The VLDB Journal (2017) 26:467–492
DOI 10.1007/s00778-017-0460-y

REGULAR PAPER

On efficiently finding reverse *k*-nearest neighbors over uncertain graphs

Yunjun Gao[1,2,3] · **Xiaoye Miao**[1] · **Gang Chen**[1,2] · **Baihua Zheng**[4] · **Deng Cai**[1,3] · **Huiyong Cui**[1]

Received: 13 April 2016 / Revised: 1 February 2017 / Accepted: 24 February 2017 / Published online: 17 March 2017

Abstract Reverse *k*-nearest neighbor (R*k*NN) query on graphs returns the data objects that take a specified query object *q* as one of their *k*-nearest neighbors. It has significant influence in many real-life applications including resource allocation and profile-based marketing. However, to the best of our knowledge, there is little previous work on R*k*NN search over uncertain graph data, even though many complex networks such as traffic networks and protein–protein interaction networks are often modeled as uncertain graphs. In this paper, we systematically study the problem of *reverse k-nearest neighbor search on uncertain graphs* (UG-R*k*NN search for short), where graph edges contain uncertainty. First, to address UG-R*k*NN search, we propose three effective heuristics, i.e., GSP, EGR, and PBP, which minimize the original large uncertain graph as a much smaller *essential uncertain graph*, cut down the number of possible graphs via the newly introduced *graph conditional dominance relationship*, and reduce the validation cost of data nodes in order to improve query efficiency. Then, we present an efficient algorithm, termed as SDP, to support UG-R*k*NN retrieval by seamlessly integrating the three heuristics together. In view of the high complexity of UG-R*k*NN search, we further present a novel algorithm called TripS, with the help of an *adaptive stratified sampling* technique. Extensive experiments using both real and synthetic graphs demonstrate the performance of our proposed algorithms.

Keywords Uncertain graph · R*k*NN search · Stratified sampling · Query processing · Algorithm

✉ Gang Chen
cg@zju.edu.cn

Yunjun Gao
gaoyj@zju.edu.cn

Xiaoye Miao
miaoxy@zju.edu.cn

Baihua Zheng
bhzheng@smu.edu.sg

Deng Cai
dengcai@cad.zju.edu.cn

Huiyong Cui
cuihy@zju.edu.cn

1 College of Computer Science, Zhejiang University, Hangzhou, China

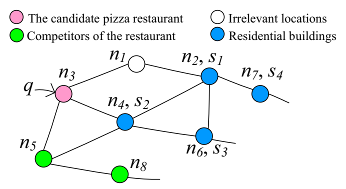
2 The Key Lab of Big Data Intelligent Computing of Zhejiang Province, Zhejiang University, Hangzhou, China

3 State Key Laboratory of CAD&CG, College of Computer Science, Zhejiang University, Hangzhou, China

4 School of Information Systems, Singapore Management University, Singapore, Singapore

1 Introduction

Management of uncertain graphs (or networks) has become an extremely important topic recently [8,15,16,23,29,32,35,36,53,58], because graph structured data that are *uncertain* or *noisy* by nature appear in diverse applications such as traffic networks and protein–protein interaction (PPI) networks. Uncertainty exists commonly due to various reasons such as anonymous communication data and data collected through automated sensors [3], and such uncertainty is usually encoded in the graph edges. Hence, a general uncertain graph model is utilized in this paper, where the graph edges are represented as several possible values with probabilities. This graph model considers not only the existence uncertainty (where an edge will virtually disappear if the edge weight is relatively large) but also the weight uncertainty (that allows each edge to have different weights).



(a)

edge	ω_1, p_1	ω_2, p_2
$e_{1,2}$	(2, 0.3)	(3, 0.7)
$e_{1,3}$	(1, 0.5)	(4, 0.5)
$e_{2,4}$	(3, 0.6)	(5, 0.4)
$e_{2,6}$	(3, 0.9)	(5, 0.1)
$e_{2,7}$	(2, 1.0)	–
$e_{3,4}$	(3, 0.8)	(4, 0.2)
$e_{3,5}$	(4, 0.4)	(5, 0.6)
$e_{4,5}$	(2, 0.7)	(3, 0.3)
$e_{4,6}$	(2, 1.0)	–
$e_{5,8}$	(1, 0.7)	(4, 0.3)

(b)

Fig. 1 Illustration of an uncertain graph. **a** Graph structure. **b** Possible weights

Given a query node q and a data node set S, a reverse k-nearest neighbor (RkNN) query finds a set of data nodes $s \in S$ such that q is one of s's kNN objects. RkNN search has large application base including location-based services, decision support, profile-based marketing, cluster and outlier detection, traffic networks, adventure games, and molecular biology [6,10,18,52]. In particular, there are two popular versions of RkNN search, i.e., the monochromatic one and the bichromatic one. The monochromatic RkNN query is alike the definition above, and all objects are of the same type. In contrast, there are two distinct object types S_1 and S_2 in the bichromatic RkNN query [7,22,52]. Specifically, given two data node sets S_1 (e.g., the customer set) and S_2 (e.g., the restaurant competitors) and a query object q, a bichromatic RkNN query retrieves the objects $s \in S_1$ such that q is one of their k-nearest neighbors from S_2.

Example 1 **RkNN search on uncertain traffic networks** In traffic networks, the weight of a road segment indicates the travel time (e.g., the driving time) of this road segment. The exact road weight is very hard to obtain, as a result of the decreasing frequency of data collection to save resources, data device limitations, and wireless network traffic. In other words, because of the difficulty of measuring the exact traffic time of every road segment and the large scale of a traffic network, the road weight is usually uncertain [29]. Take the network shown in Fig. 1 as an example; assume that the nodes denote the interesting landmarks on the road network, and the weights of each edge given in Fig. 1b represent the possible travel time values with the probabilities. For instance, the weight of edge $e_{1,2}$ between nodes n_1 and n_2 is 2 with probability 0.3 and 3 with probability 0.7. It indicates that passing the road n_1, n_2 takes two time units with probability 30%, and 3 time units with probability 70%.

Consider the scenario of evaluating the impact of opening a pizza restaurant at a selected location; the manager wants to examine how many residential areas would find the new pizza restaurant as their k-nearest choices [2,14,44], where the traveling time between a residential area and the pizza restaurant is a critical factor, and should be taken into account [30]. In addition, to cut down costs when carrying out an advertising campaign, it would be profitable for a restaurant owner to send menu cards only to those customers who have his/her restaurant as one of the k-closest pizza restaurants. For instance, for a candidate pizza restaurant location in node n_3 of Fig. 1, RkNN search can find all the residential buildings such that n_3 as one of their k-nearest pizza restaurants from the uncertain traffic network. Note that the network also contains some irrelevant nodes, such as node n_1 (e.g., shopping malls) in Fig. 1a. It is worth mentioning that the uncertain graph structure in Fig. 1 serves as a running example throughout the paper.

Example 2 **RkNN search on uncertain PPI networks** In protein–protein interaction (PPI) networks, nodes denote proteins, and edges represent interactions among them. The edge of a protein pair indicates the reliability of the interaction between the two proteins [28]. Since the interactions are derived through noisy and error-prone lab experiments, each edge is associated with an uncertainty value [4,19,36]. RkNN search can help to provide the best correlation of protein essentiality and placement of proteins in PPI networks [33]. In particular, RkNN topology, consisting of all proteins and the edges to every protein from their RkNNs (generated from the PPI network), is used to examine the connection of essential proteins and their placement. The related results show that essential proteins are more likely to be proteins with many RkNNs. Furthermore, RkNN search plays an important role in many gene ontology processes [33]. Besides, RkNN search is employed to identify novel inflammatory bowel disease (IBD)-related proteins via PPI networks in recent work [42]. An observed protein whose influenced proteins (i.e., RkNNs) are mostly known IBD-related proteins is statistically identified as a novel IBD-related protein. Therefore, the RkNN query with the statistical enrichment test is a great tool to identify IBD-related proteins in order to better understand the complex disease mechanism.

RkNN retrieval on deterministic graphs has been well studied, including both the monochromatic version and the bichromatic version [52]. Nonetheless, uncertain graph processing brings new challenges, because of the ♯P-hard complexity. Thus, applying directly existing RkNN query algorithms on deterministic graphs to answer UG-RkNN queries cannot achieve good performance. Consequently, motivated by these, in this paper, we focus on the problem of RkNN *search on uncertain graphs* (UG-RkNN search for short), where the uncertain graph is modeled as a set of deterministic possible graphs under *possible world semantic* [16,24,29,36,54]. Based on the uncertain graph model, given a query node q and a data node set S on an uncertain graph, a (monochromatic) UG-RkNN query finds a set of data nodes $s \in S$ such that q is one of s's kNN objects with probability not smaller than a specified probability threshold θ.

This paper is dedicated to the development of algorithms that can support UG-RkNN queries efficiently. Intuitively, in order to obtain UG-RkNN query result, we can convert an UG-RkNN query to multiple RkNN queries on all the possible graphs, which can be supported by existing RkNN search algorithms. Let E, V, and $\Omega(e)$ be the sets of edges, nodes, and possible weights associated with an edge e on an uncertain graph, respectively. It is not hard to find that there are in total $\Pi_{e \in E}|\Omega(e)|$ possible graphs w.r.t. a specified uncertain graph. If all the edges share the same number of weights w (i.e., $\forall e \in E, |\Omega(e)| = w$), the total number of R$k$NN queries we have to process for answering an UG-RkNN query is $w^{|E|}$, with the time complexity of $O(w^{|E|} \cdot (|V|^2 \cdot \lg |V| + |V| \cdot |E|))$ that is extremely expensive.

In order to make sure UG-RkNN query processing is practical, we have to minimize the number of possible graphs (i.e., $w^{|E|}$) as many as possible. To this end, we propose three novel heuristics to significantly improve search performance. First, we present *graph structure pruning* (GSP) heuristic to reduce the value of $|V|$ (and hence $|E|$) significantly by replacing an original graph with a much smaller *essential uncertain graph* and therefore cut down the number of possible graphs we have to evaluate. The effectiveness of GSP is proved by analysis as the number of nodes in the essential graph is *theoretically* bounded, and is demonstrated via experimental studies. Second, we present *equivalent graph removing* (EGR) heuristic to further reduce the number of possible graphs we have to evaluate during UG-RkNN search, by removing *equivalent possible graphs* that share the same result sets with the evaluated possible graphs. Nevertheless, simply identifying those equivalent possible graphs is as costly as finding RkNN query result on those possible graphs, as explained in Sect. 4. As a solution, we formulate *graph conditional dominance relationship* to locate certain equivalent possible graphs with low cost. Third, we develop *probability bound pruning* (PBP) heuristic to identify the objects that are qualified or unqualified UG-RkNN query results as early as possible during the search, which helps to minimize the validation cost of data nodes. Based on GSP, EGR, and PBP heuristics, we propose an exact algorithm called SDP to tackle UG-RkNN search.

Due to the $\sharp$P-hard complexity of UG-RkNN search, there is no exact algorithm which can handle UG-RkNN search in polynomial time. In view of this, we present a novel sampling algorithm, termed as TripS, to support UG-RkNN retrieval efficiently. Specifically, on the top of *essential uncertain graph* derived by GSP, TripS evaluates a group of sampled possible graphs by using an *adaptive stratified sampling* technique (denoted as ASSP) to get the query result, instead of the exhaustive evaluation over every possible graph. As theoretically guaranteed, ASSP, the adaptive stratified sampling technique utilized in TripS algorithm, is unbiased, and its variance is at least as good as that of Monte Carlo sampling. TripS takes $O(N \cdot (|E| + |V|^2 \cdot \lg |V|))$ time with N being the total number of samples. In brief, our key contributions in this paper are summarized as follows.

– We identify and formalize the problem of UG-RkNN query processing under a general graph model with uncertain edge weights.
– We present three effective heuristics GSP, EGR, and PBP, to effectively shrink the scale of uncertain graph and to reduce the number of possible graphs we have to evaluate during search. Based on these heuristics, we develop an *exact* algorithm, i.e., SDP, for UG-RkNN retrieval.
– We propose a novel algorithm, namely TripS, that integrates GSP with an adaptive stratified sampling algorithm, to perform UG-RkNN queries in high efficiency with theoretical guarantee. We also analyze the complexities of our presented algorithms, and extend our techniques to support bichromatic UG-RkNN search.
– We conduct extensive experiments with both real and synthetic graphs to demonstrate the effectiveness and efficiency of our proposed heuristics and algorithms.

The rest of the paper is organized as follows. First, we present preliminaries of our work in Sect. 2. Then, we describe GSP and EGR heuristics for UG-RkNN search in Sects. 3 and 4, respectively. Our SDP and TripS algorithms are proposed in Sects. 5 and 6, respectively. We report experimental results and our findings in Sect. 7. Finally, we review related work in Sect. 8 and conclude our work with some directions for future work in Sect. 9.

2 Preliminaries

In this section, we first formalize the uncertain graph model and our UG-RkNN problem, and then, we present a baseline algorithm, denoted as Baseline, for UG-RkNN search. Table 1 lists the symbols used frequently throughout this paper.

2.1 Problem formulation

Definition 1 (Uncertain graph) An uncertain weighted graph $\mathcal{G} = (V, E, \Omega, P)$, in which V denotes a set of nodes and E represents a set of edges. Each edge e is a random variable, $\Omega(e)$ is the sample space of e, and $P(x \in \Omega(e))$ is the probability mass function of e. In other words, Ω: $E \rightarrow \mathcal{P}(\mathcal{R})$, i.e., Ω maps each edge to a set of real values; and P: $E \rightarrow (\mathcal{R} \rightarrow [0, 1])$ is a function, which maps each edge to its probability mass function.

In fact, $\Omega(e)$ is a set of positive random events, i.e., $\Omega(e)(\subseteq \Omega) = \{\omega_1(e), \omega_2(e), \ldots, \omega_s(e)\}$, in which $\omega_i(e)$s

Table 1 Symbols and description

Notation	Description
$\mathcal{G}(V, E, \Omega, P)$	An uncertain graph
$\Phi(\mathcal{G})$	A set of all possible graphs for $\mathcal{G}$
$G(V, E, W)$	A possible graph in $\Phi(\mathcal{G})$
$\Omega(e)$	The set of all weights for edge $e \in E$
$\omega(e)$	A weight of edge e with $\omega(e) \in \Omega(e)$
w	The cardinality of $\Omega(e)$ for $\forall e \in E$
$\Pr(G)$	The existence probability of a possible graph G
$G^{\min}$	The possible graph with the minimum weight in every edge of $\mathcal{G}$
$G^{\max}$	The possible graph with the maximum weight in every edge of $\mathcal{G}$
S	A relevant data node set (included in a node set V) for a query node in $\mathcal{G}$
wd	Data density, i.e., the ratio of the number of nodes containing data objects to the total number of nodes, i.e., $d = \frac{\lvert S\rvert}{\lvert V\rvert}$
$SP_G(n_1, n_2)$	The shortest path between nodes $n_1, n_2 \in V$ on G
$d_G(n_1, n_2)$	The shortest path distance between nodes $n_1, n_2 \in V$ on G, i.e., the sum of all the edge weights on $SP_G(n_1, n_2)$
$d^k_G(n, S)$	The shortest path distance from a node $n \in V$ to its kth nearest neighbor (only for a data object set S) on G
$\mathcal{G}_e$	The essential uncertain graph of $\mathcal{G}$
D_{G_i}	The set of equivalent possible graphs for G_i
kNN(G, n, S)	The k-nearest data nodes in S to a node n on G
RkNN(G, q)	The RkNN result set of a query node q on G
Traverse(n, d, G)	A function returns all the nodes located at most d far from a node n on G

$(i = 1, 2, \ldots, s)$ are the positive weights of an edge e. Function P assigns each weight $\omega_i(e) \in \Omega(e)(\subseteq \Omega)$ of the edge e a probability $p(e, \omega_i(e))$ which indicates the likelihood of $\omega(e)$ with $\sum_{\omega_i(e)\in\Omega(e)} p(e, \omega_i(e)) = 1$. As formally defined in Definition 1, an uncertain graph actually refers to a graph whose edges have one or multiple weights with respective probabilities. Note that our uncertain graph model is similar as the one proposed in [29], and we focus on the discrete random variable case for every edge weight because of the following reasons: (i) the real-world uncertain data does not always follow any common probability distribution, e.g., Gaussian distribution; (ii) it is not always possible to capture accurately the probability density function (pdf) of real-world data; and (iii) sampling is often employed to obtain the distribution of random data in reality, which discretizes continuous random variables. In addition, our uncertain graph model is more general than the common model used in [16,24,36,54]. Specifically, our model considers not only the existence probabilities of edges (i.e., existence uncertainty) but also allows each edge to have different weights with different probabilities (i.e., weight uncertainty).

An uncertain graph $\mathcal{G}$ can generate a set of *possible graphs* (denoted as set $\Phi(\mathcal{G})$), as formalized in Definition 2. Different from the uncertain graph, a possible graph is a *deterministic* graph with each edge e having exactly one constant weight $\omega(e)$. It refers to a single instance of an uncertain graph $\mathcal{G}$, with every edge assigned *one and only one* weight. Since the likelihood of a given weight $\omega_i(e)$ could be very different from that of another weight $\omega_j(e)$, the chance that one possible graph G_i presents could be very different from the presence probability of another possible graph G_j. In order to capture how likely a specified possible graph may appear, we introduce the *existence probability*, as defined in Eq. 1. As a summary, an uncertain graph $\mathcal{G}$ is actually the union of all the possible graphs derived based on $\mathcal{G}$ with corresponding existence probabilities, i.e., $\mathcal{G} = \{(G_i, \Pr(G_i)) \mid G_i \in \Phi(\mathcal{G})\}$.

