World Wide Web (2014) 17:161–188
DOI 10.1007/s11280-012-0200-6

Aggregate nearest neighbor queries in uncertain graphs

Zhang Liu · Chaokun Wang · Jianmin Wang

Received: 4 April 2012 / Revised: 16 September 2012 /
Accepted: 20 December 2012 / Published online: 14 February 2013

Abstract Most recently, uncertain graph data begin attracting significant interests of database research community, because uncertainty is the intrinsic property of the real-world and data are more suitable to be modeled as graphs in numbers of applications, e.g. social network analysis, PPI networks in biology, and road network monitoring. Meanwhile, as one of the basic query operators, *aggregate nearest neighbor* (ANN) query retrieves a data entity whose aggregate distance, e.g. sum, max, to the given query data entities is smaller than those of other data entities in a database. ANN query on both certain graph data and high dimensional data has been well studied by previous work. However, existing ANN query processing approaches cannot handle the situation of uncertain graphs, because topological structures of an uncertain graph may vary in different possible worlds. Motivated by this, we propose the aggregate nearest neighbor query in uncertain graphs (UG-ANN) in this paper. First of all, we give the formal definition of UG-ANN query and the basic UG-ANN query algorithm. After that, to improve the efficiency of UG-ANN query processing, we develop two kinds of pruning approaches, i.e. structural pruning and instance pruning. The structural pruning takes advantages the monotonicity of the aggregate distance to derive the upper and lower bounds of the aggregate distance for reducing the graph size. Whereas, the instance pruning decreases the number of possible worlds to be checked in the searching tree. Comprehensive experimental

Z. Liu (✉)
Department of Computer Science and Technology, Tsinghua University,
100084 Beijing, People's Republic of China
e-mail: liuzhang08@mails.tsinghua.edu.cn

C. Wang · J. Wang
School of Software, Tsinghua University,
100084 Beijing, People's Republic of China
e-mail: chaokun@mail.tsinghua.edu.cn

J. Wang
e-mail: jimwang@tsinghua.edu.cn

results on real-world data sets demonstrate that the proposed method significantly improves the efficiency of the UG-ANN query processing.

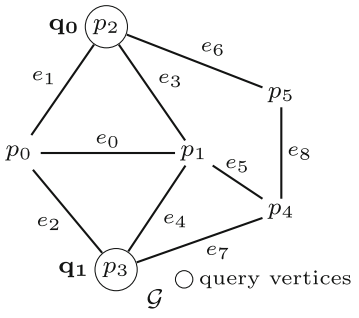
Keywords Uncertain graph · Aggregate nearest neighbor · Query processing

1 Introduction

Aggregate nearest neighbor (ANN) query [15, 17, 22] is one of the most useful operators for analyzing networks and graph data, e.g. social networks [19], traffic networks [22], and biology networks [2]. ANN query aims to find a data point whose aggregate distance to a set of query points is minimum. Formally, given a data set S, a query $Q \subseteq S$ and a distance metric $d(\cdot, \cdot)$, the ANN query retrieves a data point $p \in S$ such that $\forall p' \in S \setminus \{p\}$, $f_{q \in Q} d(p, q) \leq f_{q \in Q} d(p', q)$ holds, where f is an aggregate function, e.g., max, sum [17]. ANN query has proved itself useful in numbers of real-world applications. Take the social network as an example, vertices represent people and edges describe the familiarity between two persons. Given a community composed of several persons in the social network, we want to find the leader/center of the community. Using all people in a community as the query, ANN query gets the person whose (sum) distances to all members in the community are minimum. And the person may be the leader of the community.

Another typical scenario is the *multi-example image query* [25] in the content-based image retrieval (CBIR) system. If a user wants to retrieve an image related to the concept of *sunset*, a common solution adapts the *query-by-example* style. In detail, the user is demanded to give several photos about sunset as the query. The multi-example image query gets the image which minimizes the sum of distances to all query images in the image collection.

Most recently, research in uncertain graphs [19, 23, 26] is becoming popular, because (1) uncertainty is the intrinsic property of real-world data, which is usually generated by measurement error, periodical sampling [18] and data transmission delay [6] etc; (2) data in numbers of applications are preferred to be modeled as graphs, e.g. social network [3, 19], traffic network [5, 11, 22] and protein-protein



edge	adj. vertex	possible weights
e_0	(p_0, p_1)	2.5[0.2], 3.0[0.8]
e_1	(p_0, p_2)	1.7[0.6], 1.9[0.4]
e_2	(p_0, p_3)	2.0[0.5], 2.2[0.5]
e_3	(p_1, p_2)	1.9[0.3], 2.1[0.7]
e_4	(p_1, p_3)	1.5[0.2], 1.8[0.8]
e_5	(p_1, p_4)	1.8[0.9], ∞[0.1]
e_6	(p_2, p_5)	4.5[0.4], 4.7[0.6]
e_7	(p_3, p_4)	4.0[0.5], 4.2[0.5]
e_8	(p_4, p_5)	2.5[0.8], ∞[0.2]

Figure 1 Aggregate nearest neighbor query on a undirected, weighted graph $\mathcal{G}$. $p_0 \sim p_5$ are vertices of $\mathcal{G}$. The weight of each edge is represented by a series of possible values with probabilities (in form of *value[probability]*). Especially, *weight* $= \infty$ indicates the edge will be absent sometimes. The default aggregate function is max. Vertices p_2 and p_3 are selected as the ANN query vertices (drawn with *solid circle*)

interactions (PPI) network [2]. Take the scenario of anti-pirate missions as a concrete example. The nautical charts can be modeled as an uncertain graph as shown in Figure 1. Locations such as anchorages and harbors are vertices ($p_0 \sim p_5$) and sea lanes connecting locations are represented as edges ($e_0 \sim e_8$). The edge weight w_i reveals the time cost of traveling through the sea lane e_i. Usually, variances on weather condition and sea condition make the time costs uncertain. We classify this kind of uncertainty as *weight uncertainty* in which the weighted value associated with an edge is non-deterministic. Weight uncertainty affects distances among vertices, but cannot change the topological structure of a graph. To describe the weight uncertainty, we model the weight of an edge as a random variable, rather than a single value. For example, we can have the prior knowledge that the weight of edge e_0 is 2.5 with probability 0.2 or 3.0 with probability 0.8 in the uncertain graph shown in Figure 1.[1] Notice that some sea lanes may be closed or not available sometimes because of extreme weather events. Accordingly, the corresponding edge in an uncertain graph will be absent with a certain probability, which we call *edge existence uncertainty*. Considering this kind of uncertainty, we use the notion $w_i = \infty$ to describe the random event of edge e_i being absent. For instance, one of the weight values of the edge e_5 is ∞ with probability 0.1 in Figure 1. In other words, there is no edge between vertices p_1 and p_4 with probability 0.1.

Another motivating example is that a couple of friends who live in different places in a city want to get together for a meeting. In this case, the road network can be modeled as an uncertain graph where the vertexes are places in the city and edges denote the roads connecting different places. The weight on each edge is the time cost estimated by the current traffic condition and the historical date. The estimated time cost may not be accurate and can be modeled as a random variable. In general, the uncertain graph model is an effective tool to describe highly dynamic networks (e.g. Web and social networks [14]) whose links between vertices change rapidly over time.

The two sources of uncertainty (i.e. weight and edge existence uncertainty) challenge the traditional ANN query processing approaches on both practicality and effectiveness. To explain the influence of uncertainty on ANN query, let us go back to the anti-pirate example. In the escorting mission, a warship always escorts several merchantmen or tankers simultaneously. And the escorted ships may be located in different locations, e.g. p_2 and p_3 in Figure 1. In order to protect merchantmen and tankers against being hijacked, the warship prefers to move to a location where it can rush to any escorted ship as soon as possible. The selection of mooring location is a typical application of ANN query. However, the traditional ANN definition cannot apply to the uncertain graph directly, because the uncertainty on graph structure leads to a variation on ANN result as shown in Example 1.

Example 1 As shown in Figure 2, $I_{\mathcal{G}}^0$ and $I_{\mathcal{G}}^1$ are two certain graphs derived from the uncertain graph $\mathcal{G}$ given in Figure 1. Weights of all edges are determined and labeled on edges. Suppose the query vertices are $Q = \{q_0, q_1\} = \{p_2, p_3\}$. We use the shortest

[1] Following the previous work [1, 10, 12, 20, 21, 24], we only discuss the discrete random variants (DRV) case in this paper because of the following reasons: (1) The probability density function (PDF) of real-world data can hardly be captured accurately; (2) The real-world uncertain data do not always follow any existence probability distributions, e.g. Gaussian distribution. (3) Sampling is often utilized to obtain the distribution of random data in reality, which discretizes continuous random variables.

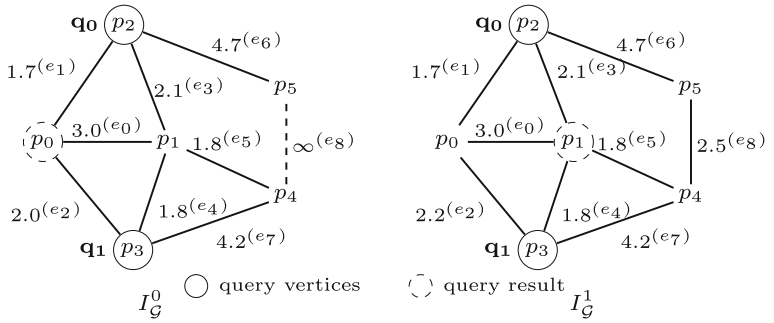


Figure 2 Two certain graphs ($I^0_{\mathcal{G}}$ and $I^1_{\mathcal{G}}$) derived from the uncertain graph $\mathcal{G}$. The aggregate function is max. Taking $\{p_2, p_3\}$ as the query set Q, the ANN vertex in each certain graph is drwan with *dashed circle*

path distance between two vertices as the distance and the aggregate function is max. In $I^0_{\mathcal{G}}$, the aggregate distance between p_0 and Q ($\max\{1.7, 2.0\} = 2.0$) is smaller than the aggregate distances of other vertices. Therefore, the ANN result is p_0 (drawn with dashed circle). In $I^1_{\mathcal{G}}$, the weights of e_2 and e_8 changes to 2.2 and 2.5, respectively. Thus, the ANN query on $I^1_{\mathcal{G}}$ retrieves p_1, because the aggregate distance between p_1 and Q ($\max\{1.8, 2.1\} = 2.1$) is the smallest one in those of all vertices.

The above example illustrates the uncertainty of the ANN results caused by the weight uncertainty in a graph. Previous work on ANN query processing, which handle certain graphs [22], certain multi-dimensional data [17] and uncertain multi-dimensional data [13], cannot work on uncertain graphs directly. This situation motivates us to exploit the semantics of *aggregate nearest neighbor query in uncertain graphs* (UG-ANN) and the corresponding query processing approach for answering the UG-ANN efficiently.

Contribution We focus on the aggregate nearest neighbor query in uncertain graphs in this paper. To the best of our knowledge, this is the first work addressing the aggregate nearest neighbor query in uncertain graphs. The primary contributions of this paper are summarized as follows.

- We formalize the problem of aggregate nearest neighbor query in uncertain graphs and propose a novel aggregate nearest neighbor query, i.e. MOST-PROB UG-ANN, and its two variants, i.e. TOP-k UG-ANN and THRESHOLD UG-ANN (Section 2).
- We propose a basic UG-ANN query framework which enables pruning methods to improve the efficiency of query processing (Section 3.1).
- We develop a structural pruning method, removing vertices and edges which are not involved in the UG-ANN query from the original uncertain graph (Section 3.2). The efficiency of UG-ANN query is benefited from the reduction of graph size.
- We present an instance pruning method which decreases the number of instances in the searching tree (Section 3.3).

– We conduct comprehensive experiments on real datasets. Experimental results show the optimized query algorithms with pruning methods significantly improve the efficiency of UG-ANN query processing (Section 4).

Organization The rest of this paper is organized as follows. We give some basic concepts and formalize the problem of aggregate nearest neighbor query in uncertain graphs in Section 2. In Section 3, we propose the UG-ANN query processing approaches, including the brute-force algorithm and the structural and instance pruning methods. Experimental results are discussed in Section 4. We review the related work in Section 5. Finally, we conclude this paper in Section 6.

