outside cluster $\{s,w,u\}$ can be pruned.* □

4.2 Single-source queries

We next describe how to perform candidate generation when the query set of source nodes is a singleton, i.e., queries are formulated as $RS(\{s\},\eta)$.

Given a query node s, there exists a single path in the RQ-tree index $\mathcal{T}$ from the leaf cluster $\{s\}$ to the root of $\mathcal{T}$. Our candidate-generation strategy traverses all clusters along this path in a bottom-up fashion i.e., starting from the leaf cluster and going towards the root. The traversal of the path stops as soon as it encounters a *candidate cluster* C^*, whose upper bound $U_{out}(\{s\},C^*)$ on outreach probability is smaller than η. More formally:

$$C^*(\{s\},\eta) = \operatorname*{arg\,max}_{\substack{C \supseteq \{s\},\\ U_{out}(\{s\},C) < \eta}} U_{out}(\{s\},C).$$

Observation 2 ensures that (*i*) C^* is the smallest "valid" candidate cluster, i.e., the cluster that guarantees that the discarded set $\overline{C^*}$ is as large as possible; and (*ii*) all nodes $t \notin C^*$ have $R(\{s\},t) < \eta$, i.e., no true positive is discarded.

Note that, during our bottom-up traversal of $\mathcal{T}$, the upper-bound values $U_{out}(\{s\},\cdot)$ are computed in a lazy fashion according to the strategy outlined in Algorithm 1. To further speed-up query processing, one may consider pre-computing the upper-bound values $U_{out}(\{s\},C)$, for all clusters $C \in \mathcal{T}$ and all nodes $s \in C$. However, such a pre-computation would lead to an increase of the index storage space and, more importantly, the index building time, which would become $\Omega(nm)$, thus unaffordable for large graphs.

Running time. Our candidate generation consists of two steps: the bottom-up traversal of the tree $\mathcal{T}$, and the computation of the upper-bound values U_{out} during that traversal. The first step is linear in the height h of the tree $\mathcal{T}$. The second step requires performing a max-flow computation for each cluster visited during the traversal. As a result, the overall running time of computing the upper-bound values U_{out} is expressed as h max-flow computations. According to Observation 3, the max-flow computation can be performed in the subgraph induced by the nodes in the cluster and the neighbors of each of such nodes. Thus, one can upper bound the running time of each max-flow instance by using the size of the subgraph in the last (i.e., the largest-sized) cluster encountered during the traversal. Let $\tilde{n}$ and $\tilde{m}$ denote the number of nodes and arcs in that subgraph. First of all, we note that, using appropriate data structures to store the tree $\mathcal{T}$, the subgraph induced by the that cluster can be derived in $\mathcal{O}(\tilde{n}+\tilde{m})$ time. Then, concerning max-flow computation, one of the fastest existing max-flow algorithms is the one by Goldberg and Tarjan [16], whose running time is $\tilde{\mathcal{O}}(\tilde{n}\tilde{m})$, where the $\tilde{\mathcal{O}}$ notation hides logarithmic factors. Assuming that the tree $\mathcal{T}$ is balanced (see Section 6), then $h = \mathcal{O}(\log n)$. Moreover, as $\tilde{n} \ll n$, it is reasonable to assume that $n = \mathcal{O}(\tilde{n}^k)$, with k constant. This way, it holds that $h = \mathcal{O}(\log \tilde{n}^k) = \mathcal{O}(\log \tilde{n})$, and, therefore, the overall time complexity of the candidate-generation phase is $\tilde{\mathcal{O}}(\tilde{n}\tilde{m}h) = \tilde{\mathcal{O}}(\tilde{n}\tilde{m}\log\tilde{n}) = \tilde{\mathcal{O}}(\tilde{n}\tilde{m})$.

4.3 Multiple-source queries

In case of queries containing multiple source nodes, one could follow exactly the same candidate-generation strategy as in the single-source case: retrieve the smallest cluster in the index tree that contains all nodes of the query source set S. However, such a strategy may not be very effective in the multiple-source case. The reason is that the cluster enclosing all nodes in S might be a large cluster placed very close to the root of the RQ-tree $\mathcal{T}$. This would affect the efficiency of query processing, as a larger portion of $\mathcal{T}$ would be visited before encountering the desired candidate set, and thus a large number of candidate nodes would need to be verified. Therefore, we discuss next how to select a *set* of clusters (rather than a single cluster common to all source nodes) that may achieve better pruning.

Multiple-source candidate clusters. Our goal is to derive a set of clusters $\{C_i\}_{i=1}^k$ of $\mathcal{T}$ whose union set $C_\cup = \bigcup_i C_i$ meets the following requirements: (*i*) all source nodes belong to $C_\cup$; (*ii*) the property of having no false negatives discarded still holds, that is no false negatives are present among the nodes outside $C_\cup$; and (*iii*) the size of $C_\cup$ is minimum, so to guarantee maximum pruning.

We translate the above requirements into an optimization problem. Requirements (*i*) and (*iii*) are straightforward to formulate, while for requirement (*ii*) we first need to derive some theoretical results, which are formally stated in Lemma 1 and Theorem 3.

LEMMA 1. *Let $\{C_1,\cdots,C_k\}$ be a set of clusters in $\mathcal{T}$ and $\{S_1,\cdots,S_k\}$ be a set of source node sets, where $S_i \subseteq C_i$, for all i, and $S_i \cap S_j = \emptyset$, for all $i \neq j$. Let also $C_\cup = \bigcup_i C_i$ and $S_\cup = \bigcup_i S_i$. It holds that $U_{out}(S_\cup, C_\cup) \leq 1 - \prod_i (1 - U_{out}(S_i, C_i))$.*

PROOF. Given any two (disjoint) sets of nodes $X, Y \subseteq N$, let $\mathcal{C}^*(X,Y)$ denote the most-likely cut from X to Y (as defined in Definition 2). Let also $\Pr(\neg\mathcal{C}^*(X,Y)) = \prod_{a\in\mathcal{C}^*(X,Y)}(1-p(a))$ be the probability that $\mathcal{C}^*(X,Y)$ does not exist. First, we note that, by definition, the probability $\Pr(\neg\mathcal{C}^*(X,Y))$ cannot be smaller than the probability that any single valid cut from X to Y does not exist. Given any superset $Y' \supseteq Y$ (such that $X \cap Y' = \emptyset$) it is easy to see that $\mathcal{C}^*(X,Y')$ is a valid cut from X to Y too. Thus,

$$\Pr(\neg\mathcal{C}^*(X,Y)) \geq \Pr(\neg\mathcal{C}^*(X,Y')),$$

for all $Y' \supseteq Y, X \cap Y' = \emptyset$, which implies that

$$\Pr(\neg\mathcal{C}^*(S_i,\overline{C_\cup})) \geq \Pr(\neg\mathcal{C}^*(S_i,\overline{C_i})),$$

as, clearly, $\overline{C_i} \supseteq \overline{C_\cup}$ (and $S_i \cap \overline{C_\cup} = \emptyset$). Furthermore, notice that $\bigcup_i \mathcal{C}^*(S_i,\overline{C_\cup})$ is a valid cut from $S_\cup$ to $\overline{C_\cup}$. Hence, based on the same argumentation as above, the following holds:

$$\Pr(\neg\mathcal{C}^*(S_\cup,\overline{C_\cup})) \geq \Pr(\neg\textstyle\bigcup_i \mathcal{C}^*(S_i,\overline{C_\cup})).$$