Definition 2 (Possible graph) Given an uncertain graph $\mathcal{G}$, a possible world of uncertain graph (so-called *possible graph*), denoted as $G = (V, E, W)$, is an instance of $\mathcal{G}$, where $W : E \to \mathbb{R}$ is the weight function that assigns each edge $e \in E$ a positive weight $\omega_G(e)$ with $\omega_G(e) \in \Omega(e)$. The existence probability of a possible graph G [16,24], denoted as $\Pr(G)$, is derived below.

$$\Pr(G) = \prod_{e\in E} p(e, \omega_G(e)) \tag{1}$$

Given a set of possible graphs w.r.t. an uncertain graph $\mathcal{G}$, we would like to point out that two graphs, denoted as $G^{\min}$ and $G^{\max}$, are *special*. $G^{\min}$ ($G^{\max}$) refers to the possible graph in which every edge is assigned to its minimum (maximum) possible weight, i.e., $\forall e \in E$, $\forall \omega_i(e) \in \Omega(e)$, and $\forall G \in \Phi(\mathcal{G})$, $\omega_{G^{\min}}(e) \leq \omega_G(e) \leq \omega_{G^{\max}}(e)$. We plot $G^{\min}$ and $G^{\max}$ of our sample uncertain graph in Fig. 2 for ease of understanding.

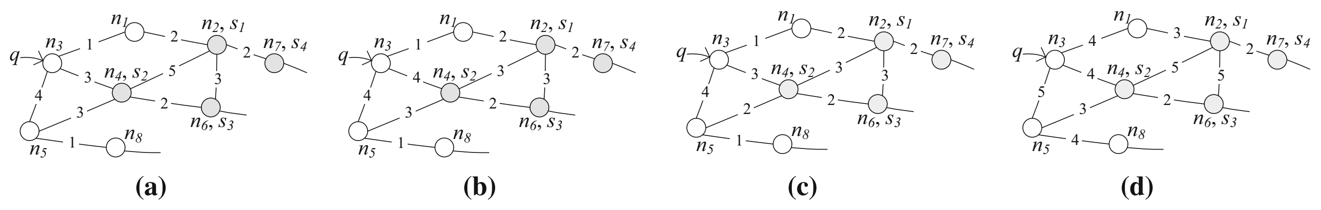


Fig. 2 Illustration of possible graphs for the uncertain graph in Fig. 1. **a** G_1: $\Pr(G_1) = 0.0036288$. **b** G_2: $\Pr(G_2) = 0.0013608$. **c** $G^{\min}$: $\Pr(G^{\min}) = 0.0127008$. **d** $G^{\max}$: $\Pr(G^{\max}) = 0.0001512$

The reason we emphasize on possible graphs $G^{\min}$ and $G^{\max}$ is the following observation. Given a pair of nodes n_i and n_j, its shortest path distance derived from $G^{\min}$ ($G^{\max}$) provides a lower bound (an upper bound) for the shortest path distance derived from any possible graph, as stated in Lemma 1 where function $d_G(n_i, n_j)$ returns the shortest path distance between n_i and n_j in a graph G.

Lemma 1 *Given an uncertain graph $\mathcal{G}(V, E, \Omega, P)$, let G_t be a possible graph in $\Phi(\mathcal{G})$. For any pair of connected nodes $n_i, n_j \in V$, it is guaranteed that $d_{G^{\min}}(n_i, n_j) \le d_{G_t}(n_i, n_j) \le d_{G^{\max}}(n_i, n_j)$.*

Proof Let $d(SP_G(n_i, n_j))$ denote the corresponding distance of the shortest network path between n_i and n_j (i.e., $SP_G(n_i, n_j)$) on a graph G. Based on the property that $\forall e \in E$ and $\forall \omega_i(e) \in \Omega(e)$, $\omega_{G^{\min}}(e) \le \omega_{G_t}(e) \le \omega_{G^{\max}}(e)$, we have

$$\begin{aligned} d_{G^{\min}}(n_i, n_j) &= d_{G^{\min}}(SP_{G^{\min}}(n_i, n_j)) \\ &\le d_{G^{\min}}(SP_{G_t}(n_i, n_j)) \\ &\le d_{G_t}(SP_{G_t}(n_i, n_j)) = d_{G_t}(n_i, n_j). \end{aligned}$$

Thus, $d_{G^{\min}}(n_i, n_j) \le d_{G_t}(n_i, n_j)$ holds. Similarly, we can derive that $d_{G_t}(n_i, n_j) \le d_{G^{\max}}(n_i, n_j)$ holds. Therefore, the proof completes. □

Based on the uncertain graph model, we formalize RkNN search and UG-RkNN query below. Note that function $d_G^k(s, S)$ returns the shortest path distance from s to its kth nearest node in set S on a deterministic graph G.

Definition 3 (RkNN search [52]**)** Given a deterministic graph $G = (V, E, W)$, a relevant data set $S \subseteq V$, and a query node $q \in V$, a RkNN *query* returns all the objects $s \in S$ that have q as one of their k-nearest neighbors, i.e.,

$$\mathrm{R}k\mathrm{NN}(G,\ q) = \{s \in S \mid d_G(q, s) \le d_G^k(s, S)\}. \tag{2}$$

Definition 4 (UG-RkNN search) Given an uncertain graph $\mathcal{G} = (V, E, \Omega, P)$, a relevant data set $S \subseteq V$, a user-specified parameter θ ($0 < \theta \le 1$), and a query node $q \in V$, an UG-RkNN *query* retrieves all the objects $s \in S$ that take q as one of their k-nearest neighbors with probability being at least θ. Formally,

$$\text{UG-R}k\text{NN}(\mathcal{G},\ q) = \{s \in S \mid \sum_{s \in \mathrm{R}k\mathrm{NN}(G \in \Phi(\mathcal{G}),\ q)} \Pr(G) \ge \theta\}$$

Back to our sample graph illustrated in Fig. 1. Suppose $S = \{n_2, n_4, n_6, n_7\}$, for the query node q (i.e., n_3), node n_2 is a R2NN object of q on the possible graph G_1 depicted in Fig. 2, as $d_{G_1}(q, n_2) = d_{G_1}^2(n_2, S)$ (=3), referring to Definition 3. Similarly, nodes n_4 and n_7 are also the R2NN objects of q on G_1. In addition, given an uncertain graph, the probability of node $s \in S$ being an UG-R2NN object of q is the accumulative existence probability of all the possible graphs on which s is one of q's RkNN objects. Therefore, for the sample uncertain graph shown in Fig. 1, the probability of nodes n_2, n_4, and n_7 being UG-R2NN objects is 0.0036288 on G_1.

The formalization of UG-RkNN search in Definition 4 follows the previous work on RkNN queries for uncertain data [5,6,26]. Compared with the one based on expected distances or expected ranks [36], it has two main advantages: (i) it calculates accurately the probability being an answer object for each object, which improves search accuracy and (ii) UG-RkNN search under this definition is adaptive and flexible, since the user-specified probability threshold θ is tunable. It is worthwhile to mention that our UG-RkNN definition does follow some important previous efforts on querying uncertain graphs [16,24,25,29].

In addition, it is easy to find that the UG-RkNN problem is in ♯P-hard class. First, kNN search on uncertain graphs (UG-kNN) is a ♯P-complete problem, as proved in [24]. Second, through computing the kNN objects for every data node in S on an uncertain graph using UG-kNN algorithm and returning the data nodes $o \in S$ such that a query node q belongs to o's kNN objects, the UG-RkNN query can be handled. Hence, UG-kNN problem could be reduced to UG-RkNN search in polynomial time, and UG-RkNN search is a ♯P-hard problem.

We would like to mention that due to space limitation, we present UG-RkNN search by mainly considering one input dataset S (i.e., *monochromatic* version), but the techniques proposed in this paper can be easily adjusted to support *bichromatic* version, which is discussed at the end of Sect. 6 and experimentally confirmed in experimental evaluation of Sect. 7. For the same reason, we assume that (i) the data nodes in S and query node q are located on nodes (i.e., $S \subseteq V$ and $q \in V$), while our techniques could directly support the case where they are located on edges, and (ii) the graph edges have independent probability mass functions, while our techniques can also be generalized to support the UG-RkNN query when the weights of a set of edges are correlated [13,40], where the existence probability of any possible world (i.e., the joint probability mass function) can be derived by inference techniques of probabilistic graph models (PGMs) such as junction tree and belief propagation. In particular, the pruning rule of GSP (to be presented in Sect. 3) is always available no matter edge weights are independent or not.

Furthermore, our proposed techniques in this paper can be generalized to tackle the UG-RkNN query under the graph model with uncertain nodes (e.g., considering the presence or non-presence of nodes and their adjacent edge with a given probability), which has a large application base in unreliable P2P networks and uncertain RDF graphs [27]. However, we would like to leave it to explore UG-RkNN search under such graph model (with uncertain nodes) in our future work.

2.2 Baseline algorithm

As mentioned in Sect. 1, the UG-RkNN query can be answered by performing RkNN search on every possible graph. Next, we present this brute-force solution, denoted as Baseline, with its pseudo-code depicted in Algorithm 1. Baseline takes as inputs an uncertain graph $\mathcal{G}$, a node set S, a query node q, and parameters k and θ, and outputs the result set S_G. It maintains the probability $\Pr(s)$ for each node $s \in S$ to record the accumulative probability of q being one of s's kNN objects, which is initialized to 0 (line 1). Thereafter, for every possible graph G from $\Phi(\mathcal{G})$, Baseline retrieves its RkNN objects using *lazy-EP* algorithm proposed in [52], and updates $\Pr(s)$ of answer objects s accordingly (lines 2–5). Finally, the algorithm returns all the nodes s with $\Pr(s) \geq \theta$, and terminates.

Algorithm 1 Baseline Algorithm (Baseline)

```
Input: an uncertain graph $\mathcal{G} = (V, E, \Omega, P)$, a data node set $S \subseteq V$,
a query node $q$, and parameters $k, \theta$
Output: the result set $S_G$ of an UG-R$k$NN query
/* lazy-EP$(G, S, q, k)$: the algorithm proposed in [52] is to return the
result set of a R$k$NN query on a deterministic graph $G$.*/
1: initialize $\Pr(s \in S) \leftarrow 0$
2: for each possible graph $G \in \Phi(\mathcal{G})$ do
3:     $\Pr(G) \leftarrow \prod_{e \in E} p(e, \omega_G(e))$     // by Eq. 1
4:     for each data object $s \in lazy\text{-}EP(G, S, q, k)$ do
5:         $\Pr(s) \leftarrow \Pr(s) + \Pr(G)$
6: return $S_G$ containing all the nodes $s \in S$ with $\Pr(s) \geq \theta$
```

RkNN search on a deterministic graph takes $O(|V|^2 \cdot \lg|V| + |V| \cdot |E|)$ time, if it is assumed that a Dijkstra-alike algorithm is adopted [16,29]. Consequently, Baseline needs $O(\Pi_{e \in E}|\Omega(e)| \cdot (|V|^2 \cdot \lg|V| + |V| \cdot |E|))$ time to perform UG-RkNN search. If we assume that all the edges share the same number of distinct weights, denoted as w, Baseline takes $O(w^{|E|} \cdot (|V|^2 \cdot \lg|V| + |V| \cdot |E|))$ time. Thus, Baseline is very *inefficient*, and we have to improve the efficiency of UG-RkNN retrieval for its practicability in real-life applications.

3 Graph structure pruning

Based on the above discussion, we understand that there are in total $\Pi_{e \in E}|\Omega(e)|$ possible graphs corresponding to a given uncertain graph $\mathcal{G}$. Given the fact that $\Omega(e)$ is determined by the applications, we can consider reducing the value of $|E|$. As a result, we propose a novel pruning technique, i.e., *graph structure pruning* (GSP), to reduce the number of nodes (i.e., $|V|$) and the number of edges (i.e., $|E|$) accordingly. Specifically, GSP is to serve the purpose of reducing the number of possible graphs we have to consider during UG-RkNN search. Before presenting GSP, we first introduce the concept of *dominance* between two deterministic graphs, formally defined in Definition 5.

Definition 5 (Graph dominance) Given two possible graphs $G_i = (V, E, W')$ and $G_j = (V, E, W'')$ w.r.t. an uncertain graph $\mathcal{G}$, we say that G_i dominates G_j, denoted as $G_i \prec G_j$, iff (i) $\forall e \in E, \omega_{G_i}(e) \leq \omega_{G_j}(e)$, and (ii) $\exists e' \in E$, $\omega_{G_i}(e') < \omega_{G_j}(e')$.

Take the possible graphs shown in Fig. 2 as an example. There is no dominance relationship between G_1 and G_2 due to $\omega_{G_1}(e_{3,4}) < \omega_{G_2}(e_{3,4})$ and $\omega_{G_1}(e_{2,4}) > \omega_{G_2}(e_{2,4})$. Graph $G^{\min}$ dominates G_1 since edges $e_{2,4}$ and $e_{4,5}$ both have smaller weights in $G^{\min}$ than their weights in G_1, and all the other edges have the same weights in $G^{\min}$ and G_1. Obviously, $G^{\min}$ dominates all the other possible graphs, while $G^{\max}$ is dominated by all the other possible graphs.

The reason we introduce the graph dominance relationship is that we want to present two important properties on the shortest path distances between nodes in a given possible graph G_j and another possible graph G_i that dominates G_j, as stated in Lemma 2.

Lemma 2 *Given an uncertain graph $\mathcal{G}$, let $G_i, G_j \in \Phi(\mathcal{G})$ be two possible graphs such that $G_i \prec G_j$. It is confirmed that (i) $\forall n_1, n_2 \in V$, their shortest path distance on G_i is bounded by that on G_j, i.e., $\forall n_1, n_2 \in V$, $d_{G_i}(n_1, n_2) \leq d_{G_j}(n_1, n_2)$; and (ii) $\forall n \in V$ and $\forall S \subseteq V$, the shortest path distance between n and its kth nearest neighbor in S on G_i cannot exceed that on G_j, i.e., $\forall n \in V$ and $\forall S \subseteq V$, $d^k_{G_i}(n, S) \leq d^k_{G_j}(n, S)$.*

Proof As $G_i \prec G_j$, it is certain that $\forall e \in E$, $\omega_{G_i}(e) \leq \omega_{G_j}(e)$, and $\exists e' \in E$, $\omega_{G_i}(e') < \omega_{G_j}(e')$ according to Definition 5. First, we prove statement (i) $d_{G_i}(n_1, n_2) \leq d_{G_j}(n_1, n_2)$ as follows.

$$\begin{aligned} d_{G_i}(n_1, n_2) &= d_{G_i}(SP_{G_i}(n_1, n_2)) \leq d_{G_i}(SP_{G_j}(n_1, n_2)) \\ &\leq d_{G_j}(SP_{G_j}(n_1, n_2)) = d_{G_j}(n_1, n_2). \end{aligned}$$

Then, for statement (ii), we can get

$$\begin{aligned} d^k_{G_j}(n, S) &= \max\{d_{G_j}(n, s) \mid s \in k\text{NN}(G_j, n, S)\} \\ &\geq \max\{d_{G_i}(n, s) \mid s \in k\text{NN}(G_j, n, S)\} \\ &\geq d^k_{G_i}(n, S). \end{aligned}$$

Here, kNN(G, n, S) denotes the set of k objects in S that are closest to n on G. Thus, we have $d_{G_i}(n_1, n_2) \leq d_{G_j}(n_1, n_2)$ and $d^k_{G_i}(n, S) \leq d^k_{G_j}(n, S)$. The proof completes. □

Again, we use G_1 and $G^{\min}$ depicted in Fig. 2 as an example. As $G^{\min} \prec G_1$, the distance between any two nodes on $G^{\min}$ is definitely bounded by that on G_1, e.g., $d_{G^{\min}}(n_5, n_2)(= 5) < d_{G_1}(n_5, n_2)(= 7)$ and $d_{G^{\min}}(n_2, n_3) = d_{G_1}(n_2, n_3) = 3$. In addition, the distance from a given node n to its kth nearest node from S in $G^{\min}$ cannot exceed that in G_1. Take node n_2 as an example. Given $S = \{n_2, n_4, n_6, n_7\}$, we have $d^3_{G^{\min}}(n_2, S) = d_{G^{\min}}(n_2, n_4) = 3$ and $d^3_{G_1}(n_2, S) = d_{G_1}(n_2, n_4) = 5$. It is worth noting that the kth nearest node of a specified node n in one possible graph might be different from its kth nearest node in another possible graph, although n_2 has the same third nearest node (i.e., n_4) in both $G^{\min}$ and G_1.

Based on Lemma 2 and the fact that $G^{\min}$ ($G^{\max}$) dominates (is dominated by) all the other possible graphs, we derive the *upper bound* of the distance between a given node n and its kth nearest node in any possible graph, and use this upper bound to prune away all the nodes that cannot be one of n's kNN objects in any given possible graph, as stated in Lemma 3. Here, function Traverse$(n, d^k_{G^{\max}}(n, S), G^{\min})$ returns all the nodes in the graph $G^{\min}$ that are located at most $d^k_{G^{\max}}(n, S)$ away from n.