2 Preliminaries

In this section, we give some concepts and definitions concerning about the uncertain graph. Firstly, we propose a novel *uncertain graph model* which is able to reveal both weight uncertainty and edge existence uncertainty. Secondly, the concept of *uncertain graph instance* is given to bridge the gap between uncertain and certain graphs. Thirdly, we discuss the *distance metrics* in uncertain graph instances, i.e. shortest path distance and aggregate path distance. Finally, we formalize the problem of aggregate nearest neighbor query in uncertain graphs.

2.1 Uncertain graph

Uncertain graph model The uncertain graph, denoted as $\mathcal{G}$, considered in this paper is an undirected, weighted graph, which is defined as a 3-tuple,

$$\mathcal{G} := (V,\ E,\ \Omega)\ , \tag{1}$$

where V is the set of vertices, $V = \{v_1, v_2, \ldots, v_n\}$, $|V| = n$. E is the set of edges, $E = \{e_1, e_2, \ldots, e_m\}$, $|E| = m$. Ω is the function mapping an edge to a random variable, $\Omega : E \to RV_c$, where RV_c is the set of random variables. For convenience, $\Omega(e_i)$ is also denoted as ω_i. The notation $\omega_i = x_j$ ($x_j \in \mathbb{R}$ and $j = 1, 2, \ldots, s$) denotes the weight of edge e_i equals to x_j, reflecting the weight uncertainty. To describe the existence uncertainty of edges, we use the notion $\omega_i = \infty$ to represent the random event that *edge e_i is absent*.

An uncertain graph can derive a series of *certain* graphs, or we call *possible instances of uncertain graph*, by determining weights on all edges. Please note that the *possible instance of uncertain graph* we used in this paper is the same as the *possible world* [7].

Possible instance of uncertain graph Given an uncertain graph $\mathcal{G}(V,\ E,\ \Omega)$, a *possible instance of uncertain graph* (or *instance* for short), denoted as $I_{\mathcal{G}}$, is defined as a 3-tuple,

$$I_{\mathcal{G}} := (V,\ E,\ W)\ , \tag{2}$$

where V, E are the same as those of uncertain graphs. W is the weight function, $W : E \to \mathbb{R}$. For simplicity, $W(e_i)$ is also denoted as w_i. $I_{\} \in \mathcal{G}$ represents the instance $I_{\mathcal{G}}$ is derived from the uncertain graph $\mathcal{G}$.

For each instance $I_{\mathcal{G}} \in \mathcal{G}$, there is an existence probability associated with $I_{\mathcal{G}}$. We use the notion $\exists\ I_{\mathcal{G}}$ to refer to the event that $I_{\mathcal{G}}$ *exists among all instances derived from* $\mathcal{G}$. Therefore, $Pr(\exists\ I_{\mathcal{G}})$ denotes the existence probability of $I_{\mathcal{G}}$, $Pr(\exists\ I_{\mathcal{G}}) \in [0, 1]$. For convenience, we also use $Pr(I_{\mathcal{G}})$ to represent the existence probability of $I_{\mathcal{G}}$ in the rest of this paper. The existence probability of $I_{\mathcal{G}}$ is calculated as

$$Pr(I_{\mathcal{G}}) = \prod_{i=1}^{|E|} Pr(\omega_i = w_i)\ , \tag{3}$$

where $\omega_i = w_i$ represents the random event that *the weight of* e_i *equals* w_i. We can enumerate all instances if the weight of edge is a *discrete* random variable. Suppose $I^1_{\mathcal{G}}, I^2_{\mathcal{G}}, \ldots, I^c_{\mathcal{G}}$ represent all instances derived from $\mathcal{G}$, where c is the number of instances derived from $\mathcal{G}$. Therefore, $\sum_{i=1}^{c} Pr(I^i_{\mathcal{G}}) = 1$ holds.

Distance metric We can adopt the well-established definition of *shortest path distance* [22] and *aggregate distance* [17] in instances derived from an uncertain graph, because these instances are certain graphs.

Given two vertices v_i and v_j in an instance $I_{\mathcal{G}}$, if there exists a path connecting v_i and v_j, the *length* of the path is calculated by summing up all weighted values of edges on the path. The *shortest path*, denoted as $v_i \rightsquigarrow v_j$, is the path whose length is minimum in all paths between v_i and v_j. Furthermore, we use $E_{v_i \rightsquigarrow v_j}(I_{\mathcal{G}})$ to denote the set of edges on $v_i \rightsquigarrow v_j$ in $I_{\mathcal{G}}$. Then, the **shortest path distance** between v_i and v_j in an instance $(I_{\mathcal{G}})$ is defined as the length of the shortest path between v_i and v_j, denoted as $d_{v_i \rightsquigarrow v_j}(I_{\mathcal{G}})$. Especially, $d_{v_i \rightsquigarrow v_j}(I_{\mathcal{G}}) = \infty$, if there is no path connecting v_i and v_j in $I_{\mathcal{G}}$.

Definition 1 (Aggregate path distance) Given an instance $I_{\mathcal{G}} \in \mathcal{G}$, a set of query vertices $Q \subseteq V$ and a vertex $v \in V$, the **aggregate path distance** of v, denoted as $d^{\alpha}(v, Q, I_{\mathcal{G}})$, is defined as the aggregation (e.g. sum, max, min[2]) of the shortest path distances between v and all vertices in Q. Formally,

$$d^{\alpha}(v, Q, I_{\mathcal{G}}) = \begin{cases} \sum\limits_{q \in Q} d_{v \rightsquigarrow q}(I_{\mathcal{G}}) & (\text{sum})\ , \\ \max\limits_{q \in Q} d_{v \rightsquigarrow q}(I_{\mathcal{G}}) & (\text{max})\ , \\ \min\limits_{q \in Q} d_{v \rightsquigarrow q}(I_{\mathcal{G}}) & (\text{min})\ . \end{cases} \tag{4}$$

Specifically, $d^{\alpha}(v, Q, I_{\mathcal{G}}) = \infty$ with the aggregation function *max* or *sum* iff $\exists q \in Q$ such that there does not exist a path from v to q in $I_{\mathcal{G}}$. On the contrary, the aggregate distance is always 0 when the aggregate function is *min*.

Each edge on the shortest paths from a vertex v to the query Q in $I_{\mathcal{G}}$ is called an *on-path edge* in $I_{\mathcal{G}}$. We use $E^{\alpha}_{v}(I_{\mathcal{G}}, Q)$ to denote the set of all on-path edges from v to Q in $I_{\mathcal{G}}$, i.e. $E^{\alpha}_{v}(I_{\mathcal{G}}, Q) = \bigcup_{q \in Q} E_{v \rightsquigarrow q}(I_{\mathcal{G}})$.

[2]We do not detail the ANN query processing with the aggregate function *min* in this paper, because any query vertex $q \in Q$ is trivially an ANN vertex with the aggregate distance $d^{\alpha}(q, Q, I_{\mathcal{G}}) = 0$.

Example 2 Consider $I^0_{\mathcal{G}}$ (ref. to Figure 2) in Example 1. The shortest path from p_4 to p_3 is e_5e_4, $E_{p_4 \rightsquigarrow p_3}(I^0_{\mathcal{G}}) = \{e_4, e_5\}$ and $d_{p_4 \rightsquigarrow p_3}(I^0_{\mathcal{G}}) = 1.8 + 1.8 = 3.6$. Taking $\{p_2, p_3\}$ as query vertices Q, the aggregate path distance (max) from p_4 to Q is $d^{\alpha}(p_4, Q, I^0_{\mathcal{G}}) = \max\{d_{p_4 \rightsquigarrow p_2}(I^0_{\mathcal{G}}),\ d_{p_4 \rightsquigarrow p_3}(I^0_{\mathcal{G}})\} = \max\{3.9,\ 3.6\} = 3.9$. $E^{\alpha}_{p_4}(I^0_{\mathcal{G}}, Q) = \bigcup_{q \in \{p_2, p_3\}} E_{p_4 \rightsquigarrow q}(I^0_{\mathcal{G}}) = \{e_3, e_5\} \bigcup \{e_4, e_5\} = \{e_3, e_4, e_5\}$.

2.2 Problem definition

In this subsection, we formalize the problem of UG-ANN and give a practical form of the UG-ANN query. First of all, based on the uncertain graph model and the aggregate path distance, we introduce the aggregate nearest neighbor query to instances.

Definition 2 (ANN in certain graph) Given an uncertain graph $\mathcal{G}$, an instance $I_{\mathcal{G}} \in \mathcal{G}$, a set of query vertices $Q \subseteq V$, the vertex $v \in V$ is defined as the **aggregate nearest neighbor (ANN)** w.r.t. Q in $I_{\mathcal{G}}$ iff $\forall v' \in V \setminus \{v\},\ d^{\alpha}(v, Q, I_{\mathcal{G}}) \leq d^{\alpha}(v', Q, I_{\mathcal{G}})$.

The set of all aggregate nearest neighbor vertices w.r.t. Q in $I_{\mathcal{G}}$ is denoted as $ANN(I_{\mathcal{G}}, Q)$. Formally, $ANN(I_{\mathcal{G}}, Q) := \{v \mid \forall v' \in V \setminus \{v\},\ d^{\alpha}(v, Q, I_{\mathcal{G}}) \leq d^{\alpha}(v', Q, I_{\mathcal{G}})\}$.

The ANN results may vary w.r.t. the same query Q in different instances derived from $\mathcal{G}$, as shown in Example 1. For a vertex v which is the ANN vertex in $I_{\mathcal{G}}$, it has a chance to be selected as an ANN vertex in $\mathcal{G}$. In other words, the probability of v being an ANN vertex in $\mathcal{G}$ is greater than 0. Therefore, we define a vertex v as the ANN in uncertain graph w.r.t. a query Q if v is the ANN vertex w.r.t. Q in at least one instance derived from $\mathcal{G}$.

Definition 3 (ANN in uncertain graph) Given an uncertain graph $\mathcal{G}$, a set of query vertices $Q \subseteq V$, a vertex $v \in V$ is defined as the **aggregate nearest neighbor in uncertain graph (UG-ANN)** w.r.t. Q in $\mathcal{G}$ iff $\exists\ I_{\mathcal{G}} \in \mathcal{G}$ such that $v \in \text{ANN}(I_{\mathcal{G}}, Q)$.

We use $UG\text{-}ANN(\mathcal{G}, Q)$ to denote the set of all UG-ANN w.r.t. Q in $\mathcal{G}$. Formally, $UG\text{-}ANN(\mathcal{G}, Q) := \bigcup_{I_{\mathcal{G}} \in \mathcal{G}} ANN(I_{\mathcal{G}}, Q)$. Consider $\mathcal{G}$ in Figure 1, we have $\{p_0, p_1\} \subseteq UG\text{-}ANN(\mathcal{G}, Q)$, because $p_0 \in ANN(I^0_{\mathcal{G}}, Q)$ and $p_1 \in ANN(I^1_{\mathcal{G}}, Q)$ as shown in Example 1. The event "*v is a UG-ANN vertex w.r.t. Q in $\mathcal{G}$*" is a probability event, because the weights of edges are random variables. We use the notion $v \in UG\text{-}ANN(\mathcal{G}, Q)$ to represent the above event. The probability of v being a UG-ANN vertex (or we call the *ANN probability* of v) is denoted as $Pr(v \in UG\text{-}ANN(\mathcal{G}, Q))$.

Notice that, sometimes, the number of UG-ANN vertices is large and the ANN probabilities of some UG-ANN vertices may be very low. However, users may be more interested in or have to choose one vertex as the result in some scenarios. The vertex is expected to have the largest ANN probability. Therefore, we propose the Most-Prob UG-ANN query.

Definition 4 (Most-Prob UG-ANN) Given an uncertain graph $\mathcal{G}$, a set of query vertices $Q \subseteq V$, the **Most-Probability UG-ANN** query, denoted as Most-Prob UG-ANN, finds the vertex $v \in V$ with the largest ANN probability.

Calculation of ANN probabilities is different according to the types of random variables we used to model edge weights in uncertain graph, i.e. continuous random variable (CRV) and discrete random variable (DRV).