Finally, the probability that none of the arcs in the union of multiple cuts exists is lower-bounded by the product of the probability that any single arc in the union cut does not exist, that is:

$$\Pr(\neg\textstyle\bigcup_i \mathcal{C}^*(S_i,\overline{C_\cup})) \geq \prod_i Pr(\neg\mathcal{C}^*(S_i,\overline{C_\cup})).$$

In summary, based on the above results, we have:

$$\underbrace{\Pr(\neg\mathcal{C}^*(S_\cup,\overline{C_\cup}))}_{1-U_{out}(S_\cup,C_\cup)} \geq \Pr(\neg\textstyle\bigcup_i \mathcal{C}^*(S_i,\overline{C_\cup})) \geq$$
$$\geq \prod_i Pr(\neg\mathcal{C}^*(S_i,\overline{C_\cup})) \geq \prod_i \underbrace{Pr(\neg\mathcal{C}^*(S_i,\overline{C_i}))}_{1-U_{out}(S_i,C_i)},$$

which implies that $U_{out}(S_\cup,C_\cup) \leq 1-\prod_i(1-U_{out}(S_i,C_i))$. The lemma follows. □

Based on the above lemma, we can now provide the ultimate condition to be ensured for having no false negatives outside $C_\cup$. As formally stated in Theorem 3, such a condition is expressed as $1-\prod_{i\in[1..k]}(1-U_{out}(C_i\cap S,C_i)) < \eta$.

THEOREM 3. *Let S be a set of source nodes and $\{C_1,\cdots,C_k\}$ be a set of clusters in $\mathcal{T}$ such that $C_i \cap S \neq \emptyset$, for all i, and $\{C_i\cap S\}_{i=1}^k$ forms a partition of S. Let also $C_\cup$ denote the union set $\bigcup_i C_i$. It holds that:*

$$1-\prod_i(1-U_{out}(C_i\cap S,C_i)) < \eta \;\Rightarrow\; R(S,t) < \eta,$$

for all $t \in \overline{C_\cup}$.

PROOF. For each node $t \in \overline{C_\cup}$, we have

$$R(S,t) \leq R_{out}(S,C_\cup) \leq U_{out}(S,C_\cup) \leq$$
$$1-\prod_i(1-U_{out}(C_i\cap S,C_i)) < \eta. \quad \text{(from Lemma 1)} \quad \square$$

The optimization problem we are interested in can now be precisely characterized.

PROBLEM 2 (MULTIPLE-SOURCE CANDIDATE GENERATION). *Given an RQ-tree index $\mathcal{T}$ and a set of source nodes S, select a set of clusters $\{C_1,\ldots,C_k\}$ of $\mathcal{T}$ so that, for the union set $C_\cup = \bigcup_{i=1}^k C_i$, the following holds:*

(i) $S \subseteq C_\cup$;

(ii) $1-\prod_i(1-U_{out}(C_i\cap S,C_i)) < \eta$;

(iii) $|C_\cup|$ *is minimum.* □

The above problem can be solved by using a dynamic-programming algorithm with $\mathcal{O}(|S|n\log n)$ max-flow computations. However, such an exact algorithm may be too slow in practice. For this purpose, we introduce next a faster greedy heuristic.

Heuristic multiple-source candidate generation. The idea of our heuristic is to perform a number of bottom-up traversals of $\mathcal{T}$ in parallel, one for each $s \in S$. Similar to the single-source case, each traversal proceeds along the path that connects the node s to the root of $\mathcal{T}$. Traversals are performed in a round-robin way and terminate when the following condition is met. Let C_i denote the current cluster in $\mathcal{T}$ that encloses node s_i at a certain point of the traversals, for all $s_i \in S$ (note that any two nodes $s_i, s_j \in S$ can be enclosed by the same cluster $C_i = C_j$). Our procedure stops when it reaches the *minimum-sized* union set $C_\cup = \bigcup_i C_i$ for which condition (ii) of Problem 2 is satisfied, i.e., $1-\prod_i(1-U_{out}(C_i\cap S,C_i)) < \eta$. The final candidate set C^* corresponds to the union set $C_\cup$ of the last clusters reached by the traversal.

Running time. The running time analysis of the (heuristic) multiple-source candidate generation roughly follows the analysis of the single-source case. We need to perform $\mathcal{O}(|S|\log n)$ max-flow computations—contrast to the $\mathcal{O}(|S|n\log n)$ max-flow computations required by the exact method, and $\mathcal{O}(|S|\log n)$ computations of U_{out}. The overall time complexity is therefore $\tilde{\mathcal{O}}(|S|\tilde{n}\tilde{m})$.

5. QUERY PROCESSING: VERIFICATION

Though guaranteed not to miss any true positive, the candidate set C^* generated according to our candidate-generation strategies may still contain false positives, i.e., nodes t for which $R(S,t) < \eta$. To filter as many of such false positives as possible out of C^*, we propose two verification methods: one method is more suited for precision, while the other method guarantees better recall. Moreover, the two proposed methods allows for trading-off between accuracy and efficiency in a different way. The high-precision method is in general very fast, while the efficiency of the high-recall method can easily be tuned (at a price of lower accuracy) by playing with a parameter (i.e., the number of samples).

Next we describe the proposed verification methods. Both verifications take as input the candidate set eventually generated by candidate generation. As a result, there is no distinction between single- and multiple-source verification.

5.1 Verification based on a lower bound on reliability

The first verification method we propose exploits a lower bound on $R(S,t)$, for any source node set S and a node $t \notin S$. The idea is that if the lower bound is $\geq \eta$, then one can safely conclude that t belongs to the solution set.

Several lower bounds on (two-terminal) reliability have been defined in the context of device networks, including Kruskal-Katona bound, polynomial-based, edge-packing-based, and cutset-enumeration-based bounds [10, 11, 14, 29]. However, these bounds require extensive computations to be performed on the entire network (their time complexity is typically in the order of $\mathcal{O}(n^k)$, with $k \geq 2$, or even more). We recall that our lower bound needs to be exploited during online query evaluation, thus it must be extremely efficient. For this purpose, we derive a novel and simpler lower bound, denoted $L_R(S,t)$, that is based on the concept of *most-likely path* from S to t, and has the advantage of being really fast.

DEFINITION 3 (MOST-LIKELY PATH). *Given a set of nodes S and a node $t \notin S$, the* most-likely path $\mathcal{P}^*(S,t)$ *from S to t is defined as*

$$\mathcal{P}^*(S,t) = \arg\max_{\substack{\mathcal{P}\in\mathbf{P}(s,t),\\ s\in S}} \prod_{a\in\mathcal{P}} p(a), \tag{4}$$

where $\mathbf{P}(s,t)$ denotes the set of all paths from s to t. □

The following theorem states that the desired lower bound L_R corresponds to the probability of the most-likely path.