Lemma 3 *Given an uncertain graph $\mathcal{G}$, let $G \in \Phi(\mathcal{G})$ be a possible graph, $n \in V$ be a node, and $S \subseteq V$ be a data node set. It is confirmed that $d^k_{G^{\min}}(n, S) \leq d^k_G(n, S) \leq d^k_{G^{\max}}(n, S)$. Consequently, the k-nearest nodes to n are located at most $d^k_{G^{\max}}(n, S)$ away from n in $G^{\min}$, i.e., kNN$(G, n, S) \subseteq$* Traverse *$(n, d^k_{G^{\max}}(n, S), G^{\min})$.*

Proof We prove kNN$(G, n, S) \subseteq$ Traverse$(n, d^k_{G^{\max}}(n, S), G^{\min})$ by contradiction. Assume that there is a node $n' \in V$ such that one of its k-nearest nodes o in $G_i \in \Phi(\mathcal{G})$ has its distance to n' on $G^{\min}$ larger than $d^k_{G^{\max}}(n', S)$, i.e., $\exists o \in k$NN(G_i, n', S) such that $d_{G^{\min}}(n', o) > d^k_{G^{\max}}(n', S)$. On the one hand, based on Lemma 2 and the assumption that $o \in k$NN(G_i, n', S), it is certain that $d_{G^{\min}}(o, n') \leq d_{G_i}(o, n') \leq d^k_{G_i}(n', S)$. On the other hand, according to Definition 5, we have $G^{\min} \prec G \prec G^{\max}$, for any $G \in \Phi(\mathcal{G}) - \{G^{\min}, G^{\max}\}$. Then, we can guarantee $d^k_{G^{\min}}(n, S) \leq d^k_G(n, S) \leq d^k_{G^{\max}}(n, S)$ based on the statement (ii) of Lemma 2, i.e., $d^k_{G^{\min}}(n', S) \leq d^k_{G_i}(n', S) \leq d^k_{G^{\max}}(n', S)$. Next, we can derive that $d_{G^{\min}}(o, n') \leq d^k_{G^{\max}}(n', S)$, i.e., $o \in$ Traverse$(n', d^k_{G^{\max}}(n', S), G^{\min})$, which contradicts with our assumption. The proof completes. □

Algorithm 2 Graph Structure Pruning (GSP)

Input: an uncertain graph $\mathcal{G} = (V, E, \Omega, P)$, a data node set $S \subseteq V$, a query node q, and parameter k
Output: an essential uncertain graph $\mathcal{G}_e$ and a candidate result set N_c

```
1: initialize $H \leftarrow \{\langle q, 0\rangle\}$, and $N_c \leftarrow N_g \leftarrow \varnothing$
2: get $G^{\min}$ and $G^{\max}$ of $\mathcal{G}$
3: while $H$ is not empty do
4:     $\langle n, d_{G^{\min}}(n, q)\rangle \leftarrow$ de-heap$(H)$
5:     if $n$ is not visited then
6:         $d^{\max} \leftarrow d^k_{G^{\max}}(n, S)$ and mark $n$ as visited
7:         if $d_{G^{\min}}(n, q) \leq d^{\max}$ then
8:             $N_c \leftarrow N_c \bigcup \{n\}$    // Lemma 4
9:             $N_g \leftarrow N_g \bigcup$ Traverse$(n, d^{\max}, G^{\min})$    // Heuristic 1
10:        for each adjacent unvisited node $n_i$ of $n$ do
11:            en-heap$\langle n_i, d_{G^{\min}}(n, q) + d_{G^{\min}}(n, n_i)\rangle$
12: construct $\mathcal{G}_e$ by nodes in $N_g$ with the corresponding edges
13: return $\mathcal{G}_e$ and $N_c$
```

Applying Lemma 3 in UG-RkNN search, we are certain that for any given node n, if a query node $q \notin$ Traverse$(n, d^k_{G^{\max}}(n, S), G^{\min})$, then q cannot be one of n's kNN objects, and thus, node n cannot be an UG-RkNN object for q, as stated in Lemma 4.

Lemma 4 (Candidate result set) *Given an uncertain graph $\mathcal{G}$, a node set $S \subseteq V$, and a query node q, let N_c be the set of nodes $n \in V$ satisfying $d_{G^{\min}}(n, q) \leq d^k_{G^{\max}}(n, S)$, i.e.,*

$$N_c = \{n \in V \mid d_{G^{\min}}(n, q) \leq d^k_{G^{\max}}(n, S)\}. \quad (3)$$

Then, UG-RkNN$(\mathcal{G}, q) \subseteq N_c$*, and nodes in $(V - N_c)$ can be safely pruned.*

Proof Assume that the above statement is invalid, and there is at least one answer object $n' \in$ UG-RkNN$(\mathcal{G}, q)$ such that $n' \in (V - N_c)$ with $d_{G^{\min}}(n', q) > d^k_{G^{\max}}(n', S)$. According to Lemma 2, we have $d_{G_i}(n', q) \geq d_{G^{\min}}(n', q)$, and hence, $d_{G_i}(n', q) > d^k_{G^{\max}}(n', S) \geq d^k_{G_i}(n', S)$, i.e., n' is not an answer object. Therefore, our assumption is invalid, and the proof completes. □

Based on the fact that set N_c is a super set that covers all the answer objects for an UG-RkNN query, we present *graph structure pruning* (GSP) in Heuristic 1. GSP forms an *essential uncertain graph*, denoted as $\mathcal{G}_e$, which is a subgraph of $\mathcal{G}$ and meanwhile contains all the candidate nodes in N_c for an UG-RkNN query. The other node set N_g is formed by all the nodes in the formed essential uncertain graph, which is a super set of N_c, i.e., $N_c \subseteq N_g$, as stated in Lemma 5.

Heuristic 1 (Graph structure pruning) *Given an uncertain graph $\mathcal{G}$, a data node set $S \subseteq V$, and a query node q, let N_g represent the set of nodes returned by* $\mathsf{Traverse}(n, d^k_{G^{\max}}(n, S), G^{\min})$ *for the nodes $n \in N_c$ (satisfying $d_{G^{\min}}(n, q) \leq d^k_{G^{\max}}(n, S)$), i.e.,*

$$N_g = \bigcup_{n \in N_c} \mathsf{Traverse}(n, d^k_{G^{\max}}(n, S), G^{\min}). \tag{4}$$

For UG-RkNN search, it only needs to consider the subgraph, namely the essential uncertain graph *$\mathcal{G}_e$, formed by N_g and corresponding edges between the nodes in N_g.*

Proof To the contrary, we assume that evaluating $\mathcal{G}_e$ is not sufficient to answer UG-RkNN search on $\mathcal{G}$, as there is at least one object/node $s' \in (V - N_g)$ such that (i) s' is an answer object in $G_i \in \Phi(\mathcal{G})$, or (ii) s' directly affects the decision on whether a data object $o \in N_c$ is an answer object in $G_j \in \Phi(\mathcal{G})$ or not. First, we prove case (i) is impossible. According to Lemma 4, answer object $s' \in N_c$ and $s' \in \mathsf{Traverse}(s', d^k_{G^{\max}}(s', S), G^{\min})$. Therefore, we can confirm that $s' \in N_g$ based on Eq. 4, which contradicts with our assumption that $s' \in (V - N_g)$. Hence, case (i) is impossible. Next, we prove case (ii) is also invalid. According to Lemma 3 and Eq. 4, for each candidate object $o \in N_c$, all its possible k-nearest nodes are included in N_g, i.e., $k\mathrm{NN}(G_j, o, S) \subseteq \mathsf{Traverse}(o, d^k_{G^{\max}}(o, S), G^{\min}) \subseteq N_g$. Thus, $\forall s' \in (V - N_g)$, it cannot change the fact that any object $o \in N_c$ is or is not an answer object, i.e., case (ii) is also invalid. The proof completes. □

Lemma 5 *Given node sets N_c and N_g w.r.t. a given uncertain graph $\mathcal{G}$, it is confirmed that $N_c \subseteq N_g$. Accordingly, $\mathcal{G}_e$ covers all the candidates in N_c for the UG-RkNN query.*

Proof The proof is very straightforward, and thus, we skip it due to limited space. □

Heuristic 1 suggests that only an essential uncertain graph $\mathcal{G}_e$, a subgraph of a specified uncertain graph $\mathcal{G}$, is necessary for processing an UG-RkNN query. Thus, we present GSP algorithm as the implementation of Heuristic 1 to form the essential uncertain graph $\mathcal{G}_e$, with its pseudo-code listed in Algorithm 2. First, GSP initializes a min-heap H, a candidate result set N_c, and a node set of an essential uncertain graph N_g, and then, it gets the possible graphs $G^{\min}$ and $G^{\max}$ (lines 1–2). Note that all the entries in H correspond to nodes $n \in V$ in the form of $\langle n, d_{G^{\min}}(n, q)\rangle$, sorted in ascending order of their distances to a query node q on $G^{\min}$. Next, GSP starts the network expansion to retrieve the candidate answer objects, guided by H (lines 3–11). GSP de-heaps the head entry $\langle n, d_{G^{\min}}(n, q)\rangle$ from H, and processes n only if n has not yet been visited until H is empty. It drives the upper bound $d^{\max}$ of n's distance to its kth nearest node in any possible graph, and compares $d^{\max}$ with the lower bound of the distance from n to q (lines 6–7). When the lower bound does not exceed $d^{\max}$, node n is inserted into N_c based on Lemma 4, and the nodes returned by function $\mathsf{Traverse}(n, d^{\max}, G^{\min})$ are added to N_g based on Heuristic 1 (lines 8–9). In the sequel, GSP en-heaps all n's unvisited adjacent nodes into H for expansion later (lines 10–11). Thereafter, GSP constructs $\mathcal{G}_e$ based on nodes in N_g, and then, it returns both N_c and $\mathcal{G}_e$ to demonstrate the algorithm (lines 12–13).

Example 3 In order to illustrate how GSP forms an essential uncertain graph for ease of understanding, we assume that an UG-R1NN ($k = 1$) query is issued from a node n_3 on the sample uncertain graph depicted in Fig. 1 with $S = \{n_2, n_4, n_6, n_7\}$. Table 2 lists the procedure of GSP, based on its possible graphs $G^{\min}$ and $G^{\max}$ shown in Fig. 2. GSP first evaluates the query node n_3 from a min-heap H, and gets $d^1_{G^{\max}}(n_3, S) = 4$ from $G^{\max}$, which is larger than $d(n_3, q)$ $(= 0)$. Thus, GSP sets N_c as $\{n_3\}$, inserts all the nodes that are located within distance $d^1_{G^{\max}}(n_3, S) = 4$ to n_3 on $G^{\min}$ into N_g $(= \{n_3, n_1, n_4, n_2, n_5\})$, and en-heaps n_3's adjacent nodes $\{n_1, n_4, n_5\}$ into H. Similarly, GSP visits the nodes n_1, n_4, n_2, and n_5 in order (refer to Table 2). Once H is empty, the essential uncertain graph is constructed based on the node set $N_g = \{n_3, n_1, n_4, n_2, n_5\}$, and $N_c = \{n_3, n_1\}$ forms a candidate result set for UG-RkNN search.

Since GSP forms a subgraph $\mathcal{G}_e$ that contains fewer nodes and hence fewer edges, it helps to reduce the number of possible graphs. Take our sample graph as an example. GSP

Table 2 Procedure of graph structure pruning

Object	$d^{\max}$	N_c	N_g	H
$\langle n_3(q), 0\rangle$	4	$\{n_3\}$	$\{n_3, n_1, n_4, n_2, n_5\}$	$\langle n_1, 1\rangle, \langle n_4, 3\rangle, \langle n_5, 4\rangle$
$\langle n_1, 1\rangle$	3	$\{n_3, n_1\}$	$\{n_3, n_1, n_4, n_2, n_5\}$	$\langle n_4, 3\rangle, \langle n_2, 3\rangle, \langle n_5, 4\rangle$
$\langle n_4, 3\rangle$	2	$\{n_3, n_1\}$	$\{n_3, n_1, n_4, n_2, n_5\}$	$\langle n_2, 3\rangle, \langle n_5, 4\rangle$
$\langle n_2, 3\rangle$	2	$\{n_3, n_1\}$	$\{n_3, n_1, n_4, n_2, n_5\}$	$\langle n_5, 4\rangle$
$\langle n_5, 4\rangle$	3	$\{n_3, n_1\}$	$\{n_3, n_1, n_4, n_2, n_5\}$	$\varnothing$

forms a $\mathcal{G}_e$ with only 5 nodes and 6 edges, and it cuts down the number of possible graphs to $\Pi_{e \in E'} |\Omega(e)| = 64$ where E' is the edge set of $\mathcal{G}_e$.

In what follows, we prove the correctness of GSP heuristic in Lemma 6 and approximate the size of an essential uncertain graph $\mathcal{G}_e$ (in terms of $|N_g|$) in Lemma 7.

Lemma 6 (Correctness) *Given an uncertain graph $\mathcal{G}$ and a query node q, if a node $n \in V$ is not visited during the traversal of $G^{\min}$ in* Algorithm 2*, node n cannot be an answer object on any possible graph $G_i \in \Phi(\mathcal{G})$, that is,* GSP *algorithm is correct with no false negative* (*i.e., there are* no missing answers) *and no false positive* (*i.e., there are* no false answers) *for UG-RkNN search.*

Proof Assume, to the contrary, that an answer object s on one possible graph $G_i \in \Phi(\mathcal{G})$ is not visited during the traversal of $G^{\min}$. Since s is not visited during the traversal of $G^{\min}$, there must be at least one node n on the path $SP_{G_i}(n', q)$ on a graph G_i satisfying $d_{G^{\min}}(n, q) > d^k_{G^{\max}}(n, S)$. According to Lemma 3, node q is not included in kNN(G, n, S) for $\forall G \in \Phi(\mathcal{G})$ (i.e., n cannot be an answer object for $\mathcal{G}$). Thus, we have

$$\begin{aligned} d_{G_i}(n', q) &= d_{G_i}(n', n) + d_{G_i}(n, q) \\ &> d_{G_i}(n', n) + d^k_{G_i}(n, S) \\ &\geq \max\{d_{G_i}(n', o) \mid o \in k\mathrm{NN}(G_i, n, S)\} \\ &= d^k_{G_i}(n', S). \end{aligned}$$

Then, we can conclude that the node n' (data object s) is not a real answer object on G_i, and the assumption is invalid. Thus, for any unvisited node n in Algorithm 2, n cannot be an answer object on any possible graph $G \in \Phi(\mathcal{G})$, i.e., GSP algorithm has no false negative for UG-RkNN search.

On the other hand, we know that set N_c covers all the answer objects for an UG-RkNN query, as stated in Lemma 4, and all the candidate nodes $n \in N_c$ with all their k-nearest neighbors are included in the essential uncertain graph $\mathcal{G}_e$ for a given uncertain graph $\mathcal{G}$ based on Heuristic 1 and Lemma 5. Consequently, it is guaranteed that GSP has no false positive. Hence, GSP is correct for UG-RkNN retrieval, and the proof completes. □

Lemma 7 (Effectiveness) *Given an UG-RkNN query w.r.t. an uncertain graph $\mathcal{G}$, assume that every node n contains a data object* (*i.e., data density d =1*)*, and let ρ denote the maximal average degree in $\mathcal{G}$. The cardinality of node set N_g of the essential uncertain graph $\mathcal{G}_e$, i.e., $|N_g|$, is bounded by* $\min(\frac{\rho^{2k+1}-1}{\rho-1}, |V|)$.

Proof First, we prove that the k-nearest nodes (kNNs) of a query node q must appear in the nearest k layers of the graph in Fig. 3, which is similar to the breadth-first search (BFS)

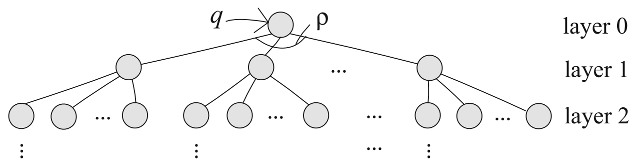


Fig. 3 Illustration for the proof of Lemma 7

traversal tree from q on any graph $\mathcal{G}$. Assume, to the contrary, that one node n located outside those k layers is one of q's kNNs. Then, there must be at least k nodes o (different to q and n) on the shortest path from q to n. Thus, the shortest path distances from q to those k nodes o are certainly smaller than that between q and n, regardless of the edge weights. Hence, node n cannot be one of q's kNNs, which contradicts our assumption. Similarly, kNNs of any node in V are within its closest k layers on the BFS traversal tree in Fig. 3. Based on Heuristic 1, N_g contains all the objects o' that may be answer objects and o''s all possible nearest neighbor data nodes. Thus, N_g at most contains the first $2k$ layers of nodes in Fig. 3, i.e., $|N_g| \leq \sum_{i=0}^{2k} \rho^i = \frac{\rho^{2k+1}-1}{\rho-1}$. Considering that $N_g \subseteq V$, we have $|N_g| \leq \min(\frac{\rho^{2k+1}-1}{\rho-1}, |V|)$. The proof completes. □

It is important to note that the larger the value of $\frac{|V|-|N_g|}{|V|}$ is, the more effective the GSP is. Take the real co-authorship network DBLP with 1,226,749 nodes and 5,557,805 edges as an example. Let us set the node degree ρ to the average node degree 4.5 and k to 2. When data density $d = 1$, $|N_g(\mathrm{DBLP})| \leq \frac{4.5^5-1}{4.5-1} \approx 527$. Clearly, the size of nodes in an essential uncertain graph is much smaller than the node set size in an original uncertain graph. Moreover, the lower bound of the effectiveness of GSP heuristic for DBLP can be approximated as $\frac{|V|-527}{|V|} \approx 99.96\%$, meaning that at least 99.96% nodes in DBLP can be pruned by GSP heuristic for an UG-RkNN query.