In the DRV case, the ANN probability of v is calculated as the sum of the existence probabilities of instances in which v is the ANN vertex. Formally,

$$Pr(v \dashv \text{UG-ANN}(\mathcal{G}, Q)) = \sum_{i=1}^{c} (Pr(I_{\mathcal{G}}^{i}) \mid v \in ANN(I_{\mathcal{G}}^{i}, Q)) , \tag{5}$$

where c is the number of instances derived from $\mathcal{G}$. The following example illustrates how the Most-Prob UG-ANN query works.

Example 3 Consider the uncertain graph $\mathcal{G}$ shown in Figure 1. Query vertices $Q = \{p_2, p_3\}$ and the aggregate function is max. After enumerating all instances derived from $\mathcal{G}$ and accumulating the probabilities of ANN vertices, we get the ANN probabilities of all vertices in $\mathcal{G}$ as shown in Table 1. $Pr(p_1 \dashv \text{UG-ANN}(\mathcal{G}, Q)) = 0.65$ and $Pr(p_0 \dashv \text{UG-ANN}(\mathcal{G}, Q)) = 0.35$. Therefore, the result of Most-Prob UG-ANN query w.r.t. Q in $\mathcal{G}$ is p_1.

2.3 Variants of UG-ANN queries

Except for the Most-Prob UG-ANN query, we propose two variants of UG-ANN queries, i.e. *top-k UG-ANN* query and *threshold UG-ANN* query, to meet the need of real-world applications. Please note that the algorithms proposed in this paper can answer all of the above UG-ANN queries and we just give the version of algorithms processing the Most-Prob UG-ANN query.

Definition 5 (Top-k UG-ANN) Given an uncertain graph $\mathcal{G}$, a set of query vertices $Q \subseteq V$ and a user preferred integer k, the **top-k UG-ANN query**, denoted as k-UG-ANN, finds k vertices $\{v_{s_1}, v_{s_2}, \ldots, v_{s_k}\} \subseteq V$ with the largest ANN probabilities, $0 \leq s_1, s_2, \ldots, s_k < |V|$.

The set of all top-k UG-ANN vertices is represented by k-UG-ANN$(\mathcal{G}, Q)$. Formally, $\forall v' \in V \setminus k\text{-UG-ANN}(\mathcal{G}, Q)$ and $\forall v \in k\text{-UG-ANN}(\mathcal{G}, Q)$, $Pr(v \dashv \text{UG-ANN}(\mathcal{G})) \geq Pr(v' \dashv \text{UG-ANN}(\mathcal{G}))$. The following example illustrates the top-k UG-ANN query on an uncertain graph with $k = 2$.

Example 4 Consider the uncertain graph $\mathcal{G}$ shown in Figure 1. p_2 and p_3 are selected as query vertices and the aggregate function is max. After enumerating all instances derived from $\mathcal{G}$ and accumulating the probabilities of ANN vertices, we get the ANN probabilities of all vertices in $\mathcal{G}$ as shown in Table 1. Top-k UG-ANN query retrieves k vertices which have the largest ANN probability. In this case of $k = 2$, p_0

Table 1 ANN probabilities of vertices in $\mathcal{G}$ (ref. to Figure 1)

Vertex	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
ANN prob.	0.35	0.65	0	0	0	0	0	0	0

and p_1 have the largest ANN probabilities. Specifically, $Pr(p_1 \dashv \text{UG-ANN}(\mathcal{G}, Q)) = 0.65$ and $Pr(p_0 \dashv \text{UG-ANN}(\mathcal{G}, Q)) = 0.35$. Therefore, the result of top-2 UG-ANN query w.r.t. Q in $\mathcal{G}$ is $\{p_0, p_1\}$.

An alternative ANN query requirement is that the user wants to get the ANN vertices with a preferred confidence. Thus, we develop the threshold UG-ANN query to enable user to assign a probability threshold for ANN query.

Definition 6 (Threshold UG-ANN) Given an uncertain graph $\mathcal{G}$, a set of query vertices $Q \subseteq V$ and a user preferred threshold ϵ ($0 < \epsilon \leq 1$), the **threshold UG-ANN query**, denoted as ϵ-UG-ANN, finds all vertices $v \in Q$ whose ANN probability is larger than or equal to ϵ. Formally,

$$\epsilon\text{-UG-ANN}(\mathcal{G}, Q) := \{v \mid Pr(v \dashv \text{UG-ANN}(\mathcal{G}, Q)) \geq \epsilon\}. \tag{6}$$

Again, we use the uncertain graph $\mathcal{G}$ shown in Figure 1 to illustrate the probabilistic threshold UG-ANN query.

Example 5 Consider the uncertain graph $\mathcal{G}$ and query vertices Q depicted in Figure 1. The aggregate function is max. The user preferred threshold $\epsilon = 0.5$. According to Table 1, ϵ-UG-ANN$(\mathcal{G}, Q) = \{p_1\}$, because $Pr(p_1 \dashv \text{UG-ANN}(\mathcal{G}, Q)) = 0.65 > 0.5$.

3 UG-ANN query processing

In this section, we present the UG-ANN query processing algorithms. First of all, the brute-force algorithm for UG-ANN query, called BF-UG-ANN, is given. After that, a *structural pruning* algorithm is proposed to reduce the computational costs of enumerating uncertain graph instances. Besides, a UG-ANN candidate filtering is used during the structural pruning. Finally, an instance pruning algorithm is proposed to remove instances in which ANN queries are not necessary to be executed. Table 2 summarizes the frequent used notations in the remainder of this paper.

Table 2 Frequent used notations

Symbol	Description
$\mathcal{G}$	The uncertain graph
$I_{\mathcal{G}}$	A possible instance of uncertain graph $\mathcal{G}$
ω_i	The random variable representing the weight of e_i in $\mathcal{G}$
w_i	The deterministic weight of e_i in $I_{\mathcal{G}}$
$\text{ANN}(I_{\mathcal{G}}, Q)$	The set of ANN vertices in $I_{\mathcal{G}}$ w.r.t. the query Q
$\text{UG-ANN}(\mathcal{G}, Q)$	The set of UG-ANN vertices w.r.t. Q in $\mathcal{G}$
$I_{\mathcal{G}}^{\min}$, $I_{\mathcal{G}}^{\max}$	The min-weighted (max-weighted) instance
$D_{\min}^{\alpha}$, $D_{\max}^{\alpha}$	The minimal (maximal) aggregate distance
$v \rightsquigarrow u$	The shortest path from vertex v to vertex u
$d_{v \rightsquigarrow u}(I_{\mathcal{G}})$	The shortest path distance between v and u in $I_{\mathcal{G}}$
$d^{\alpha}(v, Q, I_{\mathcal{G}})$	The aggregate distance from v to Q in $I_{\mathcal{G}}$

3.1 Brute-force algorithm

A brute-force approach (called BF-UG-ANN) of processing the UG-ANN query is to enumerate all instances derived from an uncertain graph $\mathcal{G}$. For each instance $I_{\mathcal{G}}$, we find the ANN vertices v_{ANN}, calculate the existence probability $Pr(I_{\mathcal{G}})$ according to (5), and add the existence probability to the ANN probabilities of v_{ANN}. Detailed steps of the BF-UG-ANN algorithm are given in Algorithm 1.

Algorithm 1 BF-UG-ANN($\mathcal{G}$, Q)

Input : $\mathcal{G}$ — the uncertain graph (V, E, W);
Q — the query vertices set;
Output : $V_{\text{UG-ANN}}$ — the UG-ANN vertices set;

1 $Pr_{1\ldots|V|} \leftarrow \{0, 0, \ldots, 0\}$;
2 **for each** instance $I_{\mathcal{G}} \in \mathcal{G}$ **do**
3 $\quad Pr(I_{\mathcal{G}}) \leftarrow$ Calculate the existence probability of $I_{\mathcal{G}}$;
4 $\quad$ **for each** vertex $v \in V$ **do**
5 $\quad\quad d^{\alpha}(v, Q, I_{\mathcal{G}}) \leftarrow$ Calculate the aggregate distance between v and Q in $I_{\mathcal{G}}$ using the Dijkstra's algorithm [8];
6 $\quad v_k \leftarrow \arg\min_{v \in V} d^{\alpha}(v, Q, I_{\mathcal{G}})$; /* Find the ANN vertices in $I_{\mathcal{G}}$ */
7 $\quad Pr_k \leftarrow Pr_k + Pr(I_{\mathcal{G}})$;
8 $V_{\text{UG-ANN}} \leftarrow$ Return vertices with the largest ANN probability;

Firstly, the ANN probabilities of all vertices in $\mathcal{G}$ are set to 0 (line 1 in Algorithm 1). And then, all instances derived from $\mathcal{G}$ are enumerated to calculate the ANN vertex and the ANN probability (line 2–7). In detail, for each instance $I_{\mathcal{G}}$, the existence probability $Pr(I_{\mathcal{G}})$ is calculated according to (3) (line 3). And the Dijkstra's algorithm [8] based ANN query approach is utilized for searching the ANN vertex, say v (line 5). We then increase the ANN probability of v by $Pr(I_{\mathcal{G}})$ (line 7). Finally, the probability of all vertices is obtained and we can get the vertex with the largest ANN probability (line 8).

The computational cost of the BF-UG-ANN algorithm is expensive, because all instances derived from $\mathcal{G}$ have to be enumerated. Specifically, the number of instances is $\mathcal{O}(S^{|E|})$, where S is the maximal number of distinct weight values on an edge. For each instance, it takes $\mathcal{O}(|V|^2 \lg |V| + |V| \cdot |E|)$ time to find the ANN vertex, if the Fibonacci heap [9] is used for computing the aggregation distance. Therefore, the time complexity of the BF-UG-ANN algorithm is $\mathcal{O}(S^{|E|}(|V|^2 \lg |V| + |V| \cdot |E|))$.

3.2 Structural pruning

3.2.1 Subgraph extraction

The high time cost of BF-UG-ANN is caused by instance enumeration. The number of instances increases exponentially with the number of edges. To improve the efficiency of UG-ANN query processing, an effective optimization method is subgraph extraction, i.e. removing edges and vertices which are *far* from the query vertices.

The first problem is to decide whether removing a vertex or an edge will affect the UG-ANN query result. Assume the uncertain graph $\mathcal{G}'$ is generated by removing

a vertex v (or an edge e) from an uncertain graph $\mathcal{G}$. If the UG-ANN vertices and their ANN probabilities in $\mathcal{G}$ are the same as those in $\mathcal{G}'$, the query result cannot be affected and the vertex/edge pruning operation is safe. Notice that the paths from the ANN vertex to each query vertex construct a tree which we call an *ANN path tree*. The root of the ANN path tree is the ANN vertex in the instance. A vertex (or an edge) will not affect the UG-ANN query result if it is not in any ANN path trees of an instance. In other words, given an uncertain graph $\mathcal{G}$ and a set of query vertices Q, we can remove a vertex v (or an edge e) from $\mathcal{G}$ without affecting the UG-ANN query result if $\nexists I \in \mathcal{G}$ such that v (or e) is in the ANN path tree of I.

The above rule provides a solution of pruning vertices and edges. However, given a vertex v or an edge e, it is still difficult to determine whether we can remove it safely without enumerating all instances derived from $\mathcal{G}$. Fortunately, some criteria are helpful to identify some edges and vertices which can be removed safely. First of all, we introduce some lemmas to help pave the way for addressing the vertex and edge pruning criteria.

Lemma 1 (Monotonicity of aggregate distance) *Given an uncertain graph $\mathcal{G}$ and an instance $I_\mathcal{G} \in \mathcal{G}$, $I'_\mathcal{G}$ is an instance generated from $I_\mathcal{G}$ by removing one edge or increasing the weight value of one edge. The aggregate distance of the vertex v in $I_\mathcal{G}$ is not greater than that of $I'_\mathcal{G}$. Formally, $d^\alpha(v, Q, I_\mathcal{G}) \leq d^\alpha(v, Q, I'_\mathcal{G})$.*

Proof If a vertex v is not connected with all query vertices in the instance $I_\mathcal{G}$, it cannot reach the query vertex in the instance $I'_\mathcal{G}$. Thus, $d^\alpha(v, Q, I_\mathcal{G}) = d^\alpha(v, Q, I'_\mathcal{G}) = \infty$. Otherwise, there exist paths between vertex v and each query vertex $q \in Q$. For an edge $e \in E$ in $I_\mathcal{G}$, there are two conditions: (i) e is not on the aggregate path, and (ii) e is on the paths between v and Q.