THEOREM 4. *Given a set of source nodes S and a node $t \notin S$, it holds that $R(S,t) \geq L_R(S,t) = \prod_{a\in\mathcal{P}^*(S,t)} p(a)$. Here, a denotes an arc on the path $\mathcal{P}^*(S,t)$.*

PROOF. By definition, $R(S,t)$ is the probability that *at least one path* from a node $s \in S$ to t exists. Hence, $R(S,t)$ is larger than or equal to the probability that *any single path* from some $s \in S$ to t exists, that is,

$$R(S,t) \geq \prod_{a\in\mathcal{P}} p(a), \text{ for all } s \in S \text{ and all } \mathcal{P} \in \mathbf{P}(s,t).$$

Therefore, we have

$$R(S,t) \geq \max_{\substack{\mathcal{P}\in\mathbf{P}(s,t)\\ s\in S}} \prod_{a\in\mathcal{P}} p(a) = \prod_{a\in\mathcal{P}^*(S,t)} p(a),$$

which proves the theorem. □

Based on the lower bound L_R, the verification step simply consists in keeping only those nodes $t \in C^*$ such that $L_R(S,t) \geq \eta$. This way, we guarantee perfect precision.

The lower bound L_R is computed by a shortest-path computation on a weighted graph derived from $\mathcal{G}$ by assigning to each arc $a \in A$ a weight $-\log(p(a))$. An important observation here is that, the shortest-path computation can be limited to the subgraph $\tilde{\mathcal{G}}$ of $\mathcal{G}$ induced by the candidate set C^*, and this is the main reason behind the high efficiency of the proposed lower bound. The motivation is that our candidate-generation step ensures that all nodes outside the candidate set have reliability from the query sources S less than η. Hence, all paths passing through nodes not in C^* are guaranteed to have reliability less than η too and can thus be safely discarded, as the verification method would anyway keep only those nodes whose most-likely path from S has probability $\geq \eta$.

5.2 Sampling-based verification

Our second verification method performs MC-sampling to estimate the reliability of the candidate nodes, and thereby improves the recall of the lower-bounding-based verification. Unlike existing sampling methods [13, 20], sampling here is performed on a small subgraph of the input uncertain graph, i.e., the subgraph induced by the candidates only. We combine MC-sampling with a breadth first search (BFS) from the query set, and thus restrict our sampling method only inside the subgraph induced by the candidates. As a result, even though less efficient than lower-bounding-based verification, this sampling-based verification is still very fast and outperforms the baselines by an order of magnitude in efficiency (Section 7). This is indeed possible by the intrinsic characteristics of our RQ-tree index, which allows to significantly reduce the size of the subgraph where sampling is applied. One may note that, when sampling over the subgraph induced by the candidate set, the contribution of the paths passing through nodes not in the candidate set is ignored. Since all non-candidate nodes have reliability from the source set less than η, a path from the source set to a candidate node that goes through non-candidate nodes also have very small reliability as compared to η, and thus it does not significantly affect the reliability values of candidates.

Our sampling-based verification improves upon the recall of the lower-bound-based verification. As a side effect, it (slightly) decreases precision (not perfect anymore, but still very high, i.e., in the $[0.95, 1]$ range) and the efficiency. Another nice feature of sampling-based verification is that the number of samples can be used as a knob to tradeoff between efficiency and accuracy

5.3 Running time

As stated in Section 5.1, the lower-bound-based verification strategy only needs to focus on the subgraph $\tilde{\mathcal{G}}$ of $\mathcal{G}$ induced by the candidate set C^*. According to the reasoning reported in Section 4.2, the number of nodes and arcs of $\tilde{\mathcal{G}}$ are upper-bounded by $\tilde{n}$ and $\tilde{m}$, respectively. The lower-bounding-based verification strategy requires to compute the probability of the most-likely path from the source node set S to each node in the candidate set. This can be accomplished with a shortest-path distance computation in $\tilde{\mathcal{G}}$ from the source set S, which can be carried out by a simple variant of the standard Dijkstra's algorithm where the distance vector is initialized with the set source nodes S rather than a single source node. The time complexity of this Dijkstra variant remains clearly unchanged with respect to the standard algorithm, therefore the time complexity of the lower-bounding-based verification strategy is $\tilde{\mathcal{O}}(\tilde{m}+\tilde{n})$.

The sampling-based verification, on the other hand, requires to compute all nodes that are reachable from source set S in every deterministic graph sampled from subgraph $\tilde{\mathcal{G}}$ induced by the candidate set C^*. This can be accomplished by BFS in time $\mathcal{O}(K(\tilde{m}+\tilde{n}))$ time, K being the number of samples.

In Table 2 we summarize the time complexities of the various phases of the proposed query-processing strategy. It can be noted that, overall, our query processing ranges from $\tilde{\mathcal{O}}(\tilde{n}\tilde{m})$ time (single-source, lower-bounding-based verification), to $\tilde{\mathcal{O}}(|S|\tilde{n}\tilde{m} + K(\tilde{m}+\tilde{n}))$ time (multiple-source, sampling-based verification). In all cases, however, as $\tilde{n}$ and $\tilde{m}$ are very small in practice (see Section 7), the efficiency of our query processing is very high.

6. BUILDING THE RQ-TREE INDEX

In this section, we provide the guidelines for building the hierarchical structure of our RQ-tree index $\mathcal{T}$. We note that:

1. The RQ-tree should be equally effective for any reliability-search query, regardless of the source set. Intuitively, this is achieved by partitioning $\mathcal{T}$'s clusters into *balanced* children.

2. A very small height of $\mathcal{T}$ is not desirable, for example, think about the extreme case where the height of $\mathcal{T}$ is 1 (which arises when the branching factor of $\mathcal{T}$ is n): such an RQ-tree would be completely useless for our query processing strategy. Within this view, we keep the height of $\mathcal{T}$ of reasonable size by fixing the branching factor of $\mathcal{T}$ to a small number, i.e., 2 for simplicity.

3. Finally, for each cluster C in $\mathcal{T}$, and for each node $s \in C$, we require for $R_{out}(\{s\}, C)$ to be as small as possible, since this would reduce the size of the set produced during candidate generation.

Based on above requirements, we develop the following method for building an RQ-tree index $\mathcal{T}$. First, according to requirements 1) and 2), we perform a (recursive) *balanced bi-partition* of each non-leaf cluster in $\mathcal{T}$. Requirement 3), instead, provides the basis for the specific criterion to employ for defining each bi-partition.

Table 2: *RQ-tree: query-processing time complexity.*

	candidate generation	*verification*		*total*	
		lower bound	*MC-sampling*	*lower-bound verification*	*MC-sampling verification*
single-source	$\tilde{\mathcal{O}}(\tilde{n}\tilde{m})$	$\tilde{\mathcal{O}}(\tilde{m}+\tilde{n})$	$\mathcal{O}(K(\tilde{m}+\tilde{n}))$	$\tilde{\mathcal{O}}(\tilde{n}\tilde{m})$	$\tilde{\mathcal{O}}(\tilde{n}\tilde{m}+K(\tilde{m}+\tilde{n}))$
multiple-source	$\tilde{\mathcal{O}}(\lvert S\rvert\tilde{n}\tilde{m})$	$\tilde{\mathcal{O}}(\lvert S\rvert(\tilde{m}+\tilde{n}))$	$\mathcal{O}(K(\tilde{m}+\tilde{n}))$	$\tilde{\mathcal{O}}(\lvert S\rvert\tilde{n}\tilde{m})$	$\tilde{\mathcal{O}}(\lvert S\rvert\tilde{n}\tilde{m}+K(\tilde{m}+\tilde{n}))$

Particularly, for any cluster C in $\mathcal{T}$, the ideal desideratum would be to minimize the single outreach probabilities of each subset $S \subseteq C$, which is clearly unaffordable. Within this view, we first derive an upper bound that is general for the outreach probabilities of all subsets of nodes in a specific cluster, and then we search for the balanced bi-partition that minimizes this upper bound. Note that the upper bound provided here differs from the U_{out} upper bound derived in Theorem 1, because the latter is instead specific for a given set of source nodes. The expression of this general upper bound, denoted by $\mathcal{U}_{out}$, is formalized in Theorem 5.