It is worthwhile to mention that the derived bound in Lemma 7 holds without any assumption on the probability mass function of edges, other than the positive edge weights. Note that the positive weights guarantee that for a given node n, the shortest path distances from n to the accessed nodes are non-decreasing with the extending network. It indicates that all of n's k-nearest neighbors could be obtained in the k-layer extension of the network from n (if $d = 1$). Further, if the edges in the graph share the same weight, the $\lceil \log_\rho k \rceil$-layer extension from node n is sufficient for getting n's k nearest neighbors. In this case, $|N_g|$ could be tightened as $\min(\frac{\rho^{2\lceil \log_\rho k \rceil+1}-1}{\rho-1}, |V|)$.

On the other hand, the general assumption with positive weights is the main reason that the derived bound is mostly likely to be loose, even to $|V|$ (i.e., the original node size). Fortunately, GSP can prune away many nodes to reduce significantly both the size and the number of possible graphs we have to evaluate during UG-RkNN search, as demonstrated via our experimental results in Sect. 7.

4 Equivalent graph removing

In Sect. 3, we present GSP heuristic whose purpose is to replace an uncertain graph $\mathcal{G}$ with a much smaller essential uncertain graph $\mathcal{G}_e$ for supporting UG-RkNN search efficiently. Although $\mathcal{G}_e$ has successfully reduced the node size from $|V|$ to $|N_g|$, the number of possible graphs we have to consider w.r.t. $\mathcal{G}_e$ might be still large, i.e., in total $\Pi_{e\in E'}|\Omega(e)|$ possible graphs with $E' = \{e_{i,j} \in E | n_i, n_j \in N_g\}$ (i.e., E' is the edge set of $\mathcal{G}_e$). In this section, we present another pruning technique, i.e., *Equivalent Graph Removing* (EGR), which tries to reduce the number of possible graphs via removing *equivalent possible graphs*. In the sequel, we first explain the concept of *equivalent possible graph*, and then, we propose a new EGR heuristic based on this new concept to save the evaluation of certain possible graphs.

Before we formally define the concept of equivalent possible graph, we first present an intuitive example to explain why certain possible graphs are *equivalent*. To simplify our discussion, we introduce an edge set $E^{\text{sp}}_{G_i}$ for a given possible graph $G_i = (V, E, W) \in \Phi(\mathcal{G}_e)$ that consists of two subsets $E^{\text{sp}_q}_{G_i}$ and $E^{\text{sp}_n}_{G_i}$, defined in Eqs. 6 and 7, respectively. The edge set $E^{\text{sp}_q}_{G_i}$ refers to the set of edges that present in the shortest path from q to any node in V on G_i; and $E^{\text{sp}_n}_{G_i}$ denotes the set of edges that appear in the shortest paths from node n in V to its k-nearest data nodes on graph G_i (i.e., those nodes returned by function $k\text{NN}(G_i, n, S)$).

$$E^{\text{sp}}_{G_i} = E^{\text{sp}_q}_{G_i} \bigcup E^{\text{sp}_n}_{G_i} \tag{5}$$

$$E^{\text{sp}_q}_{G_i} = \bigcup_{n\in V} \{e \mid e \in SP_{G_i}(n, q)\} \tag{6}$$

$$E^{\text{sp}_n}_{G_i} = \bigcup_{n\in V} \{e \mid e \in SP_{G_i}(n, o), o \in k\text{NN}(G_i, n, S)\} \tag{7}$$

Take the graph G_i shown in Fig. 4a as an example. When d =1 (i.e., $S = \{n_1, n_2, n_3, n_4, n_5\}$), for $k = 1$, we can get $E^{\text{sp}_q}_{G_i} = \{e_{1,3}, e_{1,2}, e_{3,4}, e_{3,5}\}$ and $E^{\text{sp}_n}_{G_i} = \{e_{1,3}, e_{1,2}, e_{4,5}\}$. Note that an edge might belong to both $E^{\text{sp}_n}_{G_i}$ and $E^{\text{sp}_q}_{G_i}$ (e.g., edges $e_{1,3}$ and $e_{1,2}$). For simplicity, we only mark them once in Fig. 4.

The reason we introduce the edge set $E^{\text{sp}}_{G_i}$ is that $E^{\text{sp}}_{G_i}$ defines the part of the graph that is traversed during RkNN search. Given two possible graphs G_i and G_j (e.g., G_i and G_j depicted in Fig. 4) satisfying the three conditions of $E^{\text{sp}}_{G_i} = E^{\text{sp}}_{G_j}$ (as stated in Lemma 8), it is confirmed that they share the same result set to RkNN retrieval.

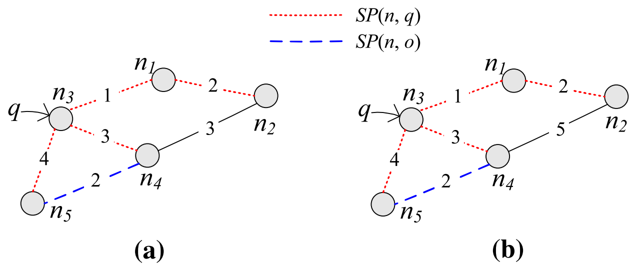


Fig. 4 Illustration of equivalent possible graphs. **a** G_i. **b** G_j

Lemma 8 *Given two distinct possible graphs G_i and G_j w.r.t. an uncertain graph, for the RkNN query issued at a node q, it is confirmed that $E^{\text{sp}}_{G_i} = E^{\text{sp}}_{G_j} \Rightarrow \text{R}k\text{NN}(G_i, q) = \text{R}k\text{NN}(G_j, q)$, where $E^{\text{sp}}_{G_i} = E^{\text{sp}}_{G_j}$ holds iff (i) $\forall e \in E^{\text{sp}}_{G_i}$, $e \in E^{\text{sp}}_{G_j}$; (ii) $\forall e \in E^{\text{sp}}_{G_j}$, $e \in E^{\text{sp}}_{G_i}$; and (iii) $\forall e \in E^{\text{sp}}_{G_i}/E^{\text{sp}}_{G_j}$, $\omega_i(e) = \omega_j(e)$.*

Proof For any object $o \in G_i/G_j$, we know that all edges on $SP_{G_i}(o, q)$ / $SP_{G_j}(o, q)$ and $SP_{G_i}(o, o')/SP_{G_j}(o, o')$ ($o' \in k\text{NN}(G_i, o, S)/k\text{NN}(G_j, o, S)$) are included in the set $E^{\text{sp}}_{G_i}/E^{\text{sp}}_{G_j}$ based on Eq. 5. As $E^{\text{sp}}_{G_i} = E^{\text{sp}}_{G_j}$, we can derive that $d_{G_i}(o, q) = d_{G_j}(o, q)$, $k\text{NN}(G_i, o, S) = k\text{NN}(G_j, o, S)$, and thus, $d^k_{G_i}(o, S) = d^k_{G_j}(o, S)$. Hence, we are certain that $o \in \text{R}k\text{NN}(G_i, q) \Rightarrow o \in \text{R}k\text{NN}(G_j, q)$, and $o \in \text{R}k\text{NN}(G_j, q) \Rightarrow o \in \text{R}k\text{NN}(G_i, q)$. Consequently, $\text{R}k\text{NN}(G_i, q) = \text{R}k\text{NN}(G_j, q)$, and the proof completes. □

We would like to highlight that the third condition of $E^{\text{sp}}_{G_j} = E^{\text{sp}}_{G_i}$ is necessary for Lemma 8. Take the possible graphs G_x and G_y in Fig. 5 as example; they satisfy the first two conditions of $E^{\text{sp}}_{G_x} = E^{\text{sp}}_{G_y}$ (which includes both of the two edges). Nevertheless, the two possible graphs G_x and G_y have different RkNN sets, since $\text{R}k\text{NN}(G_x, q) = \varnothing$ and $\text{R}k\text{NN}(G_y, q) = \{n\}$. In other words, only the first two conditions of $E^{\text{sp}}_{G_j} = E^{\text{sp}}_{G_i}$ cannot guarantee the identical RkNN result set, and thus, the third condition is necessary.

Notably, given a possible graph G_i and its RkNN result set, we can safely ignore all the possible graphs G_j with $E^{\text{sp}}_{G_j} = E^{\text{sp}}_{G_i}$ based on Lemma 8. Accordingly, we introduce the concept of *equivalent possible graph* in Definition 6. By removing all the equivalent possible graphs, we can further reduce the number of possible graphs we need to evaluate to improve query efficiency.

Definition 6 (Equivalent possible graph) Given a possible graph $G_i \in \Phi(\mathcal{G}_e)$, its equivalent possible graphs, denoted as D_{G_i}, include all the other possible graphs $G_j \in \Phi(\mathcal{G}_e)$ with $E^{\text{sp}}_{G_i} = E^{\text{sp}}_{G_j}$, i.e., $\text{D}_{G_i} = \{G_j(\neq G_i) \in \Phi(\mathcal{G}_e) \mid E^{\text{sp}}_{G_j} = E^{\text{sp}}_{G_i}\}$.

Although we understand that we can remove equivalent possible graphs D_{G_i} w.r.t. a given possible graph G_i, we also

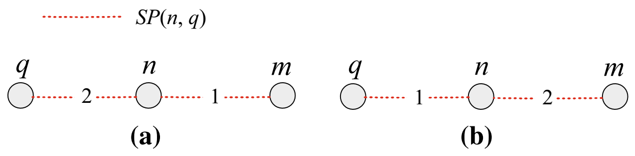


Fig. 5 Explanation for Lemma 8. **a** G_x. **b** G_y

realize that locating the edge set $E^{sp}_{G_j}$ on any possible graph $G_j \in \mathrm{D}_{G_i}$ is as expensive as finding RkNN result on G_j. In other words, we are not able to save anything via identifying and then removing equivalent possible graphs. We need to find a better way to utilize the findings that certain possible graphs actually share the same RkNN result set and evaluating one of them is sufficient. That is why we introduce a stronger dominance relationship, namely *conditional dominance*, between two possible graphs, as formally defined in Definition 7.

Definition 7 (Graph conditional dominance) For two possible graphs G_i and G_j w.r.t. an uncertain graph $\mathcal{G} = (V, E, \Omega, P)$, with a given set $E_d \subseteq E$, graph G_i E_d-dominates graph G_j conditionally, denoted as $G_i \prec_{E_d} G_j$, iff (i) $\forall e \in E_d, \omega_{G_i}(e) = \omega_{G_j}(e)$; (ii) $\forall e' \in (E - E_d), \omega_{G_i}(e') \leq \omega_{G_j}(e')$; and (iii) $\exists e'' \in (E - E_d), \omega_{G_i}(e'') < \omega_{G_j}(e'')$.

We would like to highlight that conditional dominance relationship is stronger than the dominance relationship introduced in Definition 5. For two possible graphs G_i and G_j and a given edge set E_d, if $G_i \prec_{E_d} G_j$, then $G_i \prec G_j$. However, the reverse is not always valid. Take the possible graphs shown in Fig. 2 as an example. $G^{\min}$ dominates G_1. On the other hand, it E_d-dominates G_1 conditionally for $E_d = \{e_{1,2}, e_{1,3}\}$, but not for $E_d = \{e_{1,2}, e_{2,4}\}$. In fact, when E_d is empty, conditional dominance relationship is equivalent to dominance relationship, i.e., $\prec_\varnothing \Leftrightarrow \prec$.

As stated in Lemma 9, using conditional dominance relationship based on the edge set $E^{sp}_{G_i}$, we can locate a set of possible graphs $\mathrm{D}^c_{G_i}$ with each $G_j \in \mathrm{D}^c_{G_i}$ having its $E^{sp}_{G_j} = E^{sp}_{G_i}$, i.e., $\mathrm{D}^c_{G_i} \subseteq \mathrm{D}_{G_i}$. Note that locating equivalent possible graphs in $\mathrm{D}^c_{G_i}$ does not require us to identify the edge set $E^{sp}_{G_j}$ of any possible graph $G_j \in \mathrm{D}^c_{G_i}$.

Lemma 9 *Given a possible graph $G_i \in \Phi(\mathcal{G}_e)$ with set $E^{sp}_{G_i}$ (w.r.t. a query node q), let $\mathrm{D}^c_{G_i}$ be the set of possible graphs G_j that are conditionally dominated by G_i based on $E^{sp}_{G_i}$, i.e., $\mathrm{D}^c_{G_i} = \{G_j \in \Phi(\mathcal{G}_e) \mid G_i \prec_{E^{sp}_{G_i}} G_j\}$. It is confirmed that $\mathrm{D}^c_{G_i} \subseteq \mathrm{D}_{G_i}$.*

Proof For any graph $G \in \mathrm{D}^c_{G_i}$, according to the definitions of $\mathrm{D}^c_{G_i}$ and graph conditional dominance, edges in $E^{sp}_{G_i}$ share the same weights on G and G_i, and edges in $(E - E^{sp}_{G_i})$ have larger or equal weights on G than/to those on G_i. Hence, we have $E^{sp}_G = E^{sp}_{G_i}$ based on Eq. 5. Referring to D_{G_i} formalized in Definition 6, we can confirm that graph $G \in \mathrm{D}_{G_i}$. Thus, $\mathrm{D}^c_{G_i} \subseteq \mathrm{D}_{G_i}$, and the proof completes. □

Take the possible graphs G_i and G_j depicted in Fig. 4 as an example. It is clear that $G_j \in \mathrm{D}^c_{G_i}$ and $G_j \in \mathrm{D}_{G_i}$, and meanwhile $G_i \in \mathrm{D}_{G_j}$ but $G_i \notin \mathrm{D}^c_{G_j}$.

We are now ready to present *equivalent graph removing* (EGR) technique in Heuristic 2. Its correctness is guaranteed by Lemma 8 and Lemma 9. EGR enables us to skip the evaluation of certain possible graphs and thus to boost search performance.

Heuristic 2 (Equivalent graph removing) *Given a possible graph $G_i \in \Phi(\mathcal{G}_e)$, a data node set S, and a query node q, it is confirmed that the possible graphs $G_j \in \mathrm{D}^c_{G_i} \subseteq \Phi(\mathcal{G}_e)$ have the identical RkNN result set as G_i, i.e.,* RkNN$(G_i, q) =$ RkNN(G_j, q). *Hence, those graphs G_j in $\mathrm{D}^c_{G_i}$ can be safely removed after evaluating RkNN search on G_i, with* Pr(o) *of each answer object $o \in$* RkNN(G_i, q) *updated accordingly.*

Next, we explain how to update Pr(o) of each answer object $o \in$ RkNN(G_i, q), when a possible graph G_j in $\mathrm{D}^c_{G_i}$ is removed, as mentioned in Heuristic 2. In particular, given two possible graphs G_i and G'_i, set $\mathrm{D}^c_{G_i}$ might overlap with set $\mathrm{D}^c_{G'_i}$, i.e., $\mathrm{D}^c_{G_i} \cap \mathrm{D}^c_{G'_i} \neq \varnothing$. This is because they both might conditionally dominate the same possible graph even though they do not conditionally dominate each other. Thus, given an answer object $o \in$ RkNN(G_i, q), for every possible graph $G_j \in \mathrm{D}^c_{G_i}$, the existence probability of G_j (i.e., Pr(G_j)) is added to Pr(o) (because $o \in$ RkNN(G_j, q)). However, Pr(G_j) contributes *once* only to Pr(o). That is to say, given an answer object $o \in$ RkNN$(G_a, q) \cap$ RkNN(G_b, q) and a possible graph $G_j \in \mathrm{D}^c_{G_a} \cap \mathrm{D}^c_{G_b}$, if Pr$(G_j)$ has been added to Pr(o) during the evaluation of G_a, it does not contribute to Pr(o) again during the evaluation of G_b.

It is important to note that if possible graphs $G \in \Phi(\mathcal{G}_e)$ are evaluated in a random order, it is very likely that the equivalent possible graphs in D^c_G are evaluated earlier than G, which significantly degrades the effectiveness of our presented EGR heuristic. Thus, in order to make sure that graph $G \in \Phi(\mathcal{G}_e)$ is evaluated earlier than graphs in D^c_G, we introduce the *incremental access strategy* (IAS) to decide the evaluation order of possible graphs in $\Phi(\mathcal{G}_e)$. For ease of understanding, we take the sample graph as an example to explain IAS strategy as follows.