In case (i), removing e or increasing the weight of e will not affect the aggregate path from v to Q. Thus, we have $d^\alpha(v, Q, I_\mathcal{G}) = d^\alpha(v, Q, I'_\mathcal{G})$.

In case (ii), if $I'_\mathcal{G}$ is obtained by removing edge e from $I_\mathcal{G}$, the original aggregate path will disconnect. Without loss of generality, we assume e is on the shortest path between v and the query vertex $q \in Q$, denoted as $v \rightsquigarrow q$, in $I_\mathcal{G}$. In $I'_\mathcal{G}$, there are two relations between v and q, i.e., (ii-a) v and q are disconnected and (ii-b) there is a path between v and q.

- In case (ii-a), the new aggregate distance of v is $d^\alpha(v, Q, I'_\mathcal{G}) = \infty$, thus $d^\alpha(v, Q, I_\mathcal{G}) < d^\alpha(v, Q, I'_\mathcal{G})$.
- In case (ii-b), the shortest path between v and q in $I'_\mathcal{G}$ is present in $I_\mathcal{G}$. The inequity $d_{v \rightsquigarrow q}(I_\mathcal{G}) \leq d_{v \rightsquigarrow q}(I'_\mathcal{G})$ holds, otherwise $v \rightsquigarrow q$ is not the shortest path in $I_\mathcal{G}$.

As all cases mentioned above, the lemma is immediate. □

Lemma 1 reveals the fact that, for a vertex v, the aggregate distance between v and the query vertices increases monotonically when the edge weight in the uncertain graph increases.

Among all instances derived from $\mathcal{G}$, an instance is called a *min-weighted instance*, denoted as $I^{\min}_\mathcal{G}$, if the weight of each edge e is the minimum among all possible values. Formally, $\forall I_\mathcal{G}(V, E, W) \neq I^{\min}_\mathcal{G}(V, E, W')$ and $\forall e_i \in E$, $w'_i \leq w_i$ holds. Similarly, an

instance is called *max-weighted instance*, denoted as $I_{\mathcal{G}}^{\max}$, if the weight of each edge e is the maximum among all possible values. Figure 3 illustrates the min-weighted instance and max-weighted instance derived from $\mathcal{G}$. For a vertex v, the aggregate distances between v and Q in all instances can be bounded by the aggregate distances in the min-weighted instance and the max-weighted instance.

Lemma 2 *For each vertex v, the aggregate distance w.r.t. Q in any instance $I_{\mathcal{G}}$, denoted as $d^{\alpha}(v, Q, I_{\mathcal{G}})$, is bounded by the minimum aggregate distance $d^{\alpha}(v, Q, I_{\mathcal{G}}^{\min})$ and the maximum aggregate distance $d^{\alpha}(v, Q, I_{\mathcal{G}}^{\max})$. Formally,*

$$d^{\alpha}(v, Q, I_{\mathcal{G}}^{\min}) \leq d^{\alpha}(v, Q, I_{\mathcal{G}}) \leq d^{\alpha}(v, Q, I_{\mathcal{G}}^{\max}) . \tag{7}$$

The aggregate distances of the ANN vertex in $I_{\mathcal{G}}^{\min}$ and $I_{\mathcal{G}}^{\max}$ are denoted as the *minimal ANN distance* ($D_{\min}^{\alpha}$) and *maximal ANN distance* ($D_{\max}^{\alpha}$) respectively. We have the following lemma.

Lemma 3 *Given an instance $I_{\mathcal{G}} \in \mathcal{G}$ and a set of query vertices Q, the aggregate distance of the ANN vertex v_{ANN} w.r.t. Q is bounded by $D_{\min}^{\alpha}$ and $D_{\max}^{\alpha}$. Formally,*

$$D_{\min}^{\alpha} \leq d^{\alpha}(v_{ANN}, Q, I_{\mathcal{G}}) \leq D_{\max}^{\alpha} . \tag{8}$$

Based on the above lemmas, we can obtain the criteria for identifying removable vertices and edges.

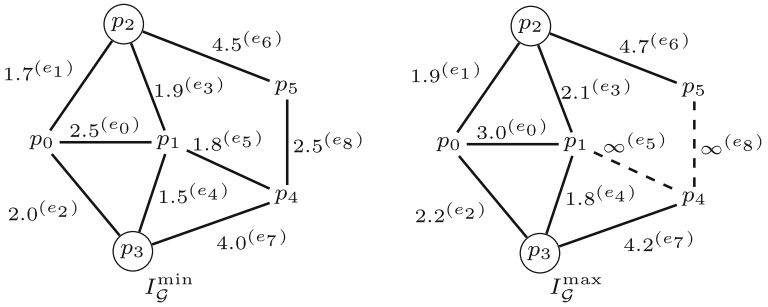


Table: Aggregate distances of vertices in $I_{\mathcal{G}}^{\min}$ and $I_{\mathcal{G}}^{\max}$

$\mathbf{d}^{\alpha}$	p_0	p_1	p_2	p_3	p_4	p_5
$\mathbf{I}_{\mathcal{G}}^{\max}$	2.2	2.1*	3.9	3.9	8.1	8.6
$\mathbf{I}_{\mathcal{G}}^{\min}$	2.0	1.9	3.4	3.4	3.7	5.8

Figure 3 An example of min-weighted instance (*top left*) and max-weighted instance (*top right*). The aggregate function is max and aggregate distances between each vertex and query vertices $\{p_2, p_3\}$ (drawn with *solid circle*) are listed in the table (*bottom*). The maximal ANN distance $D_{\max}^{\alpha}$ is 2.1 (*marked with asterisk*)

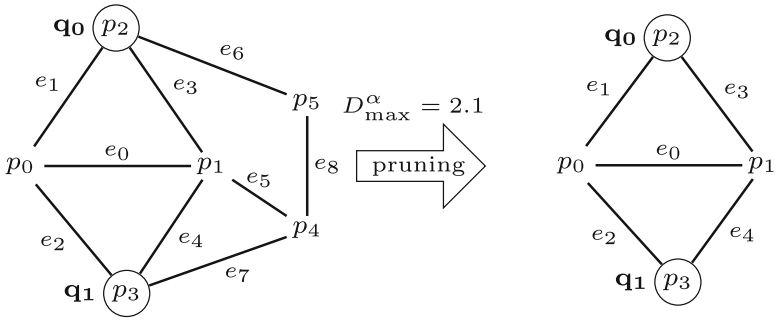
Theorem 1 *Given an uncertain graph $\mathcal{G}$ and a query vertex set $Q \subseteq V$, a vertex $v \in V$ can be removed safely if $\exists\, q \in Q$ such that the shortest path distance from v to q in $I_{\mathcal{G}}^{\min}$ is larger than $D_{\max}^{\alpha}$, i.e., $d_{v \rightsquigarrow q}(I_{\mathcal{G}}^{\min}) > D_{\max}^{\alpha}$.*

Proof We prove the theorem by contradiction. Assume a vertex v cannot be removed safely and $\exists\, q \in Q$ such that $d_{v \rightsquigarrow q}(I_{\mathcal{G}}^{\min}) > D_{\max}^{\alpha}$. Therefore, $\exists\, I_{\mathcal{G}} \in \mathcal{G}$ such that v is on the ANN path in $I_{\mathcal{G}}$. The ANN vertex in $I_{\mathcal{G}}$ is denoted as v'. And $\exists\, q \in Q$ such that v is on the shortest path from v' to q, which indicates that $d_{v' \rightsquigarrow q}(I_{\mathcal{G}}) \geq d_{v \rightsquigarrow q}(I_{\mathcal{G}})$ holds. Thus, we have $d_{v' \rightsquigarrow q}(I_{\mathcal{G}}) \geq d_{v \rightsquigarrow q}(I_{\mathcal{G}}) \geq d_{v \rightsquigarrow q}(I_{\mathcal{G}}^{\min}) > D_{\max}^{\alpha}$. According to the definition of aggregate distance, $d^{\alpha}(v', Q, I_{\mathcal{G}}) = \max_{q' \in Q}(d_{v' \rightsquigarrow q'}(I_{\mathcal{G}})) \geq d_{v' \rightsquigarrow q}(I_{\mathcal{G}})$. So $d^{\alpha}(v', Q, I_{\mathcal{G}}) > D_{\max}^{\alpha}$ and vertex v' is not an ANN vertex in $I_{\mathcal{G}}$ (Lemma 3), which is contradict to the assumption. □

Corollary 1 *An edge $e \in E$ can be removed safely if at least one of its adjacent vertices can be removed safely.*

Example 6 illustrates how to prune vertices and edges based on Theorem 1 and Corollary 1.

Example 6 Consider the uncertain graph $\mathcal{G}$ in Figure 1. The min-weighted instance $I_{\mathcal{G}}^{\min}$ and the max-weighted instance $I_{\mathcal{G}}^{\max}$ are shown in Figure 3. And we get the maximum ANN distance $D_{\max}^{\alpha} = 2.1$. After that, we calculate the shortest path distances between query vertices and all vertices as illustrated in Figure 4. For $p_0 \sim p_2$, the shortest path distances to query vertex p_2 are 1.7, 1.9 and 0, which are smaller than $D_{\max}^{\alpha}$. Thus, $p_0 \sim p_2$ cannot be pruned. Besides, we are not able to prune p_3 because $d_{p_3 \rightsquigarrow p_3} = 0 < D_{\max}^{\alpha}$. p_4 can be pruned safely because $d_{p_4 \rightsquigarrow p_2} = 3.7 > D_{\max}^{\alpha}$ and $d_{p_4 \rightsquigarrow p_3} = 3.3 > D_{\max}^{\alpha}$. Similar to p_4, p_5 should also be pruned. After removing



$\mathcal{G}$ **[before structural pruning]** $\mathcal{G}'$ **[after structural pruning]**

Table: Shortest path distance in $I_{\mathcal{G}}^{\min}$

	p_0	p_1	p_2	p_3	p_4	p_5
p_2	1.7	1.9	0	3.4	3.7	4.5
p_3	2.0	1.5	3.4	0	3.3	5.8

p_4, p_5 are pruned, because $\forall q \in Q$, $d_{p_4 \rightsquigarrow q}$ and $d_{p_5 \rightsquigarrow q}$ are larger than $D_{\max}^{\alpha}$.

Figure 4 An example of structural pruning. The set of query vertices $Q = \{p_2, p_3\}$ (drawn with *solid circle*) and the aggregation function is max

p_4, p_5 and their adjacent edges (i.e. $e_5 \sim e_8$), we get the subgraph $\mathcal{G}'$ as shown in Figure 4.

3.2.2 UG-ANN candidate filtering

Not all vertices in the subgraph are possible to be the ANN vertex in an instance. Because a vertex may be on the ANN path in some instances but is not an ANN vertex in any instance. Therefore, the computational costs of examining whether those vertices are ANN should be reduced. For example, in Figure 4, p_2 and p_3 in subgraph $\mathcal{G}'$ are not ANN vertices in any instance according to the ANN probabilities shown in Table 1, i.e., p_2 and p_3 are definitely not UG-ANN vertices. The following theorem is proposed for identifying candidates of UG-ANN vertices.

Theorem 2 *Given an uncertain graph $\mathcal{G}$ and a set of query vertices Q, a vertex $v \in V$ cannot be the UG-ANN vertex (i.e. $Pr(v \in UG\text{-}ANN(Q, \mathcal{G})) = 0$) if $d^\alpha(v, Q, I_{\mathcal{G}}^{\min}) > D_{\max}^\alpha$.*

Proof With the Lemma 3, this theorem is immediate. □

The above theorem requires for the exact aggregate distance of each vertex. For roughly pruning vertices which are not able to be UG-ANN, the following corollary is more practical.