THEOREM 5. *Given a cluster C, for all sets of source nodes $S \subseteq C$, it holds that $R_{out}(S,C) \leq \mathcal{U}_{out}(C) = 1 - \prod_{(u,v):u\in C, v\notin C}(1-p((u,v)))$.*

PROOF. By definition, $1-R_{out}(S,C)$ corresponds to the probability that no nodes $s \in S$ can reach any node t outside C. In this respect, consider all outgoing arcs of C, i.e., those arcs that connect a node in C with a node outside C. The probability that none of these arcs exists is a lower-bound for $1-R_{out}(S,C)$. As a result, it holds that $1-R_{out}(S,C) \geq \prod_{(u,v):u\in C, v\notin C}(1-p((u,v)))$, or, equivalently, $R_{out}(S,C) \leq 1-\prod_{(u,v):u\in C, v\notin C}(1-p((u,v)))$. □

Based on the above reasoning, we next formalize the optimization problem to be recursively solved for generating a bi-partition of various clusters in $\mathcal{T}$. The objective is to partition any given cluster $C \in \mathcal{T}$ into two clusters C_1 and C_2 such that (*i*) $\mathcal{U}_{out}(C_1)$ and $\mathcal{U}_{out}(C_2)$ are simultaneously minimized, and (*ii*) C_1 and C_2 have roughly the same size. Note that minimizing $\mathcal{U}_{out}(\cdot)$ is equivalent to maximizing $1-\mathcal{U}_{out}(\cdot)$, thereby our requirements are fully captured as follows:

PROBLEM 3 (BUILD-RQ-TREE). *Given a cluster $C \in \mathcal{T}$, partition C into two clusters C_1, C_2 that maximize*

$$\frac{(1-\mathcal{U}_{out}(C_1))(1-\mathcal{U}_{out}(C_2))}{|C_1|}+\frac{(1-\mathcal{U}_{out}(C_1))(1-\mathcal{U}_{out}(C_2))}{|C_2|}.$$ □

As shown in Theorem 6, Problem 3 is equivalent to MIN-RATIO-CUT [33]. As a result, Problem 3 is **NP**-hard.

THEOREM 6. *Problem 3 is* **NP***-hard.*

PROOF. We prove the theorem by a reduction from MIN-RATIO-CUT. We construct a weighted (deterministic) graph $\widehat{G}$ containing the same nodes and arcs as $\mathcal{G}$. We assign to each arc a in $\widehat{G}$ a weight $w(a) = -\log(1-p(a))$, and we make $\widehat{G}$ undirected by ignoring the directness of the arcs. Given two node sets $N_i, N_j \subseteq N$, let $A(N_i,N_j)$ denote the set of all arcs in $\widehat{G}$ between N_i to N_j. Solving MIN-RATIO-CUT on $\widehat{\mathcal{G}}$ finds a bi-partition $\{N_1,N_2\}$ of the node set N that minimizes $\frac{1}{|N_1|}\sum_{a\in A(N_1,N_2)} w(a) + \frac{1}{|N_2|}\sum_{a\in A(N_1,N_2)} w(a)$, or, equivalently, that maximizes $\frac{1}{|N_1|}\prod_{a\in A(N_1,N_2)}(1-p(a)) + \frac{1}{|N_2|}\prod_{a\in A(N_1,N_2)}(1-p(a)) = \frac{1}{|N_1|}(1-\mathcal{U}_{out}(N_1))(1-\mathcal{U}_{out}(N_2)) + \frac{1}{|N_2|}(1-\mathcal{U}_{out}(N_1))(1-\mathcal{U}_{out}(N_2))$. □

Algorithm 2 BuildRQtree

Input: an uncertain graph $\mathcal{G}=(N,A,p)$
Output: an RQ-tree index $\mathcal{T}$
1: $\mathbf{C} \leftarrow \{N\}$, $\mathcal{T} \leftarrow \{\mathbf{C}\}$
2: **repeat**
3: $\quad\mathbf{C}' \leftarrow \emptyset$
4: $\quad$**for all** $C \in \mathbf{C}$ s.t. $|C|>1$ **do**
5: $\quad\quad$build $\widehat{G}=(\widehat{N},\widehat{A},w)$, where $\widehat{N}=C$, $\widehat{A}=\{(u,v) \mid (u,v)\in A, u\in C, v\in C\}$, and $w(a)=-\log(1-p(a))$, for all $a\in\widehat{A}$
6: $\quad\quad\{C_1,C_2\} \leftarrow$ METIS $(\widehat{G})$
7: $\quad\quad\mathbf{C}' \leftarrow \mathbf{C}' \cup \{C_1,C_2\}$
8: $\quad$**end for**
9: $\quad\mathbf{C} \leftarrow \mathbf{C}'$, $\mathcal{T} \leftarrow \mathcal{T} \cup \{\mathbf{C}\}$
10: **until** $\mathbf{C}=\emptyset$

In principle, one of the existing approximation algorithms for MIN-RATIO-CUT [4] can be employed to solve our BUILD-RQ-TREE problem. But the main issues of this approach are that MIN-RATIO-CUT is hard to approximate (the best known approximation factor is polylogarithimic), and, more importantly, the existing approximation algorithms are typically inefficient. Hence, we depart from solutions having approximation guarantees and approach the problem heuristically. Specifically, in our implementation we use the well-known METIS algorithm [22], whose validity in terms of both accuracy and efficiently has been widely attested. The details of our RQ-tree building strategy are reported in Algorithm 2.

Index building time. Given a cluster C in $\mathcal{T}$, let n_C and m_C denote the number of nodes and arcs in the subgraph of the input uncertain graph $\mathcal{G}$ identified by the nodes in C, respectively. Computing a bi-partition of C by means of the METIS algorithm takes $\mathcal{O}(n_C+m_C)$ time. Running METIS on all clusters of any single level of $\mathcal{T}$ takes $\mathcal{O}(\sum_C(n_C+m_C))$. As all clusters in any single level of $\mathcal{T}$ forms a partition of the whole set of nodes in $\mathcal{G}$, the latter is equivalent to $\mathcal{O}(n+m)$. The number of levels (height) of $\mathcal{T}$ is $\mathcal{O}(\log n)$, as our RQ-tree index building strategy guarantees for $\mathcal{T}$ to be a balanced tree. As a result, the overall time complexity of building an RQ-tree index is $\mathcal{O}((n+m)\log n)$.

Index storage space. As explained above, the height of $\mathcal{T}$ is $\mathcal{O}(\log n)$. Each level of $\mathcal{T}$ contains a partition of the whole set of nodes in $\mathcal{G}$, thus each node in $\mathcal{G}$ is stored exactly one time for each level. Hence, the overall storage space required by an RQ-tree index is $\mathcal{O}(n\log n)$.