First of all, before evaluating any possible graph, IAS sorts the edges in a random order, e.g., edge $e_{1,2}$ ranked first and edge $e_{1,3}$ ranked second as illustrated in the first row of Table 3. Then, it assigns all the edges to their minimum weights, and visits the corresponding possible graph g_1 (i.e., $G^{\min}$) first. Based on g_1, it increases the weight of the first edge to generate the second possible graph for evaluation, i.e., g_2 in Table 3. As edge $e_{1,2}$ has only two possible weights 2

Table 3 Illustration of the incremental access strategy

Graph	$\omega(e_{1,2})$	$\omega(e_{1,3})$	$\omega(e_{2,4})$	$\omega(e_{2,6})$	$\omega(e_{2,7})$	$\omega(e_{3,4})$	$\omega(e_{3,5})$	$\omega(e_{4,5})$	$\omega(e_{4,6})$	$\omega(e_{5,8})$
g_1	2	1	3	3	2	3	4	2	2	1
g_2	3	1	3	3	2	3	4	2	2	1
g_3	2	4	3	3	2	3	4	2	2	1
g_4	3	4	3	3	2	3	4	2	2	1
g_5	2	1	5	3	2	3	4	2	2	1

Algorithm 3 Equivalent Graph Removing (EGR)

```
Input: an essential uncertain graph $\mathcal{G}_e$, a data node set $S$, a query node $q$, and parameter $k$
Output: a data node set $S$ with R$k$NN probabilities
1: initialize equivalent possible graph set $\Gamma_{D^c} \leftarrow \varnothing$
2: for each possible graph $G \in \Phi(\mathcal{G}_e) - \Gamma_{D^c}$ do // using IAS strategy
3:     $\Pr(G) \leftarrow \prod_{e \in E} p(e, w_G(e))$     // by Eq. 1
4:     R$k$NN$(G, q) \leftarrow E_G^{sp} \leftarrow \varnothing$, $H \leftarrow \{\langle n(q), 0\rangle\}$
5:     while $H$ is not empty do
6:         $\langle n, d_G(n, q)\rangle \leftarrow$ de-heap$(H)$
7:         if $n.flag = 1$ then continue
8:         $n.flag \leftarrow 1$
9:         insert edges on $SP_G(o \in k\mathrm{NN}(G, n, S), n)$ into $E_G^{sp}$
10:         if node $n$ has a data node $s$ with $d_G(s, q) \le d_G^k(n, S)$ then
11:             R$k$NN$(G, q) \leftarrow$ R$k$NN$(G, q) \cup \{s\}$
12:             $\Pr(s) \leftarrow \Pr(s) + \Pr(G)$
13:         if $d_G(n, q) \le d_G^k(n, S)$ then
14:             for each adjacent node $n_i$ of $n$ with $n_i.flag = 0$ do
15:                 en-heap$\langle n_i, d_G(n, q) + d_G(n, n_i)\rangle$
16:                 $E_G^{sp} \leftarrow E_G^{sp} \cup \{\overrightarrow{nn_i}\}$
17:     $D_G^c \leftarrow \{G_j \in \Phi(\mathcal{G}_e) | G \prec_{E_G^{sp}} G_j\}$     // Lemma 9
18:     $\Pr(s) \leftarrow \Pr(s) + \sum_{G' \in (D_G^c - \Gamma_{D^c})} \Pr(G')$ for each $s \in$ R$k$NN$(G, q)$
19:     $\Gamma_{D^c} \leftarrow \Gamma_{D^c} \bigcup D_G^c$
20: return $S$
```

and 3 (shown in Fig. 1), g_1 and g_2 are the only two graphs generated by changing the weights of edge $e_{1,2}$, with the weights of edges ranked behind $e_{1,2}$ all set to the minimum. Based on g_1 and g_2, it increases the weight of the second edge to generate two more graphs g_3 and g_4 for evaluation. Similarly, based on g_1, g_2, g_3, and g_4, it increases the weight of the third edge to generate another four more graphs for evaluation. The above process proceeds until the last graph $G^{\max}$ is generated and evaluated. Note that IAS guarantees that graph $G \in \Phi(\mathcal{G}_e)$ is accessed before those in D_G^c, and hence, all the RkNN evaluation of equivalent possible graphs in $\cup_{G \in \Phi(\mathcal{G}_e)} D_G^c$ could be saved.

Based on Heuristic 2 and IAS strategy, we detail the implementation of EGR, with its pseudo-code presented in Algorithm 3. First, EGR empties Γ_{D^c}, the set of equivalent possible graphs (line 1). Then, it evaluates the possible graphs in $(\Phi(\mathcal{G}_e) - \Gamma_{D^c})$ based on the strategy IAS described previously (lines 2–19). To be more specific, for each possible graph G it visits, EGR derives the existence probability $\Pr(G)$ by Eq. 1, empties RkNN set and E_G^{sp} set, and inserts a query node q into heap H (lines 3–4). The evaluation on G is performed by evaluating the nodes in H. For each de-heaped node n, it is only evaluated once which explains the reason behind $n.flag$. It marks the node n by setting $n.flag$ and inserts the edges on the shortest paths $SP_G(o \in k\mathrm{NN}(G, n, S), n)$ into E_G^{sp} set (lines 5–9). If n contains a data node s satisfying $d_G(s, q) \le d_G^k(s, S)$, s is added to RkNN set and its probability of being an answer object $\Pr(s)$ is updated accordingly. In addition, if n satisfies $d_G(n, q) \le d_G^k(n, S)$, EGR en-heaps its unmarked adjacent nodes into H, and inserts corresponding adjacent edges into E_G^{sp} (lines 10–16). The above process repeats until H is empty. Then, EGR updates Γ_{D^c} and the probability $\Pr(s)$ for every $s \in$ RkNN(G, q), and visits the next possible graph (lines 17–19). Until all the possible graphs in $(\Phi(\mathcal{G}_e) - \Gamma_{D^c})$ have been evaluated, EGR returns set S with RkNN probabilities.

Example 4 In order to facilitate the understanding of EGR, we illustrate how to derive $D_{G\min}^c$ for $G^{\min}$ depicted in Fig. 2c during UG-R1NN query processing. Table 4 lists the content of an edge set $E_{G\min}^{sp}$ and that of a heap H. First, EGR de-heaps the query node n_3 from H, and inserts edge $e_{3,4}$ into $E_{G\min}^{sp}$, since it presents on the shortest path from n_3 to its nearest neighbor $s_2(n_4)$. Then, EGR en-heaps its unmarked adjacent nodes n_1, n_4, and n_5 into H, and updates $E_{G\min}^{sp}$ to $\{e_{1,3}, e_{3,4}, e_{3,5}\}$. In this way, EGR continues evaluating the remaining nodes in H. Finally, it forms the set $E_{G\min}^{sp} = \{e_{1,3}, e_{3,4}, e_{3,5}, e_{1,2}, e_{4,6}, e_{2,7}, e_{4,5}\}$. Thus, $D_{G\min}^c$ contains 7 possible graphs on which at least one of the edges $e_{2,4}$, $e_{2,6}$, and $e_{5,8}$ has larger weight than that on $G^{\min}$ according to Lemma 9.

5 Exact algorithm

In this section, we propose an exact algorithm, termed as SDP, for processing UG-RkNN search. Note that the algorithm utilizes three key techniques, including graph structure pruning (GSP) and equivalent graph removing (EGR) presented in previous two sections, and *probability bound pruning* (PBP) discussed later. In what follows, we first present PBP

Table 4 Procedure of equivalent graph removing

Object	$E^{\text{sp}}_{G^{\min}}$	H
$\langle n_3(q), 0\rangle$	$\{e_{1,3}, e_{3,4}, e_{3,5}\}$	$\langle n_1, 1\rangle, \langle n_4, 3\rangle, \langle n_5, 4\rangle$
$\langle n_1, 1\rangle$	$\{e_{1,3}, e_{3,4}, e_{3,5}, e_{1,2}\}$	$\langle n_4, 3\rangle, \langle n_2, 3\rangle, \langle n_5, 4\rangle$
$\langle n_4, 3\rangle$	$\{e_{1,3}, e_{3,4}, e_{3,5}, e_{1,2}, e_{4,6}\}$	$\langle n_2, 3\rangle, \langle n_5, 4\rangle$
$\langle n_2, 3\rangle$	$\{e_{1,3}, e_{3,4}, e_{3,5}, e_{1,2}, e_{4,6}, e_{2,7}\}$	$\langle n_5, 4\rangle$
$\langle n_5, 4\rangle$	$\{e_{1,3}, e_{3,4}, e_{3,5}, e_{1,2}, e_{4,6}, e_{2,7}, e_{4,5}\}$	$\varnothing$

technique and then detail our SDP algorithm. As stated in Heuristic 3, PBP tries to reduce the data node validation cost for UG-RkNN retrieval.

Heuristic 3 (Probability bound pruning) *Given any possible graph $G_i \in \Phi(\mathcal{G}_e)$, a query node q, and a specified parameter θ, let $\overline{P}$ be the total existence probability of remaining unvisited possible graphs that have not been removed by* EGR *(see Heuristic 2). For a data node $o \in$ RkNN(G_i, q), if o satisfies $\Pr_{G_i}(o) \geq \theta$ or $\Pr_{G_i}(o) + \overline{P} < \theta$, then o can be pruned during UG-RkNN search. Here, $\Pr_{G_i}(o)$ denotes the probability of o to be an answer object based on currently evaluated possible graph.*

Proof The proof is intuitive, and thus, it is omitted. □

It is worth mentioning that $\Pr_{G_i}(o)$ and $\Pr_{G_i}(o) + \overline{P}$ are actually the lower bound and the upper bound of $\Pr(o)$ (i.e., o's probability to become an answer object), respectively. The fact $\Pr_{G_i}(o) \geq \theta$ means that o is definitely an answer object, and the fact $\Pr_{G_i}(o) + \overline{P} < \theta$ indicates that o is definitely not an answer object.

Our SDP algorithm to support UG-RkNN query processing is depicted in Algorithm 4. To begin with, SDP initializes the RkNN probability $\Pr(s)$ as zero for each data object $s \in S$. Then, it utilizes GSP heuristic to derive the essential uncertain graph $\mathcal{G}_e$ (line 2) in order to effectively cut down the size of the uncertain graph and thus to reduce the number of possible graphs. Based on $\mathcal{G}_e$, SDP invokes EGR (i.e., Algorithm 3), and integrates PBP heuristic to evaluate possible graphs and update $\Pr(s)$ for every data node $s \in S$ accordingly (line 3). Note that PBP can be easily integrated into EGR algorithm. Every time when the total existence probability $\overline{P}$ or any $\Pr_{G_i}(o)$ changes, PBP is used to validate/invalidate data nodes for UG-RkNN search. Finally, SDP retrieves the data objects $s \in S \cap N_c$ with $\Pr(s) \geq \theta$ to complete the process.

Time complexity of SDP First, the traversal of a graph takes $O(|V| \cdot \lg |V| + |E|)$ time, assuming that Dijkstra-alike algorithm is used [16,29]. Thus, GSP heuristic (i.e., Algorithm 2) takes $O(|V|^2 \cdot \lg |V| + |V| \cdot |E|)$ time. Second, the time complexity of EGR heuristic (i.e., Algorithm 3) is $O(w^{|E|} \cdot (|V|^2 \cdot \lg |V| + |V| \cdot |E|))$ in the worst case, as there are $w^{|E|}$ possible graphs if all the edges share the same number of weights w (i.e., $\forall e \in E$, $|\Omega(e)| = w$). In addition, PBP needs $O(w^{|E|} \cdot |V|)$ time, since the number of data objects is at most $|V|$ for every possible graph. Consequently, in the worst case, the total time complexity of SDP is $O(w^{|E|} \cdot (|V|^2 \cdot \lg |V| + |V| \cdot |E|))$.

Algorithm 4 SDP Algorithm
```
Input: an uncertain graph $\mathcal{G}$, a data node set $S \subseteq V$, a query node $q$, and parameters $k$, $\theta$
Output: the result set $S_G$ of an UG-RkNN query
1: initialize $\Pr(s \in S) \leftarrow 0$
2: $\langle \mathcal{G}_e, N_c\rangle \leftarrow$ GSP($\mathcal{G}, S, q, k$)    // Algorithm 2
3: $S \leftarrow$ EGR($\mathcal{G}_e, S, q, k$) //integrating PBP in Algorithm 3
4: insert data nodes $s \in S \cap N_c$ with probability $\Pr(s) \geq \theta$ into $S_G$
5: return $S_G$
```

Although SDP has significantly improved search performance compared with Baseline, its cost is still high. In order to further boost the performance of UG-RkNN search, we propose an adaptive sampling algorithm to be introduced in the next section.

6 Sampling algorithm

Due to the ♯P-hard complexity of UG-RkNN search as discussed in Sect. 2, there is no algorithm to exactly solve the UG-RkNN query in polynomial time. In view of this, we present an efficient algorithm, termed as TripS, which uses graph structure pruning and an adaptive stratified sampling technique, for UG-RkNN search. In particular, the framework of TripS is given in Algorithm 5. It first employs GSP (Algorithm 2) to get the essential uncertain graph (line 2). Second, on the top of essential uncertain graph, TripS conducts RkNN search on every sampled possible graph, captured by the adaptive stratified sampling technique (ASSP), for UG-RkNN retrieval (line 3). Note that the final result set contains the objects having the probabilities not smaller than $\theta' (= \theta \cdot \sum_{G \text{ is sampled}} \Pr(G))$. Given the fact that GSP heuristic has been introduced in Sect. 3, we only focus on ASSP sampling technique in this section.

Before we present ASSP, we first explain the popular metric used to evaluate the performance of the sampling algorithms. Assume that an sampling algorithm $\hat{F}$ is the estimator of true value F, an essential metric to evaluate the accuracy of sampling approach is the *mean squared*

Algorithm 5 TripS Algorithm

```
Input: an uncertain graph $\mathcal{G}$, a data node set $S$, a query node $q$, $k$, $\theta$, the parameters used in the sampling technique including the sorted edge set $E_s$, sample size $N$, threshold of sample size $N_t$, and tunable parameters $\gamma_1$, $\gamma_2$
Output: the result set $S_G$ of an UG-RkNN query
1: initialize $Pr(s \in S) \leftarrow 0$
2: $\langle \mathcal{G}_e, N_c \rangle \leftarrow$ GSP($\mathcal{G}, S, q, k$)    // Algorithm 2
3: $S \leftarrow$ ASSP($\mathcal{G}_e$, $S$, $E_s$, $q$, empty string $X$, $N$, $N_t$, $k$, $\gamma_1$, $\gamma_2$) // Algorithm 6
4: insert data nodes $s \in S \cap N_c$ with probability $Pr(s) \geq \theta'$ into $S_G$
5: return $S_G$
```

error (MSE) [16,24,25], denote as $\mathbb{E}[(\hat{F} - F)^2]$, which measures the expected difference between an estimator and the true value. It can be decomposed into two parts [45], i.e., $\mathbb{E}[(\hat{F} - F)^2] = \text{Var}(\hat{F}) + [\mathbb{E}(\hat{F}) - F]^2 = \text{Var}(\hat{F}) + [\text{Bias}(\hat{F}, F)^2]$. Here, $\text{Var}(\hat{F})$ represents the average deviation from its expectation. It is easy to find that for an unbiased estimator, the variance is simply the MSE. In other words, the key criterion to discriminate the estimators is their variances, if they are unbiased.

We use a naive *unbiased random sampling* method, i.e., Monte Carlo sampling [16,24,25] for UG-RkNN search. Specifically, Monte Carlo first draws N possible graphs, denoted as $G_1, G_2, \ldots, G_N$ from $\mathcal{G}$, and then, it finds RkNN objects for each possible graph G_i. Finally, it obtains Monte Carlo estimator by summarizing the RkNN results on the sampled possible graphs.

Different from the naive Monte Carlo estimator, the stratified sampling is an effective and popular sampling technique to tackle query processing [21,24,25]. It partitions the entire population into disjoint strata, and then represents every stratum by capturing specified number of samples in that stratum. Its main advantage is to capture the most representatives of a population. As a result, in TripS algorithm, we propose a new adaptive stratified sampling technique, namely ASSP, for UG-RkNN search, which is proved to be unbiased and achieves a variance at least as good as Monte Carlo sampling does.

Before we formally introduce the detailed ASSP technique, we first present an illustrative example to explain the basic idea of ASSP technique. To simplify discussion, we assume that an uncertain graph $\mathcal{G}$ has four edges e_1, e_2, e_3, and e_4, and every edge has two distinct weights (denoted as v_a and v_b), i.e., $\forall e$, $|\Omega(e)| = w = 2$. The weight of an edge might be different from the weight of another edge. Then, there are in total 2^4 possible graphs w.r.t. $\mathcal{G}$. In ASSP, these possible graphs (i.e., the whole sample space) are divided into several sample subspaces, and the specified number of possible graphs are captured from each sample space. As shown in Table 5, assume that the 16 possible graphs $G_1, G_2, \ldots, G_{16}$ are partitioned into 4 strata Δ_1, Δ_2, Δ_3, and Δ_4, according to the weights of edges e_1 and e_2, and the total sample size N is specified as 8. For simplification, we assume ASSP randomly captures 2 out of 4 possible graphs in every stratum in Table 5. In other words, we utilize 8 sampled possible graphs to answer UG-RkNN query approximately, instead of exploring all 16 possible graphs.