Corollary 2 *Given a vertex $v \in V$ in an uncertain graph $\mathcal{G}$ and suppose $q \in Q$ is the nearest query vertex to v. The vertex v cannot be the UG-ANN vertex in any instance (i.e. $Pr(v \in UG\text{-}ANN(Q, \mathcal{G})) = 0$) if*

$$\begin{cases} d_{v \rightsquigarrow q}(I_{\mathcal{G}}) > D_{\max}^\alpha, & \text{the aggregate function is max} \\ d_{v \rightsquigarrow q}(I_{\mathcal{G}}) > \frac{D_{\max}^\alpha}{|Q|}, & \text{the aggregate function is sum} \end{cases}$$

Example 7 After structural pruning in Example 6, the pruned uncertain graph $\mathcal{G}'$ consists of 4 vertices, i.e. $p_0 \sim p_3$, (ref. to Figure 4). As illustrated in Figure 3, the minimal aggregate distances of $p_0 \sim p_3$ are 2.0, 1.9, 3.4, 3.4 respectively. Therefore, we conclude that p_2 and p_3 cannot be ANN vertices in any instance according to Theorem 2. Because $d^\alpha(p_2, Q, I_{\mathcal{G}}^{\min}) = 3.4 > D_{\max}^\alpha$ and $d^\alpha(p_3, Q, I_{\mathcal{G}}^{\min}) = 3.4 > D_{\max}^\alpha$. Now the UG-ANN candidate set is $\{p_0, p_1\}$. We only need to check the aggregation distances of p_0 and p_1 for finding the ANN vertex in all instances.

3.2.3 Structural pruning algorithm

Based on the above discussion, we propose the structural pruning algorithm, called Stru-Prun, which achieves both *subgraph extraction* and *UG-ANN candidate filtering* simultaneously. Inputs of the Stru-Prun algorithm are the uncertain graph $\mathcal{G}$ and the set of query vertices Q. Stru-Prun is based on the Dijkstra's algorithm [8], but it follows a concurrent expansion style. Detailed steps of the structural pruning algorithm is depicted in Algorithm 2.

Initially, we construct $I_{\mathcal{G}}^{\max}$ and execute the ANN query on it to find the ANN vertex v with the maximal ANN distance $D_{\max}^\alpha$ (line 1–3 in Algorithm 2). We then construct $I_{\mathcal{G}}^{\min}$ from $\mathcal{G}$ and create a priority queue S of vertices w.r.t. the shortest path

Algorithm 2 Stru-Prun($\mathcal{G}$, Q)

```
   Input  : G — the uncertain graph;
            Q — the set of query vertices;
   Output : V_c — the UG-ANN candidate vertices set;
            G' — the subgraph;
 1 Construct the max-weighted instance I_G^max from G;
 2 Find the ANN vertex v in I_G^max;
 3 Get the maximal ANN distance D_max^α ← d^α(v, Q, I_G^max);
 4 Construct the min-weighted instance I_G^min from G;
 5 Create a min-priority queue S of vertices w.r.t. shortest path distance in I_G^min;
 6 Push vertices v ∈ Q to S with v.key = 0;
 7 Push other vertices v ∈ V \ Q to S with v.key = ∞;
 8 R ← ∅;                          /* R is the set of vertices which are in the subgraph */
 9 V_c ← ∅;                        /* V_c is the set of UG-ANN candidate vertices */
10 while S ≠ ∅ do
11     v ← Pop the vertex with the minimal key from S;
12     if v.key > D_max^α then break;
13     V_s ← V_s ∪ {v};
14     if v.key ≤ D_max^α/|Q| (sum) or v.key ≤ D_max^α (max) then
15         V_c ← V_c ∪ {v};
16     for each vertex v' adjacent to v (via edge e_i) in I_G^min do
17         if v.key + w_i < v'.key then
18             v'.key = v.key + w_i;
19 Generate the induced subgraph G' from G according to V_s;
```

distance (line 4–5). The key of vertex v in S represents the shortest path distance from v to the query vertex q which has the shortest distance from v in all query vertices. The vertex with the smallest key is in front of S and should be popped first. S is initialized as empty. We push all query vertices to S with key 0 and other vertices with key ∞ (line 6–7). Besides, we create two vertices sets V_s and V_c to record vertices preserved in the subgraph and the candidate UG-ANN vertices respectively (line 8–9).

The expansion procedure from the query vertices is then invoked, which follows the style of the Dijkstra's algorithm (line 10–18). In detail, the vertex v with the minimal key is popped from S in each iteration (line 11). If the key of v is larger than $D^{\alpha}_{\max}$, both v and the remaining vertices in S should be pruned according to Theorem 1, and then the expansion is terminated (line 12). Otherwise, v is a vertex in the subgraph, so v is added to V_s (line 13) and we further check whether v is a UG-ANN candidate according to Corollary 2 (line 14–15). After that, we update the keys of vertices adjacent to v as part of the standard Dijkstra's algorithm (line 16–18). Specifically, for each vertex v' adjacent to v via edge e in $I^{\min}_{\mathcal{G}}$, the key of v' is updated to $v.key + w_i$ if $v'.key > v.key + w_i$. If $S \neq \emptyset$, the expansion continues by handling the vertex in S with the minimal key.

After the iteration, we extract the induced subgraph [4] $\mathcal{G}'$ from $\mathcal{G}$ according to the subgraph vertices V_s. The outputs of the Stru-Prun algorithm are the subgraph $\mathcal{G}'$ and the set of UG-ANN candidates V_c.

Now we discuss the complexity analysis of the Stru-Prun algorithm. The ANN query on $I_{\mathcal{G}^{\max}}$ (line 2 in Algorithm 2) takes $\mathcal{O}(|V|^2 lg|V| + |V||E|)$ time, and

expansion (line 10–18) takes $\mathcal{O}(|V|lg|V| + |E|)$ because it adopts the expansion similar with the Dijkstra's algorithm. Therefore, the time complexity of Stru-Prun is $\mathcal{O}(|V|^2 lg|V| + |V||E|)$.

3.3 Instance pruning

The structural pruning reduces the number of vertices and edges by extracting a subgraph from the original uncertain graph. Thus, the efficiency of UG-ANN query processing is improved. However, enumerating all instances derived from the uncertain subgraph is still expensive. For example, the subgraph $\mathcal{G}'$ in Figure 4 is composed of 5 edges (i.e. $e_0 \sim e_4$). There are 2 possible weight values for each edge. Therefore, the number of instances derived from $\mathcal{G}'$ is $2^5 = 32$. In other words, we have to execute ANN queries 32 times to get the UG-ANN query result. However, we observe that we do not need to check and execute ANN queries on all instances.

3.3.1 Observation

Consider the instance $I^{\min}_{\mathcal{G}'} \in \mathcal{G}'$ after structural pruning in Figure 5. The default aggregate function is max. p_1 is the ANN vertex in $I^{\min}_{\mathcal{G}'}$, $d^{\alpha}(p_0, Q, I^{\min}_{\mathcal{G}'}) = \max\{1.9, 1.5\} = 1.9$. The on-path edges from p_0 to Q is $E^{\alpha}_{p_0}(I^{\min}_{\mathcal{G}'}, Q) = \{e_3, e_4\}$ (drawn with dashed circle). The set of non-on-path edges is defined as $\tilde{E}^{\alpha}_{p_0}(I^{\min}_{\mathcal{G}'}, Q) = E \setminus E^{\alpha}_{p_0}(I^{\min}_{\mathcal{G}'}, Q) = \{e_0, e_1, e_2\}$. For instances which are derived by increasing weights of non-on-path edges (i.e., $I^1_{\mathcal{G}'} \sim I^3_{\mathcal{G}'}$, $I^6_{\mathcal{G}'} \sim I^9_{\mathcal{G}'}$ in Figure 5), the ANN result is always p_1. Theorem 3 states the invariant of ANN vertex in a group of instances.

Theorem 3 *Given a set of query vertices Q and an instance $I_{\mathcal{G}}(V, E, W)$, the ANN vertex is p and the on-path edges from p to Q is $E^{\alpha}_{p}(I_{\mathcal{G}}, Q)$. The ANN vertex in $I'_{\mathcal{G}}$ is the same as that in $I_{\mathcal{G}}$, if $I'_{\mathcal{G}}$ satisfies all the following conditions:*

(1) $\forall\, e'_k \in E^{\alpha}_{p}(I_{\mathcal{G}}, Q), w''_k = w_k$;
(2) $\forall\, e'_k \in \tilde{E}^{\alpha}_{p}(I_{\mathcal{G}}, Q), w''_k \geq w_k$;
(3) $\exists\, e'_j \in \tilde{E}^{\alpha}_{p}(I_{\mathcal{G}}, Q), w''_k > w_k$.

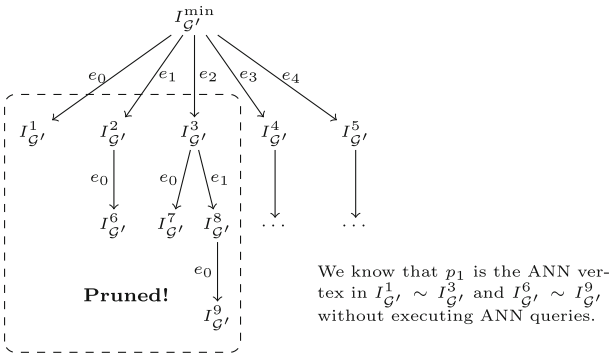


Figure 5 An example of instance pruning. $I^1_{\mathcal{G}'} \sim I^3_{\mathcal{G}'}$ and $I^6_{\mathcal{G}'} \sim I^9_{\mathcal{G}'}$ are derived from $I^{\min}_{\mathcal{G}'}$ by increasing weights on non-on-path edges (e_0, e_1 and e_2). The ANN vertex of those instances are the same as the ANN vertices in $I^{\min}_{\mathcal{G}'}$, i.e. p_1

Proof We prove this theorem by contradiction. Assume the vertices p is the ANN vertex in $I_{\mathcal{G}}$ w.r.t. Q and p is not the ANN vertex in $I'_{\mathcal{G}}$. There must exist a vertex $p' \neq p$ in $\mathcal{G}$ such that $d^{\alpha}(p', Q, I'_{\mathcal{G}}) < d^{\alpha}(p, Q, I'_{\mathcal{G}})$. With Lemma 1, we have $d^{\alpha}(p, Q, I_{\mathcal{G}}) \leq d^{\alpha}(p, Q, I'_{\mathcal{G}})$. Thus, $d^{\alpha}(p\hat{a}\check{A}\ddot{Y}, Q, I_{\mathcal{G}}) < d^{\alpha}(p, Q, I_{\mathcal{G}})$. In other words, p is not the ANN vertex in $I_{\mathcal{G}}$ which results in a contradiction. Hence the proof is complete. □

Therefore, we need not execute ANN queries in these instances. Instead, we only require to calculate the total existence probabilities of these instances (denoted as $P_N(Q, I^{\min}_{\mathcal{G}'})$) and add $P_N(Q, I^{\min}_{\mathcal{G}'})$ to the ANN probability of p_0. Calculation of the total existence probability $P_N(Q, I^{\min}_{\mathcal{G}'})$ is shown as follows.

Given an instance $I_{\mathcal{G}} \in \mathcal{G}$, we can find the ANN vertex v and the set of on-path edges $E^{\alpha}_{v}(I_{\mathcal{G}}, Q)$. The total existence probability of instances derived from $I_{\mathcal{G}}$ by increasing weights of edges in $\tilde{E}^{\alpha}_{v}(I_{\mathcal{G}}, Q)$, denoted as $P_N(Q, I_{\mathcal{G}})$, is calculated as

$$P_N(Q, I_{\mathcal{G}}) = \frac{Pr(I_{\mathcal{G}}) \cdot (1 - \prod_{e_i \in \tilde{E}^{\alpha}_{v}(I_{\mathcal{G}}, Q)} Pr(\omega_i < w_i))}{\prod_{e_i \in \tilde{E}^{\alpha}_{v}(I_{\mathcal{G}}, Q)} Pr(\omega_i = w_i)} - Pr(I_{\mathcal{G}}) \,, \tag{9}$$

where w_i is the weight value of e_i in $I_{\mathcal{G}}$. Practically, we need to calculate the $Pr(I_{\mathcal{G}}) + P_N(Q, I_{\mathcal{G}})$ during the instance pruning and it can be achieved by calculating the first item on the RHS of (9).