7. EXPERIMENTAL RESULTS

We present experiments to assess the performance of our RQ-tree-based reliability-search methods. We evaluate: index time/space performance (Section 7.2), query-processing accuracy and efficiency (Section 7.3), pruning power of RQ-tree (Section 7.4), performance with varying source set sizes (Section 7.5), and scalability (Section 7.6). Furthermore, as an example of real-world application, we show how our RQ-tree index can significantly improve upon the efficiency of the hill-climbing algorithm [23] used in the *influence-maximization* problem (Section 7.7).

The code is implemented in C++ and the experiments were performed on a single core of a 100GB, 2.50GHz Xeon server.

Table 3: *Dataset characteristics.*

Dataset	# Nodes	# Arcs
DBLP ($\mu = 2$)	684 911	4 569 982
DBLP ($\mu = 5$)	684 911	4 569 982
DBLP ($\mu = 10$)	684 911	4 569 982
Flickr	78 322	20 343 018
BioMine	1 008 201	13 445 048
Last.FM	6 899	24 144
WebGraph	10 000 000	174 918 788
NetHEPT	15 235	62 776

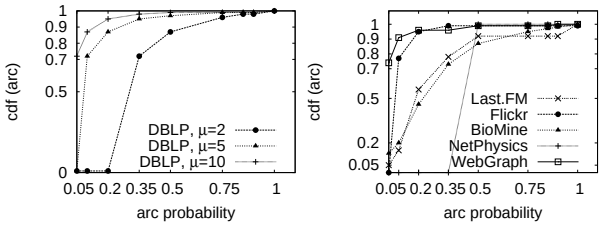

Figure 3: *Cumulative distribution of arc probabilities.*

7.1 Settings

Datasets. We involve five real-world datasets, each representing a directed uncertain graph (Table 3 and Figure 3).

DBLP (`http://www.informatik.uni-trier.de/~ley/db/`). The dataset is a subset of the popular co-authorship network used in [20, 28]. Here, the arc probabilities express the strength of the collaboration between the two incident authors. Particularly, in [20, 28], the probabilities derive from an exponential cdf of mean μ to the number of collaborations; hence, if two authors collaborated c times, the corresponding probability is $1 - \exp^{-c/\mu}$. We consider $\mu \in \{2, 5, 10\}$ in our experiments. Keeping fixed the collaborations, higher values of μ generate smaller probabilities (see Figure 3).

Flickr (`http://www.flickr.com`). Flickr is a popular online community, where users share photos, and participate in common-interest groups. We borrowed the dataset from [28], where the probability of the edge between any two users is computed assuming *homophily*, the principle that similar interests indicate social ties. In particular, [28] uses as a measure of homophily the Jaccard coefficient of the interest groups that the two users belong to.

BioMine. This is a recent snapshot of database of the BIOMINE project [30], which is a collection of biological interactions. The graph is directed, and with probability associated to the arcs quantifying the strength of the interaction [30].

Last.FM (`http://www.last.fm`). Last.FM is a music web site, where users listen to their favorite tracks, and communicate with each other based on their music preferences. We crawled a local network of Last.FM, and formed a directed graph by connecting two users if they communicated at least once. The probability on each arc (u, v) corresponds to the *influence* probability of u on v, where "influence" is interpreted according to its meaning in the influence-maximization context [23]. Following a number of works on influence maximization [12, 17, 23], the probability on any arc corresponds to the inverse of the out-degree of the node from which that arc is outgoing.

WebGraph (`http://webgraph.dsi.unimi.it`). This is the uk-2007-05 web graph data [9]. For our experiments, we use a subset containing 10M pages and 175M hyperlinks. Like *Last.FM*, the probability of the various arcs are "influence" probabilities.

NetHEPT (`http://www.arXiv.org`). This graph is created from the "High Energy Physics - Theory" section of the e-print arXiv with papers from 1991 to 2003. Like DBLP, two authors are connected by directed arcs if they co-authored at least once. This graph was used in [12] for the influence-maximization task with constant arc probabilities (0.5).

Competing methods. We evaluate the performance of our RQ-tree by focusing on both the verification strategies proposed in Section 5. Particularly, we hereinafter denote by RQ-tree-LB the variant involving lower-bounding-based verification, and by RQ-tree-MC the variant involving (Monte-Carlo-)sampling-based verification. We compare both RQ-tree-LB and RQ-tree-MC with the following baselines:

MC-Sampling. We consider a Monte-Carlo-sampling method [13] running on the whole graph. We derive K deterministic graphs by sampling the input uncertain graph according to arc probabilities. Note that the sampling is performed online, i.e., combined with a BFS from the source set, in order to improve its efficiency. Eventually, all nodes that are reachable in at least ηK sampled graphs form the answer to the reliability-search query. Such a baseline has time complexity $O(K(m + n))$.

RHT-Sampling. This method was proposed in [20] as a fast alternative to Monte-Carlo sampling for the two-terminal-reliability problem with an additional distance threshold.[3] Thus, in order to answer our reliability-search queries, it needs to be run n times, every time using a different node in the graph as target node, and the distance threshold set as d, the diameter of the uncertain graph. Following [20], the time complexity of a single execution of RHT is $\mathcal{O}(nd)$, hence, when used for reliability search, its complexity becomes $\mathcal{O}(n^2 d)$. For this purpose, even being faster than MC sampling for two-terminal reliability, RHT-sampling performs worse in the reliability-search task.

Given its quadratic complexity, we were able to obtain results for RHT in reasonable time only on our smallest datasets, i.e., *Last.FM* and *NetHEPT* (on larger graphs such as *BioMine* and *Flickr*, RHT could not finish in one day). We report such results in Table 4, where it can be evinced that our methods drastically outperform RHT: up to 2 (RQ-tree-MC) and 6 (RQ-tree-LB) orders of magnitude faster.

In the remainder of this section we focus on the larger datasets, thus leaving the RHT baseline out of the comparison.

Query workload and parameter setting. For single-source queries, we select a node uniformly at random. For multiple-source queries, we select uniformly at random a set of nodes from a subgraph of the original graph. We vary the diameter of the subgraph from 2 to 6, as these are typical distance values on most real-world graphs (small-world phenomenon), while, as the number of query nodes is usually small (up to few tens), we vary the cardinality of the query set from 2 to 20. All results are averaged over 100 sets of nodes, while the probability threshold η is varied from 0.4 to 0.8.

For all sampling methods, i.e., the baselines and our RQ-tree-MC, we observed accuracy convergence on all datasets with a number of samples K around 1 000. This is roughly the same number observed in [20, 28]. Hence, we set $K = 1\,000$ for all sampling methods.

Accuracy assessment criteria. Computing the exact answer to our reliability-search queries is computationally infeasible due to the size of our datasets. Hence, to measure accuracy of the proposed RQ-tree-based methods, we use the answer computed by MC-Sampling as a proxy. This is a reasonable choice as MC-Sampling is an unbiased estimator, thus running it for a sufficiently large number of times, its answer is expected to converge to the real answer with high probability.

[3] We use the code provided by the authors of [20].