Table 5 Illustration of stratified sampling on uncertain graphs

Sample space	Possible graph	$\omega(e_1)$	$\omega(e_2)$	$\omega(e_3)$	$\omega(e_4)$
Δ_1	G_1	v_a	v_a	v_a	v_a
	G_2	v_a	v_a	v_a	v_b
	G_3	v_a	v_a	v_b	v_a
	G_4	v_a	v_a	v_b	v_b
Δ_2	G_5	v_a	v_b	v_a	v_a
	G_6	v_a	v_b	v_a	v_b
	G_7	v_a	v_b	v_b	v_a
	G_8	v_a	v_b	v_b	v_b
Δ_3	G_9	v_b	v_a	v_a	v_a
	G_{10}	v_b	v_a	v_a	v_b
	G_{11}	v_b	v_a	v_b	v_a
	G_{12}	v_b	v_a	v_b	v_b
Δ_4	G_{13}	v_b	v_b	v_a	v_a
	G_{14}	v_b	v_b	v_a	v_b
	G_{15}	v_b	v_b	v_b	v_a
	G_{16}	v_b	v_b	v_b	v_b

Consequently, we are aware that sample allocation and stratum partition are two critical steps for ASSP. To begin with, in ASSP, we employ the proportional sample allocation (PSA) [38]. Specifically, given the total sample size N, for achieving proportional samples, we assign sample size N_i for each stratum Δ_i proportional to the population probability of the stratum. The population probability, denoted as π_i, w.r.t. a subspace Δ_i, accumulates the existence probabilities of all the possible graphs included in Δ_i, i.e.,

$$\pi_i = \sum_{G_t \in \Delta_i} \Pr(G_t) \tag{8}$$

where $\Pr(G_t)$ is the existence probability of possible graph G_t. Specifically, the sample size N_i is defined by Eq. 9.

$$N_i = \frac{\sum_{G_t \in \Delta_i} \Pr(G_t)}{\sum_{G_t \in \Phi(\mathcal{G})} \Pr(G_t)} \cdot N = \pi_i \cdot N \tag{9}$$

Note that the total probability of all possible graphs (included in $\Phi(\mathcal{G})$) is equal to one. As a result, strata with large population probabilities should be sampled heavily, whereas strata with small population probabilities deserve a relatively small sample size. It is important to point out that a naive allocation method is to assign sample size N_i by $\frac{L_i}{L} \cdot N$, where L_i is the number of possible graphs contained in stratum Δ_i, and L

Table 6 Illustration of partition based on $\gamma = 2$ edges

Sample space	$\omega(e_1)$	$\omega(e_2)$	Possible graphs
Δ'_1	v_a	v_a	G_1, G_2, G_3, G_4
Δ'_2	v_a	v_b	G_5, G_6, G_7, G_8
Δ'_3	v_b	$\star$	$G_9, G_{10}, G_{11}, G_{12}, G_{13}, G_{14}, G_{15}, G_{16}$

Table 7 Illustration of partition based on $\gamma = 3$ edges

Sample space	$\omega(e_1)$	$\omega(e_2)$	$\omega(e_3)$	Possible graphs
Δ''_1	v_a	v_a	v_a	G_1, G_2
Δ''_2	v_a	v_a	v_b	G_3, G_4
Δ''_3	v_a	v_b	$\star$	G_5, G_6, G_7, G_8
Δ''_4	v_b	$\star$	$\star$	$G_9, G_{10}, G_{11}, G_{12}, G_{13}, G_{14}, G_{15}, G_{16}$

is the total number of possible graphs. This naive method is inferior to our used method. This is because it allocates sample sizes based on the assumption that every possible graph has the same existence probability (without considering the different probabilities of the possible graphs).

Now, we start to our stratum partition methods in details. To begin with, we select γ edges from the edge set E of an uncertain graph $\mathcal{G}$ to partition the space into w^γ subspaces, assuming that each edge has w distinct weights. Note that w^γ is the largest size of subspaces that could be partitioned by γ edges. For instance, let $\gamma = 2$ in the above example. 2 edges can produce the maximal $w^\gamma = 2^2 = 4$ subspaces. This naive partition is simple but it has limitations. A small change of γ will cause a significant change of the subspace size. Thus, it is hard to achieve a flexible sample space partition. On the other hand, given γ edges, the minimum size of subspaces that could be partitioned by those γ edges is $\gamma \cdot (w - 1) + 1$.

As depicted in Tables 6 and 7, two (three) edges can partition the previous example uncertain graph into 3 (4) subspaces. Note that the notation $\star$ indicates the weight of an edge e is not specified, i.e., all the weights of the edge e could be allowed in the corresponding space. In addition, the 3 subspaces in Table 6 are actually the partition for the whole sample space, since they are disjoint between each other, and the union of them is the whole space. In other words, the $\gamma \cdot (w - 1) + 1$ subspaces that γ edges divide must satisfy that those subspace are disjoint between each other, and the union of them forms the whole space. Not any $\gamma \cdot (w - 1) + 1$ possible graph sets could form a partition of the whole sample space. For example, instead of Δ'_1, Δ'_2,, and Δ'_3 in Table 6, we assume that there are three other possible graph sets (denoted as P_1, P_2, and P_3) w.r.t. $\gamma = 2$, such that P_1 is identified by $\langle e_1(v_a), e_2(v_a)\rangle$, P_2 is labeled by $\langle e_1(v_b), e_2(\star)\rangle$, P_3 is denoted by $\langle e_1(\star), e_2(v_b)\rangle$. Then, these three possible graph sets cannot form a partition of the whole sample space. The reason is that the possible graphs with edge weights $e_1(v_b)$ and $e_2(v_b)$ could belong to P_2 and P_3 simultaneously. In other words, one possible graph may be divided into two possible graph sets, which does not satisfy the disjointness of the partition.

As γ increases, it has a linear impact on $\gamma \cdot (w-1)+1$, the minimum number of subspaces generated. However, we also notice that the partition using $\gamma \cdot (w-1)+1$ renders the diversity of the sampling, since only a small group of subspaces is considered by sampling algorithm, which is not desirable for UG-RkNN retrieval. In order to maximize the advantages of two simple partition approaches above and meanwhile minimize their limitations, ASSP adopts two tunable parameters γ_1 and γ_2. It selects $(\gamma_1 + \gamma_2)$ edges such that the space is partitioned into $w^{\gamma_1} \cdot (\gamma_2 \cdot (w - 1) + 1)$ subspaces. Note that the partition based on $(\gamma_1 + \gamma_2)$ edges is superior to the above two partitions based on γ edges, which will be further confirmed by our experimental studies to be presented in Sect. 7.

Next, we want to emphasize that ASSP is a *recursive* stratified sampling algorithm. Specifically, in every recursion, ASSP chooses $(\gamma_1 + \gamma_2)$ *unselected* edges from the edge set E of an uncertain graph, and partitions the current sample space into $w^{\gamma_1} \cdot (\gamma_2 \cdot (w - 1) + 1)$ subspaces, as discussed above. It is important to note that there are at most $(\gamma_2 - 1)$ edges with unspecified edge weights in the selected $(\gamma_1 + \gamma_2)$ edges. For example, when $\gamma = \gamma_2 = 2(3)$ in Tables 6 and 7, there is one edge (there are two edges) with unspecified weights in the selected γ edges. Without loss of generality, if an edge is selected in one recursion but its weight is still unspecified, it is regarded as an unselected edge in the next recursion. The recursion (or the sample space partition) terminates until the size of E is smaller than $(\gamma_1 + \gamma_2)$, or the allocated sample size w.r.t. the current space is small than a given sample size threshold. When the final smallest subspaces Δ_i are obtained, the Monte Carlo sampling is used to capture $\pi_i \cdot N$ possible graphs within each subspace Δ_i for UG-RkNN search, where N is the total sample size.

In addition, we also want to highlight that in order to achieve more adaptive diversity, ASSP sorts all the edges in an uncertain graph based on the *breadth-first search* (BFS) edge selection strategy that invokes BFS starting from the query

Algorithm 6 Adaptive Stratified Sampling (ASSP)

```
Input: $\mathcal{G}_e$, $S$, edge list $E_s$, $q$, the string representation of the states for sample edges $X$, sample size $N$, threshold of sample size $N_t$, $k$, $\gamma_1$, $\gamma_2$
Output: a data node set $S$ with R$k$NN probabilities
/* lazy-EP($G, S, q, k$): the algorithm proposed in [52] is to return the result set of a R$k$NN query on a deterministic graph $G$. */
1: if $|E_s| < \gamma_1 + \gamma_2$ or $N < N_t$ then
2:   for each $i$=1 to $N$ do
3:     initialize an empty string $Y$
4:     for each edge $e$ in the unsample edges from $E_s$ do
5:       randomly select a weight $\omega(e)$ for $e$
6:       insert the string representation of $\omega(e)$ to $Y$
7:     generate the possible graph $G$ based on $X + Y$
8:     if the graph $G$ has not been sampled yet then
9:       compute $Pr(G)$ by Eq. 1
10:      for each data object $s \in$ lazy-EP($G, S, q, k$) do
11:        $Pr(s) \leftarrow Pr(s) + Pr(G)$
12: else
13:   $E_1 \leftarrow$ {the first unselected $\gamma_1$ edges of $E_s$}
14:   $E_2 \leftarrow$ {the following $\gamma_2$ edges of $E_s$}
15:   let $SV_1$ be the $w^{\gamma_1}$ edge state possibilities for edge set $E_1$
16:   let $SV_2$ be the $(\gamma_2 \cdot (w-1)+1)$ edge state possibilities of $E_2$
17:   for each state vector pair $sv_1, sv_2 \in SV_1 \times SV_2$ do
18:     $Y \leftarrow$ string($sv_1$) + string($sv_2$)
19:     compute the existence probability of $Y$, i.e., $\pi_Y$
20:     $\gamma_1 \leftarrow \gamma_1 - 1$, $\gamma_2 \leftarrow \gamma_2 + 1$,
21:     ASSP($\mathcal{G}_e$, $S$, $E_s$, $q$, $X + Y$, $\pi_Y N$, $N_t$, $k$, max{$\gamma_1$, 0}, $\gamma_2$)
22: return $S$
```

object q to order edges, which will be maintained by an edge list E_s. Hence, in every recursion, ASSP first chooses γ_1 unselected edges from E_s and then another γ_2 unselected edges from E_s. In other words, ASSP actually explores the edges closer to the query node earlier. Furthermore, we decrease the value of γ_1 but increase that of γ_2 with the depth of the recursion so that ASSP actually divides more subspaces w.r.t. the edges closer to the query.

The pseudo-code of ASSP is given in Algorithm 6, which aims to find the candidate data objects with RkNN probabilities, by taking as inputs the essential uncertain graph $\mathcal{G}_e = (V, E, W, P)$, a data node set S, a query object q, an edge list E_s that maintains the edges in E sorted based on *BFS* strategy from q, the string representation of the states (i.e., the weights) for selected edges X, sample size N, sample size threshold N_t, and parameters k, γ_1, γ_2. Specifically, if $|E_s| < \gamma_1 + \gamma_2$ or $N < N_t$, ASSP captures N sample graphs from the sample space related to one stratum denoted by X via Monte Carlo sampling, and updates the RkNN probability of every data object through $lazy\text{-}EP$ algorithm (lines 1–11). Otherwise, ASSP gets the first γ_1 unselected edges from E_s and next γ_2 unselected edges from E_s, maintained by E_1 and E_2, respectively (lines 13–14). It is important to note that the E_s list during sampling is static without edge deletion or insertion. One simple method to identify the unselected edges in E_s is via the comparison between X and E_s where the first edge in E_s but not in X is just the first edge of the γ_1 edges in the current recursion (should be inserted in E_1). Then, for each state vector of those γ_1 edges in E_1 (there are w^{γ_1} state vectors in total) and each state vector of those γ_2 edges in E_2 (there are $\gamma_2 \cdot (w-1)+1$ state vectors in total), we maintain their string representation Y (line 18), and derive the corresponding existence probability π_Y of Y (line 19). After updating $X = X + Y$, $\gamma_1 = \max\{\gamma_1 - 1, 0\}$, $\gamma_2 = \gamma_2 + 1$, and $N = \pi_Y \cdot N$ for the input parameters of ASSP, ASSP is recursively recalled in line 21.

In the last part of this section, we analyze our TripS algorithm (including ASSP technique) and explain how to adapt our algorithms newly proposed to support *bichromatic* UG-RkNN query in order to demonstrate the flexibility and generality of our algorithms.

First, in TripS algorithm, the ASSP sampling technique needs to estimate the population proportion [38], according to UG-RkNN query problem in Definition 4. This is because the probability Pr(o) of every object o to be an actual answer object is actually the proportion of possible graphs G such that o is a real answer object on G. It is guaranteed that the ASSP estimator is unbiased, and the variance of ASSP is at least as good as that of Monte Carlo sampling, as stated in Theorems 1 and 2, respectively.

Theorem 1 *The* ASSP *estimator, denoted as* $\hat{F}_{\text{ASSP}}$, *is unbiased, i.e.,* $\mathbb{E}(\hat{F}_{\text{ASSP}}) = F$.

Proof The detailed proof can be found in Appendix A. □

Lemma 10 *Let N_i be the number of samples in stratum i, L_i be the number of population for stratum i, and T be the total number of strata divided by stratified sampling. The variance of* ASSP, *denoted as Var*($\hat{F}_{\text{ASSP}}$), *can be derived*

$$\text{Var}(\hat{F}_{\text{ASSP}}) = \sum_{i=1}^{T} \pi_i^2 \cdot \frac{L_i - N_i}{L_i - 1} \cdot \frac{\hat{F}_{\text{ASSP}}(1 - \hat{F}_{\text{ASSP}})}{N_i}.$$

Proof The detailed proof can be found in Appendix B. □

Theorem 2 *The variance of* ASSP, *i.e.,* Var($\hat{F}_{\text{ASSP}}$), *is at least as good as that of Monte Carlo sampling, i.e.,* Var($\hat{F}_{\text{MC}}$). *Formally,* Var($\hat{F}_{\text{ASSP}}$) $\leq$ Var($\hat{F}_{\text{MC}}$).

Proof The detailed proof can be found in Appendix C. □

Time complexity of TripS RkNN query for each possible graph takes $O(|V|^2 \cdot \lg|V| + |V| \cdot |E|)$ time. Let N be the total number of samples, since sampling N possible graphs needs at most $O(N \cdot |E|)$ time as discussed in [24], the time complexity of ASSP is $O(N \cdot (|V|^2 \cdot \lg|V| + |V| \cdot |E|))$. Therefore, the time complexity of TripS algorithm is $O(|V|^2 \cdot \lg|V| + |V| \cdot |E|) + O(N \cdot (|V|^2 \cdot \lg|V| + |V| \cdot |E|)) = O(N \cdot (V|^2 \cdot \lg|V| + |V| \cdot |E|))$.

Bichromatic UG-R*k*NN search Bichromatic UG-RkNN query is a powerful tool in many real-life applications, including business decision and resource allocation in complex networks, as explained in Sect. 1. Thus, we further explore *bichromatic* UG-RkNN search, as defined in Definition 8 based on bichromatic RkNN retrieval over deterministic graphs described in Sect. 1.

Definition 8 (Bichromatic UG-RkNN query) Given an uncertain graph $\mathcal{G} = (V, E, \Omega, P)$, two relevant data node sets $S_1, S_2 \subseteq V$, a user-specified parameter θ ($0 < \theta \leq 1$), and a query node $q \in V$, a bichromatic UG-RkNN query returns the objects $s_1 \in S_1$ that have q as one of their k-nearest neighbors from S_2 with the probability being no smaller than θ. Formally,

$$\text{UG-bR}k\text{NN}(\mathcal{G}, q) = \{s_1 \in S_1 \mid \sum_{s_1 \in \text{bR}k\text{NN}(G \in \Phi(\mathcal{G}), q)} \Pr(G) \geq \theta\}$$

It is worth noting that the presented SDP and TripS algorithms can be easily extended to support *bichromatic* UG-RkNN search, by implementing following two changes. First, let $k\text{NN}(G, n, S_2)$ denote node n's k-nearest neighbors in S_2 on a graph G, and $d_G^k(n, S_2)$ represent the distance from node n to its kth nearest neighbor in S_2 on G. For bichormatic UG-RkNN retrieval, we only need to replace $k\text{NN}(G, n, S)$ and the distance $d_G^k(n, S)$ used in SDP and TripS algorithms with $k\text{NN}(G, n, S_2)$ and $d_G^k(n, S_2)$, respectively. Second, to answer *bichromatic* UG-RkNN search, the extended SDP will focus on set S_1 only to find the objects with the probability of being answer objects no smaller than θ. The extended SDP and TripS algorithms can support bichromatic UG-RkNN query efficiently, as to be demonstrated in Sect. 7.4.

7 Experimental evaluation

In this section, we present a comprehensive experimental evaluation. The objectives are threefold. First, we would like to verify the efficiency of the exact SDP algorithm for UG-RkNN query, and three pruning heuristics, i.e., GSP, EGR, and PBP. Second, we want to demonstrate the high performance of TripS algorithm. Finally, we would like to evaluate the performance of our proposed algorithms for bichromatic UG-RkNN search. In what follows, we first present our experimental settings, and then, we report the experimental results and our findings. All algorithms were implemented in C++, and all experiments were conducted on an Intel Core 2 Duo 2.10GHz Virtual Machine with 16GB RAM, running Microsoft Windows 7 Professional Edition.

Table 8 Statistics of the datasets used

Dataset	# of nodes	# of edges	References
NA	175,813	179,179	[29]
DBLP	1,226,749	5,557,805	[16,24,25,31,36,52,54]
Facebook	4,974	97,754	[24]
STRING	45,340	10,000,000	[23,32,53,55–57,59]
ER	12,500	50,000	[16,24,25,36]

7.1 Experimental setup

Datasets: four real datasets, viz., *North America* (*NA*), *DBLP*, *Facebook*, and *STRING*, are used in our experiments, representing real uncertain traffic networks, co-authorship networks, social networks, and PPI networks, respectively, as summarized in Table 8. Note that in order to show the popularity of the datasets, Table 8 also cites the specified work that used those datasets.