3.3.2 Instance pruning algorithm

The above observation motivates us to design an instance pruning algorithm, called Inst-Prun. Inputs of the Inst-Prun algorithm are an uncertain graph $\mathcal{G}$ and a set of query vertices Q. Detailed steps of the Inst-Prun algorithm are given as Algorithm 3.

To begin with, the ANN probability of each vertex in $\mathcal{G}$ is set to 0, and the min-weighted instance $I^{\min}_{\mathcal{G}}$ is constructed as the current instance $I_{\mathcal{G}}$ for checking (line 1–2 in Algorithm 3).

We then check the instance $I_{\mathcal{G}}$ and the instances which are derived by increasing the edge weight recursively using the function Check-Instance. For the instance $I_{\mathcal{G}}$, we calculate the ANN vertex v, the on-path edges $E^{\alpha}_{v}(I_{\mathcal{G}}, Q)$ by the certain ANN query (line 1 in Function Check-Instance). For edges which are on the shortest paths, we calculate the total existence probability $P_N(Q, I_{\mathcal{G}})$ according to (9) and add $P_N(Q, I_{\mathcal{G}}) + Pr(I_{\mathcal{G}})$ to the ANN probability of v (line 2–3). For each on-path edge e, we increase the weight on e to the next value among all possible weight values for constructing a new instance $I'_{\mathcal{G}}$ (line 6–8). And we call Check-Instance on $I'_{\mathcal{G}}$ recursively to find ANN vertices in $I'_{\mathcal{G}}$ and its descendants instances in the searching tree (line 9).

After the recursive searching, we get ANN probability of each vertex and find the vertex with the largest ANN probability (line 4 in Algorithm 3).

One instance may be checked redundantly, e.g., $I^{6}_{\mathcal{G}'}$ can be derived from $I^{\min}_{\mathcal{G}'}$ by increasing e_3 and then e_4, or increasing e_4 and then e_3. Thus, we introduce a rule during the instance enumeration as follows. *For an instance $I_{\mathcal{G}}$ which is derived from another instance by increasing the weight on e, $I_{\mathcal{G}}$ can only derive instances by*

Algorithm 3 Inst-Prun($\mathcal{G}$, Q)

Input : $\mathcal{G}$ — the uncertain graph (V, E, W);
V_s — the UG-ANN candidate vertexes set;
Q — the set of query vertices;
Output : Pr — probabilities of vertex as the ANN vertex;

1 $Pr_{1\ldots|V|} \leftarrow \{0, \ldots, 0\}$;
2 $I_{\mathcal{G}}^{\min} \leftarrow$ Construct the min-weighted instance from $\mathcal{G}$;
3 CHECK-INSTANCE($\mathcal{G}, I_{\mathcal{G}}^{\min}, Q, Pr$);
4 $V_{ANN} \leftarrow$Get the vertex with the maximal ANN probability;

Function CHECK-INSTANCE($\mathcal{G}, I_{\mathcal{G}}, Q, Pr$)

Input : $\mathcal{G}$ — the uncertain graph (V, E, W);
$I_{\mathcal{G}}$ — the instance;
Q — the set of query vertices;
V_c — the UG-ANN candidate vertexes set;
Pr — probabilities of vertex as the ANN vertex;

1 $(v_i, E_{v_i}^{\alpha}) \leftarrow \text{ANN}(Q, I_{\mathcal{G}}, V_c)$;
2 Calculate the total existence probabilities $P_N(Q, I_{\mathcal{G}})$ of instances generated by increasing weights of edges in $\tilde{E}_{v_i}^{\alpha}$ (Eq. 9);
3 $Pr_i \leftarrow Pr_i + [Pr(I_{\mathcal{G}}) + P_N(Q, I_{\mathcal{G}})]$;
4 $E_{op} \leftarrow$ Select edges whose id is smaller than i in $E_{v_i}^{\alpha}$;
5 **if** E_{op} is $\emptyset$ **then** return;
6 **for each** edge $e \in E_{op}$ **do**
7 **for each** possible weight value w of edge e **do**
8 $I'_{\mathcal{G}} \leftarrow$ Set the weight value of e to w;
9 CHECK-INSTANCE($\mathcal{G}, I'_{\mathcal{G}}, Q, V_c, Pr$);

increasing the on-path edge whose id is smaller than the id of e. Example 8 shows how the INST-PRUN algorithm works.

Example 8 Recall the uncertain graph $\mathcal{G}'$ in Example 6. Figure 6 illustrates how the INST-PRUN algorithm prunes instances and gets UG-ANN result. ANN probabilities of all vertices are initiated as 0. The min-weighted instance $I_{\mathcal{G}'}^{\min}$ is the root of the searching tree. $Pr(I_{\mathcal{G}'}^{\min}) = 0.2 \times 0.6 \times 0.5 \times 0.3 \times 0.2 = 0.0036$. p_1 is the ANN vertex in $I_{\mathcal{G}'}^{\min}$ and $E_{p_1}^{\alpha}(I_{\mathcal{G}'}^{\min}, Q) = \{e_3, e_4\}$. $e_0 \sim e_2$ are not on the shortest paths, then p_1 is the ANN vertex of instances generated from $I_{\mathcal{G}}^{\min}$ by increasing weights of e_0, e_1, e_2. According to (9), the total existence probability of those instances is $P_N(Q, I_{\mathcal{G}'}^{\min}) = \frac{Pr(I_{\mathcal{G}'}^{\min}) \times (1 - Pr(\omega_0 < 2.5)\, Pr(\omega_1 < 1.7)\, Pr(\omega_2 < 2.0))}{Pr(\omega_0 = 2.5)\, Pr(\omega_1 = 1.7)\, Pr(\omega_2 = 2.0)} - Pr(I_{\mathcal{G}'}) = \frac{0.0036 \times (1 - 0 \times 0 \times 0)}{0.2 \times 0.6 \times 0.5} - Pr(I_{\mathcal{G}'}) = 0.06 - Pr(I_{\mathcal{G}'})$. $Pr(p_1 \in \text{UG-ANN}(Q, \mathcal{G}')) = P_N(Q, I_{\mathcal{G}'}^{\min}) + Pr(I_{\mathcal{G}'}) = 0.06$. There are two edges e_3, e_4 in $E_{p_1}^{\alpha}(I_{\mathcal{G}'}^{\min}, Q)$ and we can generate instances by increasing weights of e_3 and e_4 respectively. By increasing weight of e_3 to 2.1, we get $I_{\mathcal{G}'}^{1}$ with existence probability $Pr(I_{\mathcal{G}'}^{1}) = 0.0084$ (ref. to Figure 6). The ANN query is executed in $I_{\mathcal{G}'}^{1}$ and p_0 is the ANN vertex. The ANN probability of p_0 is updated to 0.0084. $I_{\mathcal{G}'}^{1}$ can generate instances by increasing weights of e_1 or e_2, because $E_{p_0}^{\alpha}(I_{\mathcal{G}'}^{1}, Q) = \{e_1, e_2\}$ and the ids of e_1, e_2 are smaller than e_3. The search process goes recursively until there is no valid edge to increase. Note that $I_{\mathcal{G}'}^{5}$ only has one child $I_{\mathcal{G}'}^{6}$ in the searching tree in Figure 6. Because $I_{\mathcal{G}'}^{5}$ is generated by increasing weight of

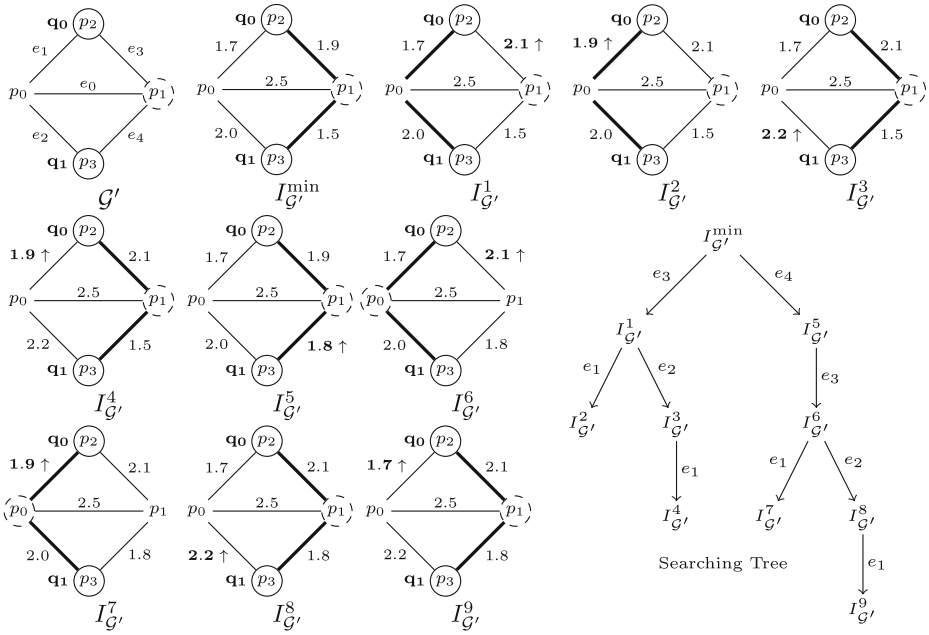


Figure 6 An example of the Inst-Prun algorithm. For each instance, the ANN vertex is drawn with *dashed circle*, query vertices are drawn with *solid circle*, the newly updated weight is marked with ↑, and the on-path edges are emphasized with *thick line*

e_4 and $E^{\alpha}_{p_1}(I^5_{\mathcal{G}'}, Q) = \{e_3, e_4\}$. Therefore, only the weight of e_3 can be increased in $I^5_{\mathcal{G}'}$. According to the searching tree illustrated in Figure 6, the total number of instances where ANN queries should be executed after instance pruning is 10. Compared with the 32 instances, Inst-Prun reduces the number of instances significantly.

3.4 The complete UG-ANN algorithm

Algorithm 4 shows the complete UG-ANN algorithm with structural pruning and instance pruning. The structural pruning method (Stru-Prun) reduces the size of $\mathcal{G}$, obtaining the query related subgraph $\mathcal{G}'$ and the set of UG-ANN candidate vertices V_s (line 1). The instance pruning method (Inst-Prun) is then used to check the instances in the searching tree while skipping the instances with the same UG-ANN vertices (line 2). After the instance pruning, the ANN probabilities of each vertex in $\mathcal{G}'$ are calculated and we can return the vertices which meet the need of different query types, i.e. Most-Prob UG-ANN query, Top-k UG-ANN query and Threshold-ϵ UG-ANN query (line 3–9).

4 Experimental evaluation

We evaluated the performance of the proposed pruning approaches for the UG-ANN query via extensive experiments. All experiments were conducted on a PC with Intel® Core 2 Duo T9400 CPU 2.53GHz processors, 4GB RAM, 160GB hard disk,

Algorithm 4 PRUN-UG-ANN($\mathcal{G}$, Q)

```
Input  : G — the uncertain graph (V, E, W);
         Q — the set of query vertices;
Output : V_ANN — the UG-ANN vertex
(G', V_s) ← STRU-PRUN(G, Q);
Pr[1, ..., |V_G'|] ← INST-PRUN(G', Q, V_s);
switch query type do
    case Most-Prob UG-ANN do
        return vertices with the largest ANN probability in Pr;
    case Top-k UG-ANN do
        return vertices with the k largest ANN probability in Pr;
    case Threshold-ϵ UG-ANN do
        return vertices with the ANN probability larger than or equal to ϵ;
```

Table 3 Data sets of real-world road networks

Name	City	$\|V\|$	$\|E\|$	$\|E\|/\|V\|$
SF	San Francisco	174,956	223,000	1.275
CA	California	21,048	29,693	1.411
NA	North America	175,813	179,179	1.019

running Microsoft Windows Server 2003 32-bits version. Experimental programs were implemented in C# and executed on Microsoft .Net framework 3.5 platform.