Table 4: *Comparison between RQ-tree and RHT-sampling baseline using smaller graphs: query-processing time (sec). On larger graphs such as* BioMine *and* Flickr, RHT*-sampling baseline does not finish in one day. Thus, for larger graphs, we compare RQ-tree with only* MC*-sampling.*

	Last.FM			*NetHEPT*		
η	RHT [20]	RQ-tree-MC	RQ-tree-LB	RHT [20]	RQ-tree-MC	RQ-tree-LB
0.4	6.21	0.1	0.008	2 353	15.97	0.010
0.6	6.21	0.08	0.007	2 353	15.96	0.008
0.8	6.21	0.08	0.006	2 353	15.64	0.006

Table 5: *RQ-tree statistics and index building time.*

	time (sec)	*size (MB)*	*height*	*# clusters*
DBLP ($\mu = 5$)	1 855	123	14	735 424
Flickr	1 649	118	11	80 726
BioMine	2 890	203	15	1 040 750

We assess accuracy using *precision* and *recall*. Denoting by T the node set outputted by any selected method and by T^* the node set produced by MC-Sampling, we define precision as $\frac{|T \cap T^*|}{|T|}$ and recall as $\frac{|T \cap T^*|}{|T^*|}$. Hence, precision and recall of MC-Sampling are *always 1*, so we avoid to report them.

7.2 Indexing performance

We report the basic statistics about the RQ-tree index in Table 5. It can be observed that the offline index building time is quite modest for all datasets: for instance, building the index on *BioMine* (1M nodes and 13M arcs) takes about 50 minutes. The space requirement is contained as well: on *BioMine* our index takes approximately only 200 MB.

7.3 Query-processing performance

We now focus on the online query-processing phase of our methods. In Table 6 we show precision, recall and query-processing time of our RQ-tree-LB and RQ-tree-MC, as well as the MC-Sampling baseline, on three larger datasets; we also report these measurements on different versions of *DBLP* where the arc probabilities are varied. Specifically, *DBLP2*, *DBLP5*, and *DBLP10* in Table 6 refer to DBLP graphs with μ = 2, 5, 10, respectively.

The accuracy behavior (in terms of precision and recall) of the proposed methods perfectly conforms the design principles of the two methods: RQ-tree-LB achieves perfect precision, while RQ-tree-MC achieves very high recall ($\geq$ 0.95). It is worth noticing that the recall of RQ-tree-LB is however reasonably high as well: up to 0.96, and 0.81 on average. This attests the validity of the lower bound proposed in Section 5.1. In general, the recall of RQ-tree-LB increases as η increases. This is due to the lower-bounding verification method, which is based on the most-likely path between source and target nodes: higher probability thresholds leads to tighter lower bounds. Moreover, RQ-tree-MC exhibits very high precision too: always $\geq$ 0.95.

We also observe that the recall of RQ-tree-MC does not really depend on arc probabilities in the three variants of the *DBLP* dataset. The recall of RQ-tree-LB instead is clearly increasing as arc probabilities decrease. This is due to the RQ-tree-LB verification method, which considers the most-likely path between source and target nodes as a lower bound, and the smaller the probabilities the tighter the lower bound.

As far as running times, both our methods are evidently faster than the MC-Sampling baseline. RQ-tree-LB is even 3–5 orders of magnitude faster. Due to its improved accuracy, RQ-tree-MC is generally slower than RQ-tree-LB, as expected, but it still guarantees a significant speed-up of at least one order of magnitude.

In addition, RQ-tree-MC and MC-Sampling exhibit improved efficiency with smaller arc probabilities in *DBLP*. This is because the smaller the arc probabilities, the smaller the number of arcs in the various sampled graphs. However, also RQ-tree-LB gets faster as the arc probabilities get smaller, as this implies smaller-sized candidate sets. Indeed, the running time of both RQ-tree-MC and MC-Sampling is higher on *BioMine*, which complies with the higher arc probabilities exhibited by such a dataset (Figure 3).

7.4 Pruning power of the RQ-tree index

We next provide an insight into the properties of the proposed RQ-tree index, particularly focusing on its pruning capabilities. Here we provide evidence of the filtering guaranteed by the RQ-tree-candidate-generation phase, with a twofold goal in mind: we evaluate the pruning power of RQ-tree and, as side effect, the tightness of the upper bound proposed in Section 4.1 which the RQ-tree-candidate-generation phase relies on. We define the two following metrics:

- **Height ratio:** the ratio of the number of clusters traversed during candidate generation over the total height of the RQ-tree;
- **Candidate ratio:** the ratio of the candidate-set size, over the total number of nodes in the graph.

Clearly, the smaller the above measurements, the higher the pruning guaranteed by RQ-tree.

In Figure 4 we report height ratio and candidate ratio on *DBLP* (all the three variants), *Flickr*, and *BioMine*. Both height ratio and candidate ratio remain quite small, i.e., in the $[0.4, 0.6]$ range on average, meaning that almost half of nodes are pruned on average. The ratios are never higher than 0.75, being even less than 0.2 (height ratio) and less than 0.05 on *DBLP10*. These results confirm the usefulness of the RQ-tree in terms of pruning, as well as the effectiveness of the proposed upper bound. We also note that both the height ratio and the candidate ratio decrease with higher η, as higher η leads to smaller answer sets and thus better pruning.

As a further insight into the candidate-generation phase, we also provide in Figure 4 evidence about (i) precision (defined as $\frac{|T \cap T^*|}{|T|}$, where T is the set produced by the RQ-tree candidate generation and T^* is the MC-Sampling answer set), and (ii) running time of the candidate generation. It can be observed that the precision improves as the probability threshold increases and the arc probabilities decrease, e.g., precision is 0.75 for $\eta = 0.8$ in *DBLP* with $\mu = 5$. However, in many cases the precision is around (or even below) 0.5, meaning that half of the candidates are not part of the final solution, thus confirming the need for verification. Running times, instead, are decreasing with smaller arc probabilities and larger probability thresholds.

7.5 Performance varying the source-set size

We next analyze the performance of the RQ-tree index for multiple-source queries. For the sake of brevity, here we focus only on the RQ-tree-LB variant. Table 7 reports query-processing results on *DBLP* with $\mu = 5$ and $\eta = 0.6$. The table reports recall of our overall query-processing method, precision of the candidate generation phase, height ratio, and query-processing time. We vary both query-set size (2, 5, 10, and 20) and query diameter d, i.e., the diameter of the subgraph (of the original uncertain graph) from which the queries are randomly selected ($d = 2, 4$, and 6). Note that, as the query diameter or the number of query nodes increases, it is more likely that the smallest cluster containing all the query nodes is close to the root of the RQ-tree, thus resulting in lower pruning/efficiency. Therefore, an interesting direction for future work is to improve the indexing strategy so to provide higher pruning capacity as the cardinality of the source set increases.