- *NA* contains 175, 813 nodes and 179, 179 edges, representing the locations and the road segments between two adjacent locations of North America, respectively.
- *DBLP* is a weighted collaboration network with 1,226,749 nodes and 5,557,805 edges. Each node denotes an author. The edge weight is set to the Jaccard distance of the published papers of the two nodes, which is based on the intuition that the more the papers they co-authored, the closer the relationship between the two authors is.
- *Facebook* includes a list of user-to-user links from the Facebook New Orleans network, in total 4, 974 users and 97,754 links. Every node signifies one anonymized user identifier.
- *STRING* is a prediction database derived from MINT, HPRD, BIOGRID, and INTACT, where each node represents a protein, and each edge stands for an interaction between two proteins with a confidence score that is used as the probability value in our study. In our experiments, we extract 45, 340 proteins with 10,000,000 interactions.

We also generate a synthetic Erdos–Renyi (*ER*) random graph, with 12, 500 vertices and 50, 000 edges. Following the popular methodology used in previous work [16,29,54,59], we simulate three possible weights for every edge with the existence probabilities under several typical distributions. Specifically, the weight distribution dtr_w can follow the Uniform (U_w for short) distribution or the Gaussian (G_w for short) distribution with mean as original weight ω (in the original graphs) and variance σ^2 as 0.1. For the uniform distribution with mean ω and variance σ^2, it means the weights are in the range $(\omega - \sqrt{3}\sigma, \omega + \sqrt{3}\sigma)$ uniformly, and thus, it is often abbreviated as *Uniform*$(\omega - \sqrt{3}\sigma, \omega + \sqrt{3}\sigma)$. For the Gaussian distribution with the mean as ω and the variance σ^2 as 0.1, it is abbreviated as *Gaussian*(ω, 0.1). In addition, we use a simple normalized approach to generate the weight probabilities. It first gets the probabilities following the probability distribution dtr$_p$ including the Uniform (U_p for short) distribution and the Gaussian (G_p for short) distribution with mean as 0.5 and variance as 0.25. Then, it normalizes the weight probabilities of one edge to sum up to one. Note that another simple method to generate probabilities is to allocate the probabilities for them uniformly. Also, more reasonable and complex techniques can be employed to set the probabilities. It is worth noting that as confirmed by the experimental study later, the probability setting is not very sensitive to the performance of our proposed heuristics and algorithms.

Algorithms: there are in total six algorithms evaluated in our experiments, as described below.

– Baseline (i.e., Algorithm 1).
– SDP (i.e., Algorithm 4). It employs three pruning heuristics including GSP, EGR, and PBP to get the exact UG-RkNN query result.
– TripS (i.e., Algorithm 5). It utilizes GSP heuristic to prune the graph scale and then uses ASSP sampling to derive the query result.
– GMC (GSP + Monte Carlo sampling). It first prunes the graph scale by our proposed GSP heuristic, and then, it derives the query result using Monte Carlo sampling based on [10].
– GR-I (GSP+ RSS-I). It employs GSP first and then uses RSS-I technique [24,25] to get the query result.
– GR-II (GSP+ RSS-II). It gets the query result via utilizing GSP and RSS-II techniques [24,25].

It is worth noting that RSS-I and RSS-II are the two special cases discussed in Sect. 6, where RSS-I partitions the whole space into w^γ subspaces, and RSS-II partitions the whole space into $\gamma \cdot (w-1)+1$ subspaces with w being the number of different weights for an edge. In addition, we set by default k = 2, θ = 0.5, d = 0.8, γ_1 = 20, and γ_2 = 30 for ASSP, γ = 50 for RSS-I and RSS-II, and sample threshold N_t =10 in every set of experiments. Unless mentioned otherwise, we report the average query execution time of 20 queries issued at random nodes.

Parameters: we explore several factors in our experiments, including weight/probability distribution dtr, graph size (i.e., the number of edges $|E|$), probability threshold θ, data density d (i.e., $|S|/|V|$, the rate of the nodes contain data objects to the total nodes in the graph), average node degree ρ (i.e., $|E|/|V|$, the average rate of the number of edges to the number of nodes), query parameter k, and sample size N. The settings of all these parameters are summarized in Table 9, where the default values are shown in bold, and $|D_E|$ denotes the original number of edges for each graph dataset. In every set of experiments, we only change one parameter, with the rest set to their defaults.

Table 9 Parameter ranges and default values

Parameter	Range
Weight distribution dtr$_w$	**Uniform**, Gaussian
Probability distribution dtr$_p$	Uniform, **Gaussian**
Graph size $\|E\|$	5, 10, 20, 100, 1000, $\|\boldsymbol{D_E}\|$
Query threshold θ	0.4, **0.5**, 0.6, 0.7
Data density d	0.7, **0.8**, 0.9, 1.0
Average node degree ρ	1, 2, 3, **4**, 5
k (of UG-RkNN query)	1, **2**, 4, 8
Sample size N	500, **1000**, 1500, 2000

Metrics: we employ the *average query time* and the *pruning rate* as the main performance metrics. Specifically, we employ the pruning rates R_{GSP}, R_{EGR}, and R_{PBP} w.r.t. Heuristic 1 GSP, Heuristic 2 EGR, and Heuristic 3 PBP, respectively. Note that the pruning rates of GSP, EGR, and PBP are calculated individually, and they prune the graph scale, the number of possible graphs, and the number of data nodes, respectively. The pruning rates are defined as follows.

– The pruning rate R_{GSP} is the ratio of the number of nodes pruned by GSP to the total number of nodes $|V|$ in the original graph, i.e., $R_{\mathsf{GSP}} = \frac{|V-N_g|}{|V|}$.
– The pruning rate R_{EGR} equals the ratio of the total number of equivalent graphs (defined in Definition 6) to the number of possible graphs w.r.t. the essential uncertain graph $\mathcal{G}_e$, i.e., $R_{\mathsf{EGR}} = \frac{|\bigcup \mathrm{D}^c|}{|\Phi(\mathcal{G}_e)|}$.
– The pruning rate R_{PBP} is the ratio of the number of data nodes identified (discarded/qualified) by PBP (denoted as Pruned(s)) to the number of data nodes S, i.e., $R_{\mathsf{PBP}} = \frac{|\mathrm{Pruned}(s)|}{|S|}$.

In addition to the pruning rate, for these three heuristics, we also report the average execution time (in seconds) of GSP (denoted by GSP-time), the times of EGR executed successfully (denoted by #EGR), i.e., the number of possible graphs pruned by EGR $|\bigcup \mathrm{D}^c|$, as well as the times of PBP executed successfully (denoted by #PBP), i.e., the number of data nodes pruned by PBP $|\mathrm{Pruned}(s)|$.

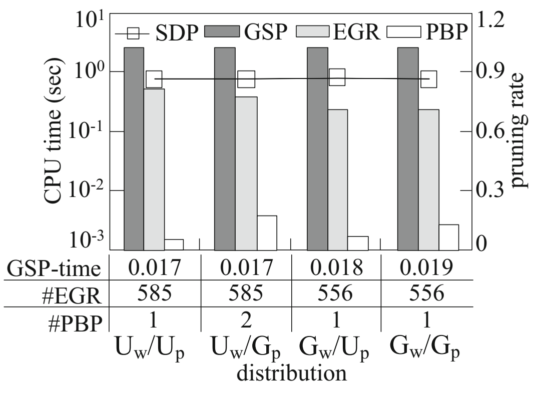
Furthermore, we introduce three different metrics to measure the accuracy of sampling algorithms, including the F_1-*score*, the *root mean square error* (RMSE), and the *relative variance*. Specifically, the F_1-score is calculated by $\frac{2PR}{P+R}$ where precision $P = \frac{TP}{TP+FP}$ and recall $R = \frac{TP}{TP+FN}$. Here, *TP*, *FP*, and *FN* denote the number of *true positives*, *false positives*, and *false negatives* for a sampling algorithm, respectively.

The RMSE metric is leveraged to measure the difference between the result derived by sampling algorithms and the real result (derived by exact algorithm SDP). The smaller the RMSE value, the smaller the difference between the result of sampling algorithm and the real result. For every query node, let $\mathbf{z_0}$ denote the exact probability vector of the $|S|$ data nodes to be UG-RkNN objects, and $\mathbf{z_i}$ be the probability vector derived by a sampling algorithm at the ith time ($i = 1, 2, \ldots, 100$). The RMSE of the result derived by sampling algorithms w.r.t. the real result, i.e., the difference between $\mathbf{z_0}$ and all $\mathbf{z_i}$s, is defined as $\sqrt{\frac{\sum_{i=1}^{100}\sum_{j=1}^{|S|}(\mathbf{z_i}[j]-\mathbf{z_0}[j])^2}{100}}$, in which $\mathbf{z_i}[j]$ is the jth dimensional value of $\mathbf{z_i}$, i.e., the probability of the jth data node in S to be a RkNN object.

In addition, as mentioned in Sect. 6, the MSE metric becomes the variance for unbiased algorithms including Monte Carlo sampling and TripS. Hence, following the previous work on uncertain graphs [16,24,25], we employ the *relative variance* to further evaluate the performance of sampling algorithms, since the exact variances of the algorithms are intractable due to the ♯P-hard complexity of UG-RkNN search. The relative variance of a sampling algorithm (denoted as $\hat{F}$) is defined as $\sigma_{\hat{F}}/\sigma_{MC}$, where $\sigma_{\hat{F}}$ and σ_{MC} are the variances of $\hat{F}$ and GMC algorithms, respectively. The smaller the relative variance, the better the performance of the sampling algorithm, as compared with Monte Carlo sampling. Let $\mathbf{z_a}$ be the average probability vector of all $\mathbf{z_i}$s, C_i be the cosine distance between $\mathbf{z_i}$ and $\mathbf{z_a}$, and $\bar{C}$ be the mean of all C_is. The variance σ is defined as $\frac{\sum_{i=1}^{100}(C_i-\bar{C})^2}{99}$.

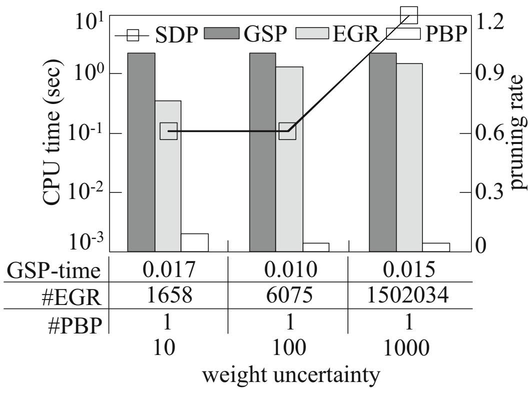
7.2 Results on exact algorithms

In the first set of experiments, we evaluate the performance of SDP under different edge weight distributions dtr$_w$ and weight probability distributions dtr$_p$. The experimental results on *NA* graph are shown in Fig. 6, where four distribution combinations (i.e., U_w/U_p, U_w/G_p, G_w/U_p, and G_w/G_p) are evaluated. It is observed that the execution time of SDP algorithm, as well as the execution time of GSP heuristic, is not very sensitive to weight/probability distributions. Three heuristics GSP, EGR, and PBP are effective in most cases. This is because heuristic GSP performs similarly under different weight distributions, and it is not affected by the weight probability distributions. Thus, without loss of generality, in the following experiments, the edge weights follow $Uniform$ (U_w) distribution, and the probabilities follow $Gaussian$ (G_p) distribution by default (i.e., dtr$_w = U_w$ and dtr$_p = G_p$).



	Uw/Up	Uw/Gp	Gw/Up	Gw/Gp
GSP-time	0.017	0.017	0.018	0.019
#EGR	585	585	556	556
#PBP	1	2	1	1

Fig. 6 UG-RkNN cost versus distribution on *NA*



	10	100	1000
GSP-time	0.017	0.010	0.015
#EGR	1658	6075	1502034
#PBP	1	1	1

Fig. 7 UG-RkNN cost versus uncertainty of weight on *NA*

Moreover, in order to further verify the effect of high weight uncertainty on the algorithm performance, we generate the weights following *Uniform*(1, 10), *Uniform*(1, 100), and *Uniform*(1, 1000) over *NA* graph, where the probability follows *Gaussian*(0.5, 0.25) by default. As depicted in Fig. 7, one can see that with the increasing of the weight uncertainty, the execute time of SDP algorithm grows remarkably. This is because the power of GSP becomes a little weak for high uncertainty, and therefore, the possible graphs we have to evaluate grow exponentially, which resulting in much overhead for query processing. Notably, the large increasing number of possible graphs we have to evaluate (with the ascend of weight uncertainty) can also be confirmed by the remarkable growth on the number of possible graphs pruned by EGR when the pruning rate of EGR does not vary significantly.

The second set of experiments compares the performance of Baseline and SDP algorithms for UG-RkNN search with various graph sizes, as depicted in Fig. 8. To vary graph size, we extract connected subgraphs from the original graphs as the experimental graphs. The extraction follows the same method used in [29], which expands the graph from a random selected vertex via breadth-first traversal. Obviously, SDP outperforms Baseline significantly, when the size of edges is larger than 10, and the average execution time of Baseline is around 10^5 s when edge size is 20. The reason is that the average query cost of Baseline grows exponentially with the increasing of edge size, which is consistent with the com-

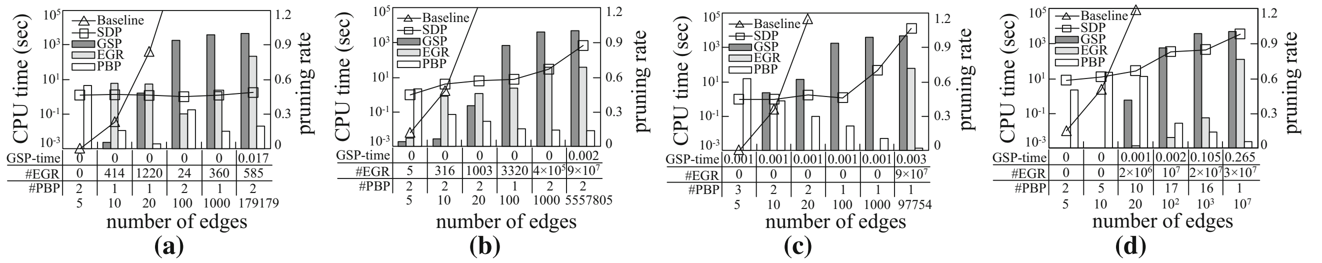


Fig. 8 UG-RkNN cost versus *graph size*. **a** *NA*. **b** *DBLP*. **c** *Facebook*. **d** *STRING*

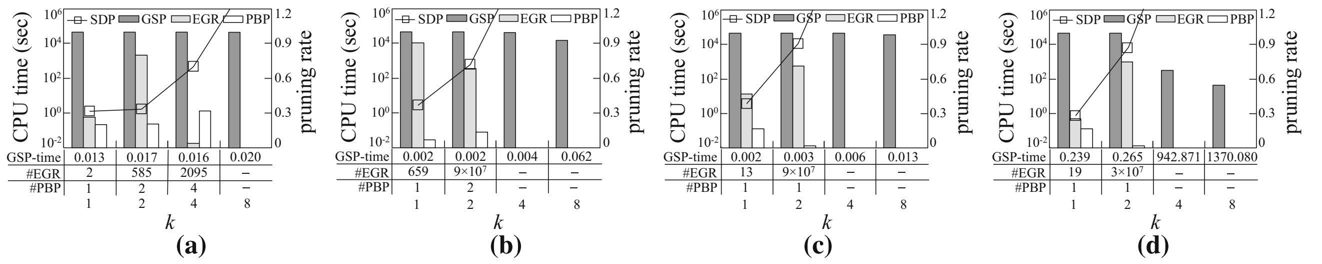


Fig. 9 UG-RkNN cost versus k. **a** *NA*. **b** *DBLP*. **c** *Facebook*. **d** *STRING*

plexity of Baseline, i.e., $O(w^{|E|} \cdot (|V|^2 \cdot \lg |V| + |V| \cdot |E|))$. Hence, the time cost of Baseline is unacceptable when edge size is larger than 20. In other words, Baseline is impractical on large graphs. In contrast, SDP is equipped with three effective heuristics GSP, EGR, and PBP, which boost the performance of UG-RkNN search.

Another observation is that GSP and EGR become more powerful when graphs turn larger in most cases. This is because when the graph is small, it limits the room for improvement. GSP takes much less time to prune more than 99% nodes of the original big graphs. It signifies at most 1% nodes are remained in the essential uncertain graphs. In addition, the power of PBP drops with the growth of graph size in most cases. The reason is that as GSP and EGR do not work efficiently on smaller graph, it gives PBP the chance to prune more data nodes. However, with the increasing power of GSP and EGR, fewer data nodes remain for PBP in UG-RkNN retrieval.