We used the US road network datasets[3] to generate the uncertain graphs for evaluation due to the lack of standard uncertain graph benchmarks with both weight uncertainty and edge existence uncertainty. As shown in Table 3, the datasets are composed of three road networks of cities or regions, i.e. San Francisco (SF), California (CA) and North America (NA). The road networks are certain graphs where vertices denote the locations and edge weights represent the length of road between two adjacent locations. To simulate the traffic conditions, the time cost on each edge was generated based on the original weight, because the time cost spent on each road is approximately linear to the length of road. For an edge e whose original weight is τ, we randomly generated s possible weight values $w_1 \ldots w_s$ from a uniform distribution with mean τ and variance 0.1τ, $s = 3$ in our experiments. All possible weight values shared the same probability, i.e., $Pr(\omega = w_i) = 1/s$, $1 \leq i \leq s$. The aggregate function we used in the following experiments was *max*, since the cases with the aggregate function *sum* are very similar to the *max* cases.

4.1 Comparative study

First of all, we compared the performance of our UG-ANN query algorithms (SE, SE-CF, and SE-CF-INST as shown in Table 4) with the brute-force algorithm (BF), because, to the best of our knowledge, there is no work addressing the problem of aggregate nearest neighbor query on uncertain graphs. Five vertices were randomly selected from each uncertain graph as the query vertices. To vary the graph size, we extracted a connected subgraph with a preferred vertex number from the original

[3] http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm

Table 4 Algorithms evaluated in experiments

Algorithm	Description
BF	The brute-force algorithm without pruning
SE	The algorithm with subgraph extraction (SE)
SE-CF	The algorithm with both subgraph extraction (SE) and candidate filtering (CF)
SE-CF-INST	The algorithm with subgraph extraction (SE), candidate filtering (CF) and instance pruning (INST)

graph as the experimental graph. The extraction adopts the breadth-first search style and starts from a random selected vertex. The average total response time of query processing on SF, CA and NA are shown in Figure 7a, b and c, respectively. Figure 7 validates that the brute-force algorithm suffers from high time complexity, i.e., the query time grows exponentially as the size of graph increases. The brute-force algorithm has to enumerate all instances in an uncertain graph and execute the certain ANN query on each instance. The total response time of executing a uncertain ANN query is exponential to the number of edges. On the other hand, the UG-ANN query algorithms equipped with the structural pruning (SE, CF) and instance pruning (INST) take very less time to process the query on the same uncertain graph.

Note that the brute-force algorithm costs the smallest time when the number of vertices $|V| = 5$, which means that all the vertexes in the uncertain graph are selected as the query vertexes. The structural pruning method thus cannot remove any vertexes, because the query related subgraph must contain all the query vertexes. Meanwhile, for each vertex $v \in V$, $d_{v \rightsquigarrow q} = 0 < D^{\alpha}_{\max}$ (ref. to Corollary 2) and each vertex may be the UG-ANN candidate. Finally, all the edges in the uncertain graph are on-path edges in all instances, since all five vertexes are query vertexes and there are very few edges ($|E| = 4$) connecting them. As a result, the query algorithms with pruning methods have to run some extra procedures for the pruning methods and, meanwhile, check all the instances as what the brute-force algorithm does.

Besides, we did not evaluate the brute-force approach in the following experiments where the number of edges in an uncertain graph was larger than 100, because

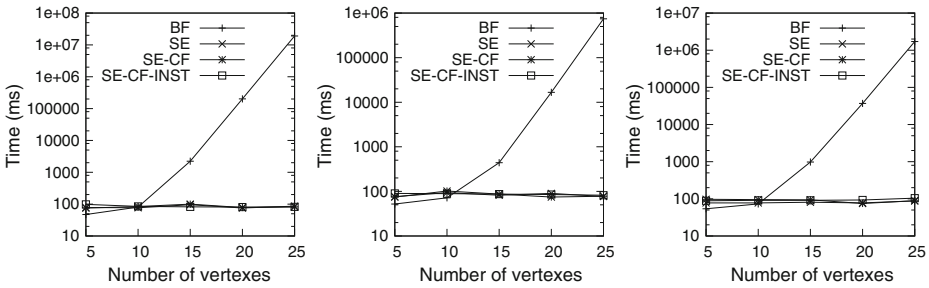


Figure 7 Comparative study between the brute force algorithm and algorithms with pruning methods

the processing time of the brute-force approach was intolerant (more than 10^6s when $|V| > 25$).

4.2 Effects of graph size

In this set of experiments, we evaluated the scalability of the proposed UG-ANN algorithm with pruning methods. We varied the vertices number from 1000 to 10, 000 at an interval of 1000 and fixed the number of query vertices to 5. The query process was composed of two phases, i.e. *structural pruning phase* and *searching phase*. The structural pruning phase contained the subgraph extraction and the candidate filtering. Whereas, the instance pruning was integrated in the searching phase. And we recorded not only the total response time of query processing but also the structural pruning time and the searching time during query processing procedure.

Structural pruning time Figure 8 displays the time for structural pruning on uncertain graphs with different sizes. It shows that the structural pruning time is superlinear to the vertex number, because we need to compute the aggregate distances of all vertices in $I_{\mathcal{G}}^{\min}$ derived from $\mathcal{G}$, and, for each vertex, the aggregate distance calculation takes $\mathcal{O}(|V|lgV + |E|)$ in the worst case. Moreover, the candidate filtering (CF) only takes few extra steps according to Algorithm 2 (line 14–15). Thus, the pruning time of the SE and SE-CF algorithms is approximately the same. Note that the time for instance pruning was not evaluated in this set of experiments, because, different from the structural pruning, the instance pruning goes along with the searching phase.

Searching time Figure 9 illustrates the searching time of UG-ANN queries on each data set. Compared with the brute-force algorithm, subgraph extraction greatly improves the efficiency of UG-ANN query processing so that it can handle uncertain graphs with thousands of vertices and edges. Besides, the candidate filtering method removes vertices from the possible ANN candidate sets and, therefore, the algorithm with both subgraph extraction and candidate filtering (SE-CF) decreases the searching time. Moreover, the SE-CF-INST algorithm achieves better performance when the instance pruning is adopted, since instances located in sub-trees of the search tree are pruned and we need not calculate the ANN vertices in those instances.

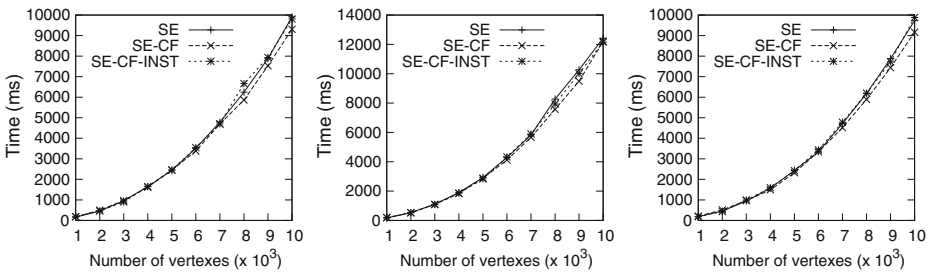


Figure 8 Effects of graph size on structural pruning time

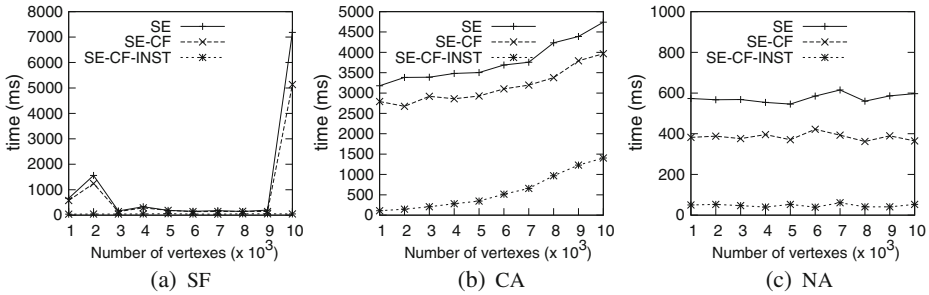


Figure 9 Effects of graph size on searching time

Total response time Finally, the total response time of query processing, which consists of pruning time and searching time, is reported in Figure 10. The SE-CF algorithm performs better than the SE algorithm, whereas the SE-CF-INST algorithm outperforms both the SE and SE-CF algorithms. The above results demonstrate the effectiveness of the proposed pruning methods, i.e. structural pruning, candidate filtering and instance pruning.

4.3 Effects of query cardinality

In this set of experiments, we evaluated the efficiency of our algorithm against the cardinality of query set. The workload was subgraphs with 10,000 vertices extracted from the CA, NA and SF data sets respectively. The extraction method is the same as that in the comparative study (Section 4.1). The query vertices were randomly selected from the extracted graphs. We varied the number of query vertices from 1 to 5. Figure 11 shows the total response time of query processing on CA, NA and SF, respectively. The total response time increases roughly as the size of query set grows. More query vertices result in a potential larger subgraph after the structural pruning and, therefore, decrease the performance of the structural pruning. Meanwhile, the results demonstrate that the SE-CF algorithm has better performance than the SE algorithm. And the SE-CF-INST algorithm has the best performance among all algorithms.

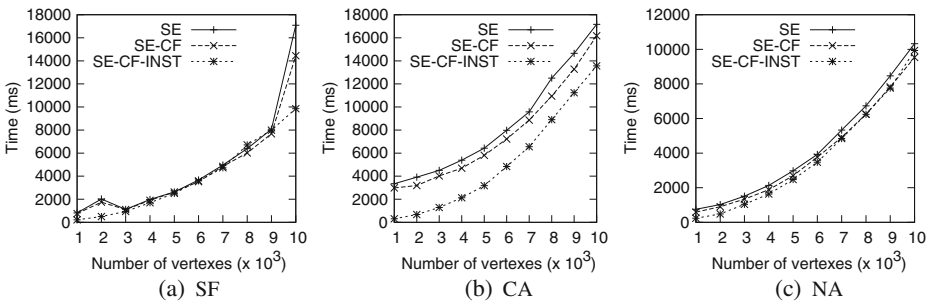


Figure 10 Effects of graph size on total response time

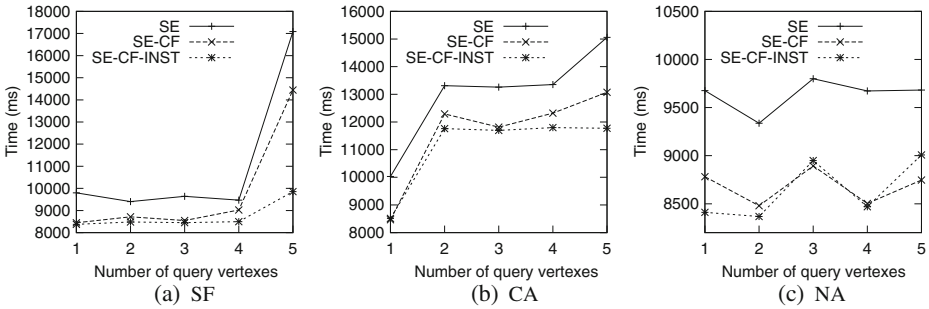


Figure 11 Effects of query size on total response time

4.4 Structural pruning performance

To evaluate the pruning power of the proposed structural pruning method and the candidate filtering method, we compute the vertex pruning rate and edge pruning rate in the same setting as the experiment in Section 4.1. The vertex pruning rate, denoted as r_v, is defined as the ratio of the removed vertices number to the original vertices number, i.e., $r_v = \frac{|V_{rm}|}{|V|}$, where V_{rm} is the set of pruned vertices. Similarly, the edge pruning rate, denoted as r_e, is defined as the ratio of the number of removed edges to the number of original edges, i.e., $r_e = \frac{|E_{rm}|}{|E|}$, where E_{rm} is the set of edges adjacent to pruned vertices.

Performance of subgraph extraction Figure 12 shows r_v and r_e w.r.t. the graph size. As shown in Figure 12a, most of vertices, i.e., more than 98 % vertices, are pruned by the subgraph extraction. And Figure 12b shows that r_e is approximately the same as r_v, since edges adjacent to the removed vertices are removed during the structural pruning phase.