Table 6: *RQ-tree: precision, recall, and query-processing time (sec) over various datasets (single-source queries).*

	precision						*recall*						*query-processing time (sec)*						
	RQ-tree-MC			RQ-tree-LB			RQ-tree-MC			RQ-tree-LB			RQ-tree-MC			RQ-tree-LB			MC
	η=0.4	η=0.6	η=0.8	η=0.4	η=0.6	η=0.8	η=0.4	η=0.6	η=0.8	η=0.4	η=0.6	η=0.8	η=0.4	η=0.6	η=0.8	η=0.4	η=0.6	η=0.8	all η
DBLP2	0.95	0.98	0.98	1	1	1	0.95	0.96	0.98	0.52	0.75	0.76	152.94	145.72	141.80	2.5	0.82	0.68	8 114
DBLP5	0.96	0.99	0.99	1	1	1	0.99	0.99	1	0.75	0.87	0.91	43.01	40.48	36.83	1.5	0.6	0.6	588.15
DBLP10	0.96	0.99	0.99	1	1	1	0.97	0.97	0.99	0.89	0.91	0.96	38.7	36.15	33.1	1.4	0.57	0.57	76.77
Flickr	0.97	0.98	0.98	1	1	1	0.98	0.99	0.99	0.76	0.79	0.83	60.23	58.6	54.75	0.21	0.2	0.17	114.29
BioMine	0.95	0.96	0.97	1	1	1	0.97	0.98	0.98	0.77	0.81	0.85	6 062	5 417	4 974	1	0.5	0.5	25 608

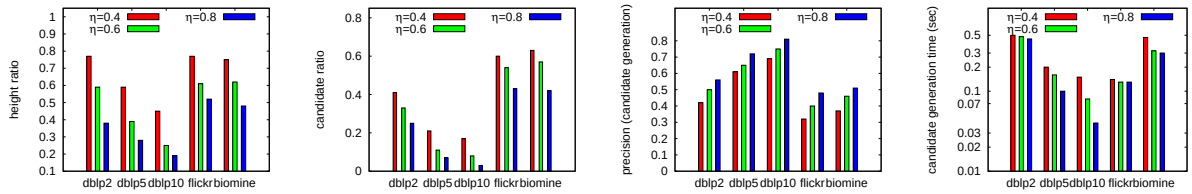

Figure 4: *RQ-tree pruning power: height ratio, candidate ratio, candidate-generation precision, and candidate-generation time (sec).*

Table 7: *RQ-tree*-LB *query-processing results on* DBLP *($\mu = 5$, $\eta = 0.6$), varying the size of the set of query nodes (first column) and the diameter (d) of the subgraph from which these nodes were picked.*

# nodes	recall			precision (candidate generation)			height ratio			RQ-tree-LB runtime (sec)			MC runtime (sec)		
	$d=2$	$d=4$	$d=6$	$d=2$	$d=4$	$d=6$	$d=2$	$d=4$	$d=6$	$d=2$	$d=4$	$d=6$	$d=2$	$d=4$	$d=6$
2	0.85	0.86	0.82	0.65	0.61	0.55	0.40	0.41	0.44	0.60	0.60	0.67	1 201	1 377	1 417
5	0.82	0.85	0.82	0.60	0.45	0.24	0.40	0.57	0.81	0.61	0.87	2.50	2 498	3 063	3 137
10	0.82	0.81	0.81	0.55	0.37	0.17	0.40	0.80	0.87	0.60	2.35	3.32	5 077	5 155	5 470
20	0.76	0.76	0.75	0.55	0.17	0.13	0.45	0.93	0.95	0.71	3.41	4.20	7 102	7 200	7 457

Table 8: *Scalability analysis using single-source queries with $\eta = 0.6$ on the* WebGraph *dataset.*

# nodes, # arcs	*size*	*height*	*# clusters*	*indexing (sec)*	*query proc. (sec)*
1M, 15M	62 MB	17	1 202 754	1 221	0.11
3M, 50M	177 MB	18	3 410 221	7 312	0.13
5M, 81M	421 MB	19	5 810 934	11 273	0.17
7M, 122M	813 MB	21	9 570 259	25 315	0.21
10M, 175M	1 220 MB	21	11 758 022	37 146	0.27

7.6 Scalability

We analyze the scalability of our RQ-tree on *WebGraph*. For this experiment, we consider subgraphs of the original *WebGraph* with a number of nodes 1M, 3M, 5M, 7M, and 10M, respectively. The corresponding index building space and time, as well as the query-processing time, are reported in Table 8. We observe that the index time increases polynomially with the number of nodes in the graph, while the query time is linear in the size of the graph. Such results assess the high scalability of our RQ-tree.

7.7 Application: Influence Maximization

The *influence-maximization* problem [23], whose primary application is *viral marketing*, has received a great deal of attention over the last decade. It requires to find a set S of k nodes that maximize the *expected spread*, i.e., the expected number of nodes that would be infected by a viral propagation started in S. The *independent cascade model* [23] is a widely used propagation model: according to which the expected spread can be formulated as $\sigma(S) = \sum_{t \in A} R(S,t)$.

The problem of finding a set S of k nodes that maximizes $\sigma(S)$ is hard. However, thanks to the submodularity of $\sigma(S)$, the Greedy algorithm that iteratively adds to S the node bringing the largest marginal gain in the objective function provides $(1-\frac{1}{e})$ approximation guarantee [23]. Unfortunately, finding the maximum-marginal-gain node requires to solve a $\#\mathbf{P}$-complete reliability problem. Hence, existing approaches usually apply sampling (e.g., Monte Carlo) to estimate the best seed node at each iteration of the Greedy algorithm. Next, we show how the classic Greedy algorithm can exploit our RQ-tree-LB method, thus achieving high speed-up and paying almost nothing in terms of accuracy.

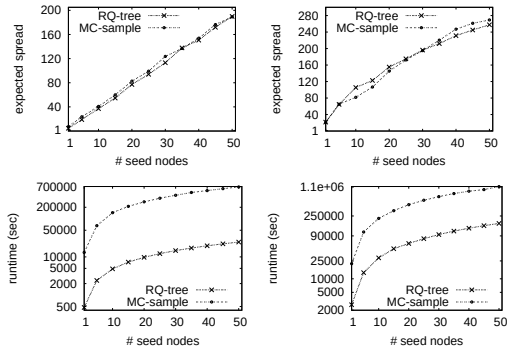

Figure 5: *Exploiting RQ-tree in the influence-maximization problem: comparison to the MC-based baseline in terms of expected spread and running time (sec) on* Last.FM *(left) and* NetHEPT *(right)*

At each iteration, given the current set of nodes S, the Greedy algorithm needs to find the node $w \in N \setminus S$ that maximizes $\sum_{t \in A} R(S \cup \{w\}, t)$. We use a histogram-based method to exploit our RQ-tree. We fix a few probability threshold values in ascending order, i.e., $\eta_1 < \eta_2 < \ldots < \eta_p$. Let $f(S, \eta_i)$ denote the size of the reliability-search set $RS(S, \eta_i)$: we compute the expected spread of S as $f(S,\eta_p)\eta_p + [f(S,\eta_p) - f(S,\eta_{p-1})]\eta_{p-1} + \ldots + [f(S,\eta_2) - f(S,\eta_1)]\eta_1$.

We compare the Greedy algorithm coupled with Monte-Carlo sampling (1 000 samples), and the same algorithm equipped with RQ-tree-LB: the results on *Last.FM* and *NetHEPT* are reported in Figure 5 (we focus only on our smallest datasets to allow MC-based

Greedy to terminate in reasonable time). For accuracy evaluation, we measure the expected spread of the node set outputted by the two competing methods via Monte-Carlo sampling. We observe that the two methods achieve roughly the same expected spread, while, as far as running time, employing RQ-tree-LB leads to at least one order of magnitude of speed-up.