Performance of candidate filtering In the same setting of the structural pruning experiments, we evaluated the candidate filtering performance of our algorithm. Candidate pruning is always executed after the structural pruning phrase and, therefore, the candidate pruning rate is defined as the ratio of pruned vertices

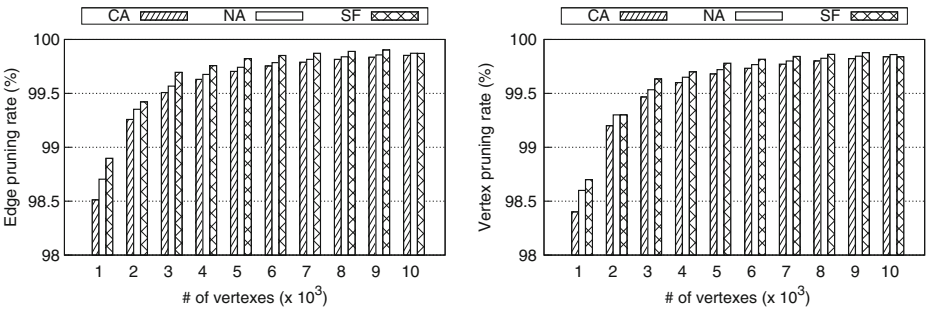


Figure 12 Effects of subgraph extraction in structural pruning

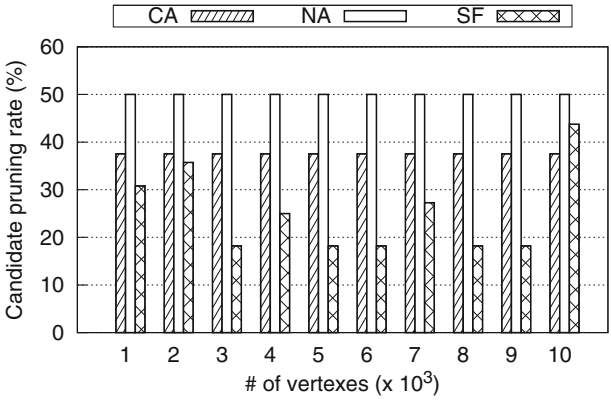


Figure 13 Performance of candidate filtering

number to the number of vertices in the subgraph after the subgraph extraction. Figure 13 depicts the candidate pruning rate w.r.t. the graph size. It validates that only parts of vertices in the subgraph are UG-ANN candidate vertices and other vertices are not necessary for aggregate distance computation.

4.5 Instance pruning performance

This set of experiments investigates the pruning capability of the proposed instance pruning approach. To measure the effectiveness of instance pruning, we first used the UG-ANN query algorithm with structural pruning (SE-CF) and got the number of instances N_S which the algorithm must enumerate and check. Based on the SE-CF algorithm, the SE-CF-INST is developed by introducing the the instance pruning approach. We recorded N_{SI}, the number of instances which the SE-CF-INST algorithm have to check. The performance of instance pruning is measured by the *instance pruning rate*, denoted as r_I. The instance pruning rate is defined as $r_I = \frac{N_S - N_{SI}}{N_{SI}}$. Figure 14 displays the instance pruning rates for uncertain graphs with different sizes. The performance of instance pruning depends on both the query vertices and the structure of the uncertain graphs. In our experiments, at least 60 % of

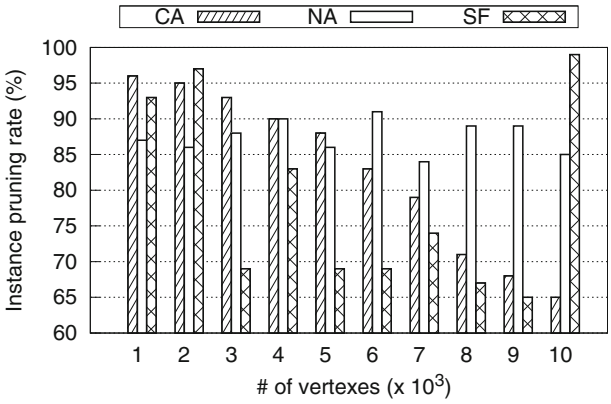


Figure 14 Performance of instance pruning

instances are pruned by the instance pruning during the UG-ANN query processing phase. In some cases, more than 90 % of instances are avoided from executing the ANN queries.

5 Related work

5.1 Aggregate nearest neighbor

Aggregate nearest neighbor (or group nearest neighbor) query processing [13, 15–17, 22] is the most related topic to ours. Papadias et al. [16, 17] first proposed the ANN query for multi-dimensional data. Three approaches, i.e. MQM, SPM and MBM, were developed to accelerate the ANN search processing. All the approaches employ the Euclidean distance in multi-dimension space. And the concepts of centroid point and minimal bound rectangle (MBR) are utilized in SPM and MBM, which is not suitable for vertices in graphs. Yiu et al. [22] noticed the fact that spatial data were usually constrained by spatial network, so they developed three methods, i.e. IER, TA and CE, for the ANN query in road network. All these approaches take advantage of the Euclidean distance for lower bound estimation, because vertices in road network are located with 2 or 3-dimensional coordinates. The primary difference of problem studied in [22] from ours is that the graph is certain, i.e., weights of edges are determined. Lian et al. [13] introduced the ANN query to uncertain data in multi-dimensional space, called PGNN. The uncertain data were modeled as a sphere region where a data point existed with a probability. To reduce the search space, a spatial pruning approach and a probabilistic pruning approach were proposed in [13]. The uncertain multi-dimensional data are quite different from uncertain graphs. In multi-dimensional space, the locations of two points are independent with each other. However, in uncertain graphs, the distances between vertices are correlative because of the uncertainty of edge weights.

5.2 Uncertain graph

Uncertain graph management [19, 23, 26] is another field related to our work. Zou et al. [26] investigated the problem of finding frequent subgraph patterns on an uncertain graph database. The frequent subgraph pattern mining is based on the expected support. For accelerating the mining process, the mining algorithm called MUSE was designed to find an approximate set of frequent patterns. The uncertain graph model of MUSE is different from ours, because MUSE only focuses on existence uncertainty of edges. However, our uncertain graph model proposed in this paper reveals both existence uncertainty and weight uncertainty of edges. Potamias et al. [19] discusses the kNN search problem in an uncertain graph. Different from our uncertain graph model, the uncertain graph model in [19] only reflects the edge existence uncertainty. In other words, it can only describe whether an edge is absent or not. Moreover, the uncertain graph defined in [19] is not a weighted graph where the distance between two adjacent vertices is always 1. Based on the uncertain graph model, three new distance metrics were proposed, i.e. median distance, majority distance and expected reliable distance. For each distance metric, the pruning algorithm was developed to reduce the search space. Most recently,

Yuan et al. [23] studied the threshold-based uncertain subgraph query problem. It aims to find the uncertain graphs whose subgraph isomorphic probability w.r.t the query graph is larger than the user preferred threshold. Our work is different from theirs in several aspects: (i) the query vertices of our problem are multiple and we investigate the aggregate distance w.r.t. all query vertices. (ii) our algorithm is not the approximate algorithm and is designed to find the exact answer.

6 Conclusion

Plenty of practical applications require aggregate nearest neighbor (ANN) query in uncertain graphs. However, existing ANN query algorithms fail to handle the uncertain graphs due to the uncertainty of edge weight. In this paper, we deal with the problem of aggregate nearest neighbor in uncertain graphs (UG-ANN). Firstly, we propose the uncertain graph model and formalize the problem of UG-ANN. Secondly, a basic UG-ANN query processing algorithm is developed. Thirdly, we propose several pruning methods to improve the efficiency of the UG-ANN query processing. Finally, extensive experiments on real data sets demonstrate that our pruning approach can improve the efficiency of UG-ANN query significantly.

In the future work, we plan to extend the proposed approach in two directions. Firstly, the proposed optimization is very effective when the set of query vertices is small compared to the uncertain graph, because the structural pruning limits the search only to the neighborhoods of the query vertices. We are now interested in the challenge that finding vertices that are close to *a large portion* of the graph. Secondly, the uncertain graph model proposed in this work assumes the uncertainty on edge weight and edge existence is independent between edges. The UG-ANN query considering the correlation between edges in the uncertain graph is another interesting future work.

Acknowledgements This work was supported by the National Natural Science Foundation of China (No. 61170064, No. 61133002), the National High Technology Research and Development Program of China (No. 2013AA013204, No. 2012AA011002) and the National HeGaoJi Key Project (No. 2010ZX01042-002-002-01). We would like to thank the anonymous reviewers and the editors for their insightful comments.

References

1. Agarwal, P.K., Cheng, S.W., Tao, Y., Yi, K.: Indexing uncertain data. In: PODS, pp. 137–146 (2009)
2. Blin, G., Sikora, F., Vialette, S.: Querying graphs in protein-protein interactions networks using feedback vertex set. IEEE-ACM Trans. Comput. Biol. Bioinform. **7**, 628–635 (2010)
3. Bohannon, J.: Counterterrorism's new tool: 'metanetwork' analysis. Science **325**(5939), 409–411 (2009)
4. Bondy, J.A., Murty, U.S.R.: Graph Theory, Graduate Texts in Mathematics, vol. 244. Springer (2008)
5. Chen, Z., Shen, H.T., Zhou, X., Yu, J.X.: Monitoring path nearest neighbor in road networks. In: SIGMOD, pp. 591–602 (2009)
6. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: SIGMOD, pp. 551–562 (2003)
7. Dalvi, N.N., Suciu, D.: Efficient query evaluation on probabilistic databases. In: VLDB, pp. 864–875 (2004)

8. Dijkstra, E.: A note on two problems in connection with graphs. Numer. Math. **1**, 269–271 (1959)
9. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM **34**(3), 596–615 (1987)
10. Hua, M., Pei, J., Zhang, W., Lin, X.: Ranking queries on uncertain data: a probabilistic threshold approach. In: SIGMOD, pp. 673–686 (2008)
11. Jensen, C.S., Kolárvr, J., Pedersen, T.B., Timko, I.: Nearest neighbor queries in road networks. In: GIS, pp. 1–8 (2003)
12. Kriegel, H.P., Kunath, P., Renz, M.: Probabilistic nearest-neighbor query on uncertain objects. In: DASFAA, pp. 337–348 (2007)
13. Lian, X., Chen, L.: Probabilistic group nearest neighbor queries in uncertain databases. IEEE Trans. Knowl. Data Eng. **20**(6), 809–824 (2008)
14. Liben-Nowell, D., Kleinberg, J.M.: The link-prediction problem for social networks. In: CIKM, pp. 556–559 (2003)
15. Luo, Y., Furuse, K., Chen, H., Ohbo, N.: Finding aggregate nearest neighbor efficiently without indexing. In: Infoscale, pp. 1–2 (2007)
16. Papadias, D., Shen, Q., Tao, Y., Mouratidis, K.: Group nearest neighbor queries. In: ICDE, pp. 301–312 (2004)
17. Papadias, D., Tao, Y., Mouratidis, K., Hui, C.K.: Aggregate nearest neighbor queries in spatial databases. ACM Trans. Database Syst. **30**(2), 529–576 (2005)
18. Pfoser, D., Jensen, C.S.: Capturing the uncertainty of moving-object representations. In: SSD, pp. 111–132 (1999)
19. Potamias, M., Bonchi, F., Gionis, A., Kollios, G.: k-nearest neighbors in uncertain graphs. PVLDB **3**(1), 997–1008 (2010)
20. Soliman, M.A., Ilyas, I.F., Chang, K.C.C.: Top-k query processing in uncertain databases. In: ICDE, pp. 896–905 (2007)
21. Yi, K., Li, F., Kollios, G., Srivastava, D.: Efficient processing of top-k queries in uncertain databases. In: ICDE, pp. 1406–1408 (2008)
22. Yiu, M.L., Mamoulis, N., Papadias, D.: Aggregate nearest neighbor queries in road networks. IEEE Trans. Knowl. Data Eng. **17**(6), 820–833 (2005)
23. Yuan, Y., Wang, G., Wang, H., Chen, L.: Efficient subgraph search over large uncertain graphs. PVLDB **4**(11), 876–886 (2011)
24. Yuen, S.M., Tao, Y., Xiao, X., Pei, J., Zhang, D.: Superseding nearest neighbor search on uncertain spatial databases. IEEE Trans. Knowl. Data Eng. **22**(7), 1041–1055 (2010)
25. Zhu, L., Zhang, A.: Supporting multi-example image queries in image databases. In: IEEE ICME, pp. 697–700 (2000)
26. Zou, Z., Li, J., Gao, H., Zhang, S.: Mining frequent subgraph patterns from uncertain graph data. IEEE Trans. Knowl. Data Eng. **22**(9), 1203–1218 (2010)