8. RELATED WORK

Reliability is a classic problem studied in device networks. A number of variants to the problem have been defined, including two-terminal reliability [32], all-terminal reliability [31], and k-terminal reliability [18], and many solutions have been proposed, either exact (see [3] for a survey) or approximate [13, 21, 23]. Approximate solutions, in particular, have mainly involved (Monte-Carlo) sampling methods [13, 23]. More recently, the problem has been studied in the context of more general types of uncertain graphs, such as social networks and biological networks [20, 28, 34], as well as in the context of clustering [27]. Particularly, Jin et al. [20] deal with distance-constrained reliability queries, i.e., they ask for the probability that any two nodes have distance no greater than a user-defined threshold.

However, all existing works on reliability fall into the class of *reliability detection*, whose goal is to determine the probability of a certain reliability event. In this work we study a novel type of reliability problem, that is *reliability search*. A problem that is closer to reliability search than the above ones is the problem of threshold-based probabilistic reachability [34], which consists in determining if two nodes are connected with probability higher than a threshold. But, like the problem in [20], the input there is a pair of nodes; hence, applying the methods in [34] to our reliability search would lead to quadratic (thus unaffordable) time complexity.

Further research on reliability has concerned the definition of polynomial-time upper/lower bounds to reliability problems [7, 10, 11, 14, 24, 29]. We have already discussed in Section 4.1 and 5.1 how the bounds proposed in this work differ from the existing ones.

9. CONCLUSIONS

In this paper we studied reliability search, a novel reliability problem for uncertain graphs. We defined RQ-tree, a novel index that allows for answering online reliability-search queries efficiently and effectively, as confirmed by an extensive experimental evaluation conducted on real-world datasets: RQ-tree provides one order of magnitude efficiency gain for MC-sampling by its own, while our overall RQ-tree-based methods outperform existing sampling methods up to five orders of magnitude in efficiency, while also exhibiting precision and recall usually ≥ 0.95 and ≥ 0.75, respectively.

Our experiments show that the performance of RQ-tree can be further improved when the arc probabilities get higher and/or the size of the source set increases. Thus, a natural direction for future work is to improve the indexing strategy in order to handle better multi-source reliability-search queries and higher arc probabilities. We also plan to study the theoretical properties of the proposed upper and lower bounds, as well as consider the case where arc probabilities are not independent.

10. REFERENCES

[1] E. Adar and C. Re. Managing Uncertainty in Social Networks. *IEEE Data Eng. Bull.*, 2007.

[2] C. C. Aggarwal. *Managing and Mining Uncertain Data*. Springer, 2009.

[3] K. K. Aggarwal, K. B. Misra, and J. S. Gupta. Reliability Evaluation A Comparative Study of Different Techniques. *Micro. Rel.*, 1975.

[4] S. Arora, J. R. Lee, and A. Naor. Euclidean distortion and the sparsest cut. In *STOC*, 2005.

[5] S. Asthana, O. D. King, F. D. Gibbons, and F. P. Roth. Predicting Protein Complex Membership using Probabilistic Network Reliability. *Genome Res.*, 2004.

[6] M. O. Ball. Computational Complexity of Network Reliability Analysis: An Overview. *IEEE Tran. on Reliability*, 1986.

[7] M. O. Ball and J. S. Provan. Disjoint Products and Efficient Computation of Reliability. *Oper. Res.*, 1988.

[8] P. Boldi, F. Bonchi, A. Gionis, and T. Tassa. Injecting Uncertainty in Graphs for Identity Obfuscation. *PVLDB*, 2012.

[9] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *WWW*, 2004.

[10] T. B. Brecht and C. J. Colbourn. Lower Bounds on Two-Terminal Network Reliability. *Disc. App. Math.*, 1988.

[11] D. Bulka and J. B. Dugan. Network s-t Reliability Bounds using a 2-Dimensional Reliability Polynomial. *IEEE Tran. Rel.*, 1994.

[12] W. Chen, Y. Wang, and S. Yang. Efficient Influence Maximization in Social Networks. In *KDD*, 2009.

[13] G. S. Fishman. A Comparison of Four Monte Carlo Methods for Estimating the Probability of s-t Connectedness. *IEEE Tran. Rel.*, 1986.

[14] J. Galtier, A. Laugier, and P. Pons. Algorithms to Evaluate the Reliability of a Network. In *DRCN*, 2005.

[15] J. Ghosh, H. Q. Ngo, S. Yoon, and C. Qiao. On a Routing Problem Within Probabilistic Graphs and its Application to Intermittently Connected Networks. In *INFOCOM*, 2007.

[16] A. V. Goldberg and R. E. Tarjan. A New Approach to the Maximum-Flow Problem. *J. ACM*, 1988.

[17] A. Goyal, W. Lu, and L. V. S. Lakshmanan. CELF++: Optimizing the Greedy Algorithm for Influence Maximization in Social Networks. In *WWW*, 2011.

[18] G. Hardy, C. Lucet, and N. Limnios. K-Terminal Network Reliability Measures With Binary Decision Diagrams. *IEEE Tran. Rel.*, 2007.

[19] M. Hua and J. Pei. Probabilistic Path Queries in Road Networks: Traffic Uncertainty aware Path Selection. In *EDBT*, 2010.

[20] R. Jin, L. Liu, B. Ding, and H. Wang. Distance-Constraint Reachability Computation in Uncertain Graphs. *PVLDB*, 2011.

[21] J. Jonczy and R. Haenni. A New Approach to Network Reliability. In *MMR*, 2007.

[22] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J.*, 1999.

[23] D. Kempe, J. M. Kleinberg, and E. Tardos. Maximizing the Spread of Influence through a Social Network. In *KDD*, 2003.

[24] A. Konak and A. Smith. An Improved General Upper Bound for All-Terminal Network Reliability. In *Industrial Eng. Res. Conf.*, 1998.

[25] N. J. Krogan, G. Cagney, and al. Global Landscape of Protein Complexes in the Yeast Saccharomyces Cerevisiae. *Nature*, 2006.

[26] D. L.-Nowell and J. Kleinberg. The Link Prediction Problem for Social Networks. In *CIKM*, 2003.

[27] L. Liu, R. Jin, C. C. Aggarwal, and Y. Shen. Reliable Clustering on Uncertain Graphs. In *ICDM*, 2012.

[28] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. k-Nearest Neighbors in Uncertain Graphs. *PVLDB*, 2010.

[29] J. S. Provan and M. O. Ball. Computing Network Reliability in Time Polynomial in the Number of Cuts. *Operations Research*, 1984.

[30] P. Sevon, L. Eronen, P. Hintsanen, K. Kulovesi, and H. Toivonen. Link Discovery in Graphs Derived from Biological Databases. In *DILS*, 2006.

[31] A. Sharafat and O. Ma'rouzi. All-Terminal Network Reliability Using Recursive Truncation Algorithm. *IEEE Tran. on Rel.*, 2009.

[32] L. G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM J. on Computing*, 1979.

[33] Y.-C. Wei and C.-K. Cheng. Ratio Cut Partitioning for Hierarchical Designs. *IEEE Trans. CAD IC and Sys.*, 1991.

[34] K. Zhu, W. Zhang, G. Zhu, Y. Zhang, and X. Lin. BMC: An Efficient Method to Evaluate Probabilistic Reachability Queries. In *DASFAA*, 2011.