The third set of experiments reports the experimental results with k varying from 1 to 8, as depicted in Fig. 9. It is worthwhile to point out that due to the high complexity and poor performance of Baseline, Baseline is impractical for these graphs. Hence, we focus on SDP algorithm only in this and the rest sets of the experiments. Evidently, the average query time of SDP ascends significantly with the growth of k. The reason is that as k grows, the number of nodes in the essential uncertain graph formed by GSP becomes larger; therefore, the number of possible graphs we have to evaluate grows exponentially. It is also consistent with our analysis presented in Lemma 7 of Sect. 3, where the pruning rate R_{GSP} drops with the growth of k. When k turns large, e.g., $k = 4$ and 8, the average query time of exact SDP algorithm is relatively long (even more than 10^6 s). This time cost is beyond the scope of acceptance. Notably, the first step of SDP algorithm, i.e., GSP heuristic, has been conducted completely. It is important to note that in Fig. 9, we only plot part of the results (i.e., the CPU time and the pruning rates) when the average query time is acceptable. The corresponding pruning rates of EGR and PBP are denoted by the dash "–," if the time cost is unacceptable.

The fourth set of experiments verifies the influence of data density d on the efficiency of SDP. The average query time

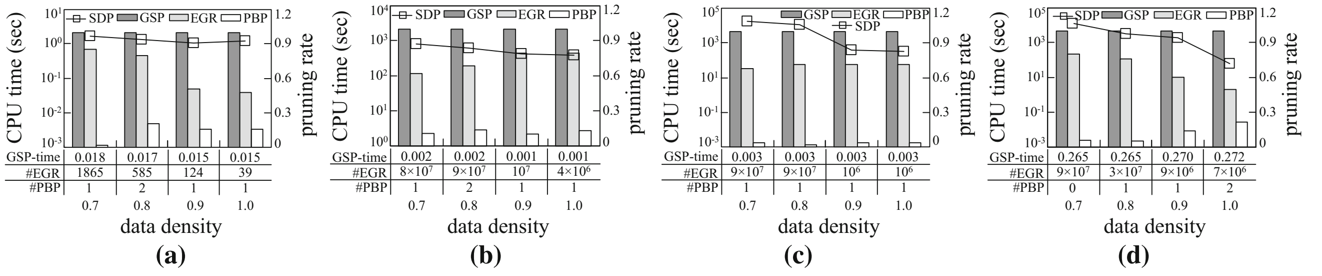


Fig. 10 UG-RkNN cost versus d. **a** *NA*. **b** *DBLP*. **c** *Facebook*. **d** *STRING*

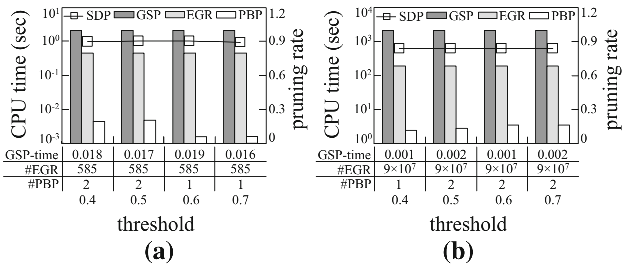


Fig. 11 UG-RkNN cost versus θ. **a** *NA*. **b** *DBLP*

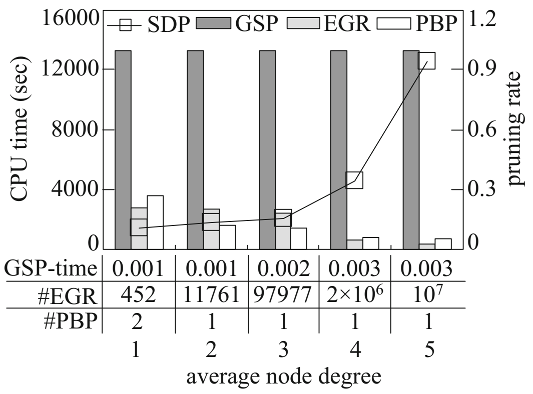


Fig. 12 UG-RkNN cost versus ρ on *ER*

of SDP and the pruning performance of three heuristics on different datasets are depicted in Fig. 10, with varying d from 0.7 to 1.0. Note that data density d is the ratio of the number of nodes containing (relevant) data objects to the total number of nodes, i.e., $|S|/|V|$. Observed that the average query time drops as d grows. The reason is that there are more data objects on the graphs with the growth of d. Therefore, for a given k, the distance from every node n to its kth nearest neighbor becomes smaller, resulting in an essential uncertain graph with smaller size. Thus, query performance improves. It is obvious that all the three heuristics are consistently effective in all cases.

The fifth set of experiments investigates the impact of parameter θ on UG-RkNN search, with the results plotted in Fig. 11. It is observed that the average query time of the algorithms is not very sensitive to θ. This is because θ only affects the performance of PBP heuristic, which is employed in the final refinement step of UG-RkNN query. Moreover, all the heuristics demonstrate significant pruning power in all cases. Note that due to the similar performance under various θs, we only report the corresponding results on *NA* and *DBLP* for clarity and simplicity.

We also study the effect of graph structure on the performance of SDP. We show the average query time of SDP and the pruning efficiency of heuristics on ER dataset in Fig. 12 by changing the average node degree ρ. Note that average node degree ρ denotes the average ratio of the number of edges to the number of nodes, i.e., $|E|/|V|$. Specifically, we set ρ to 1, 2, 3, 4, and 5, respectively, by selecting/adding the corresponding proportion of the edges from every original graph (with a given node set V). Clearly, the average query time becomes larger with a high node degree ρ. This is because for a specified number of nodes, the size of edges grows with ρ, incurring a large number of possible graphs during search. The average query time increases accordingly. This phenomenon is consistent with our analysis presented in Lemma 7 of Sect. 3, where the pruning rate R_{GSP} decreases with the growth of ρ. Also, we again observe that all three heuristics perform efficiently in all cases.

In total, SDP algorithm has a good performance, compared with Baseline. It benefits from the three effective heuristics including GSP, EGR, and PBP.

7.3 Results on sampling algorithms

In this subsection, we verify the efficiency of TripS algorithm on UG-RkNN search. First, we present the experimental results of SDP and TripS algorithms with sample size being 1000 under various k values in Table 10. We can observed that TripS algorithm performs better than SDP in several orders of magnitude for all datasets. This is because TripS answers UG-RkNN query approximately via sampling 1000 possible graphs (instead of evaluating all the possible graphs). The processing time increases with the growth of k. The reason behind is as same as the explanation of Fig. 9. Note that in Table 10, some results of SDP are denoted by the dash "–," meaning that the corresponding query time of exact SDP algorithm is beyond the scope of acceptance.

Second, we evaluate the accuracy of the sampling algorithms (including GMC, GR-I, GR-II, and TripS) by three

Table 10 Average query time (in seconds) of SDP and TripS algorithms under different k values

Dataset	$k = 1$		$k = 2$		$k = 4$		$k = 8$	
	SDP	TripS	SDP	TripS	SDP	TripS	SDP	TripS
NA	1.306	0.001	1.744	0.022	499.262	0.063	–	0.115
DBLP	2.940	0.001	636.156	0.043	–	6.375	–	32.729
Facebook	3.691	0.001	11,635.800	0.049	–	1.177	–	7.100
STRING	110.257	0.021	3,290.841	0.106	–	0.743	–	3.526
ER	3.278	0.018	4,585.634	0.046	–	0.116	–	14.729

Table 11 RMSE of the result from sampling algorithms w.r.t. the exact result

Dataset	GMC	GR-I	GR-II	TripS
NA	0.573	0.575	0.572	0.569
DBLP	0.749	0.749	0.749	0.733
Facebook	0.512	0.512	0.511	0.500
STRING	0.723	0.722	0.721	0.708
ER	0.511	0.511	0.510	0.500

metrics, i.e., the F_1-*score*, the *root mean square error* (RMSE), and the *relative variance*. From the experimental results, we observe that all the four algorithms have identical F_1-score on every dataset, e.g., the F_1-score is 0.696 on *DBLP* dataset with $\theta = 0$. This is because these algorithms find the same answer objects (with different probabilities to be RkNN objects) in most cases, while the probability information is not measured by F_1-score metric. In fact, what distinguishes these algorithms is usually the probabilities to be RkNN objects. This exactly reveals the shortcoming of the F_1-score metric for sampling algorithms. Due to this observation, we skip the experimental results w.r.t. F_1-score metric.

Table 11 lists the RMSE between the result of sampling algorithms and the real result. Notice that TripS consistently generates a smaller RMSE than its competitors, because of the adaptive recursive stratified sampling used in TripS. In other words, TripS returns the result that is the *closest* to the real result.

Figure 13 shows the relative variances and the average query time of sampling algorithms with varying sample size N. Note that we use the line (bar) to represent the relative variance (query time). Clearly, under the comparable average query time, TripS consistently achieves a higher accuracy (indicated by a lower relative variance) than GMC, GR-I, and GR-II. This is because TripS shares the same time complexity with the competitors, and thus, their runtime is similar, but TripS adopts an adaptive recursive stratified sampling technique to achieve a lower relative variance. It also signifies that our ASSP technique employed in TripS is superior to its competitors including Monte Carlo sampling, RSS-I sampling, and RSS-II sampling [24,25]. In addition, the relative variance drops (i.e., the accuracy improves) and the query time increases with the increasing sample size. The reason is that more RkNN queries need to be conducted when there are more sampled possible graphs, resulting in more expensive query cost and higher accuracy.

To sum up, TripS performs the best for UG-RkNN search. In addition, we would like to highlight that both the F_1-score metric and the RMSE metric are *inapplicable* if the exact algorithm for UG-RkNN search is impractical. Thus, due to the ♯P-hard complexity of UG-RkNN query, the relative variance is a relatively appropriate metric for sampling algorithms.

7.4 Results on bichromatic UG-RkNN search

In order to verify the efficiency of the algorithms for processing bichromatic UG-RkNN queries, we first report the average query time of SDP algorithm and the pruning rates of GSP, EGR, and PBP heuristics under default settings, as listed in Table 12. As expected, similar to the performance reported in the above (monochromatic) UG-RkNN retrieval, three pruning heuristics are always effective. Furthermore, GSP plays an important role in SDP algorithm for bichromatic UG-RkNN search.

In addition, we also report the accuracy and the average query time of GMC, GR-I, GR-II, and TripS algorithms in Tables 13 and 14, respectively. Compared with the performance of SDP shown in Table 12, it is obvious that TripS is much faster than SDP algorithm in all cases. From Tables 13 and 14, we can conclude that TripS can support bichromatic UG-RkNN queries with high accuracy and comparable time, compared with state-of-the-art algorithms including GMC, GR-I, and GR-II. Note that in this set of experiments, the data node set S in default (i.e., $|S| = 0.8 \times |V|$) is randomly partitioned into two equal-sized node sets S_1 and S_2 for bichromatic UG-RkNN search. Due to the same complexity of the algorithm as that of (monochromatic) UG-RkNN

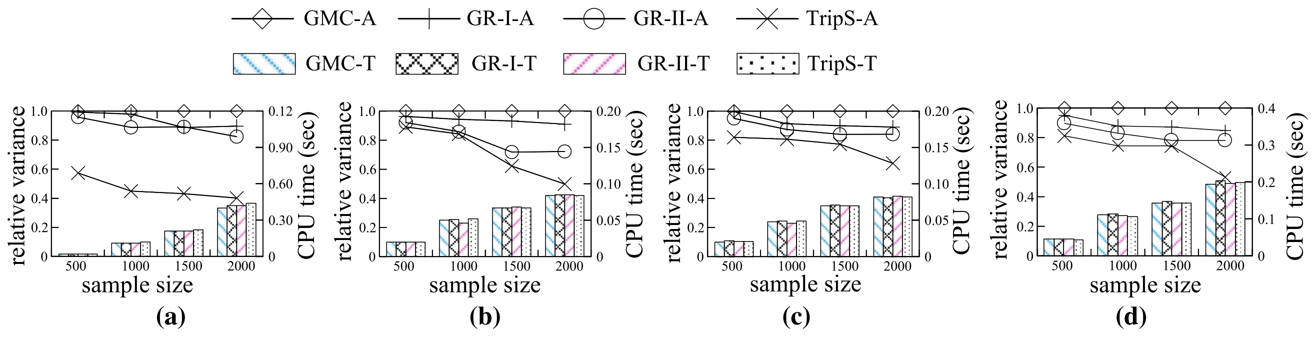


Fig. 13 UG-RkNN search versus N. **a** *NA*. **b** *DBLP*. **c** *Facebook*. **d** *STRING*

Table 12 Results on exact bichromatic UG-RkNN search

Dataset	SDP(sec)	GSP (%)	EGR(%)	PBP(%)
NA	3,571	99.99	12.06	21.83
DBLP	6,271	99.99	1.66	4.68
Facebook	167,014	99.97	11.94	6.45
STRING	5,450	99.99	60.49	8.33
ER	11,782	99.96	26.44	12.72

Table 13 Comparison of the accuracy (relative variance)

Dataset	GMC	GR-I	GR-II	TripS
NA	1	0.917	0.892	0.815
DBLP	1	0.805	0.802	0.740
Facebook	1	0.957	0.947	0.832
STRING	1	0.804	0.766	0.486
ER	1	0.257	0.242	0.224

Table 14 Comparison of average query time (in seconds)

Dataset	GMC	GR-I	GR-II	TripS
NA	3.230	3.119	3.159	3.129
DBLP	48.041	47.516	47.602	47.857
Facebook	20.608	20.485	20.518	20.614
STRING	32.162	32.503	32.389	32.268
ER	29.880	29.376	29.195	29.079

query, we omit the experimental results for bichromatic UG-RkNN search under different parameters.

8 Related work

In this section, we review the existing work on RkNN queries and the management of uncertain graphs.

8.1 Reverse k-nearest neighbor queries

RkNN search has received considerable attentions in the last decade, since the concept of reverse nearest neighbor (RNN) was first introduced in [18]. Many algorithms have been proposed in answering RNN/RkNN query and its variants in the literature [9,11,12,20,37,41,43,47–51]. Probabilistic RkNN retrieval on uncertain data has also been extensively studied. Lian and Chen [26] study the PRNN query, which retrieves the data objects with their probabilities being RNNs greater than or equal to a user-specified threshold, and propose an effective pruning method, i.e., geometric pruning (GP). Cheema et al. [6] explore the problem of RNN queries on uncertain data, and develop novel pruning rules that can effectively prune the objects that cannot be the answer objects. Bernecker et al. [5] developed a general framework for probabilistic reverse nearest neighbor queries on uncertain data. Xu et al. [49] explore interval reverse nearest neighbor (IRNN) queries over moving objects, which return the objects that maintain nearest neighboring relations to the moving query objects for the longest time in a given interval. IRNN actually considers location uncertainty of moving objects.

In addition, Yiu et al. [52] investigate the RkNN query on graphs and present a fundamental lemma that can be employed to prune the search space. Safar et al. [39] show how Voronoi diagram can be applied to spatial query processing, and in particular to RNN search. Li et al. [22] study continuous RkNN search on road networks. They utilize a new data structure, called *DLM tree*, to represent the whole monitoring region of a continuous RkNN query. Cheema et al. [7] adopt a filter-and-refinement technique to solve continuous RkNN search on Euclidean spaces and road networks, respectively. Wang et al. [46] utilize the concept of *influence zone* to address the problem of continuous monitoring of RkNN queries in road networks. The influence zone is a region in the network such that a client c is the RkNN if and only if it lies inside this zone.

Note that all the aforementioned methods are unsuitable for UG-RkNN search because they either only focus on the deterministic graph (network) or only consider the uncertainty of the data but not uncertain structure of the graph.

8.2 Uncertain graphs

Recently, how to manage uncertain graphs has become an increasingly important research topic. There are four main research lines on uncertain graphs: (i) the queries based on shortest path distances over uncertain graphs, such as k-nearest neighbor search on uncertain graphs [36], aggregate nearest neighbor retrieval on uncertain graphs [29], and distance-constraint reachability computation on uncertain graphs [16]; (ii) pattern mining on uncertain graphs, e.g., frequent subgraph pattern mining over uncertain graphs [8,23,31,59], finding (top-k) maximal cliques on uncertain graphs [32,58], and uncertain graph data mining [17]; (iii) subgraph/ supergraph (similarity) search over uncertain graphs, such as finding reliable subgraph from probabilistic graphs [15] and subgraph/ supergraph (similarity) search on uncertain graphs [53,55–57]; and (iv) keyword search over uncertain graphs [27,54]. In addition, there are some studies on the representation of uncertain graphs [1,34] and sampling over uncertain graphs [10,24,25].

We want to highlight that our work is different from the above efforts on uncertain graphs, because we focus on RkNN retrieval over uncertain graphs. This is, to the best of our knowledge, the first attempt on UG-RkNN search.

9 Conclusions

In this paper, we study the problem of *reverse k-nearest neighbor queries on uncertain graphs* (UG-RkNN query for short). We first introduce GSP heuristic to replace the original uncertain graph with a much smaller *essential uncertain graph*, which reduces the size of uncertain graph significantly for UG-RkNN search. The effectiveness of GSP is guaranteed by a theoretical analysis, and is confirmed via experimental study. Then, we propose EGR heuristic to remove the *equivalent possible graphs* to further reduce the number of possible graphs. A novel concept of *graph conditional dominance relationship* is introduced to enable an efficient implementation for EGR. We also present PBP heuristic to minimize the validation cost of the data nodes. Combining these three newly presented heuristics, we propose SDP algorithm to support exact UG-RkNN search. We further present a novel sampling algorithm, i.e., TripS, based on an adaptive stratified ASSP sampling and GSP heuristic, for UG-RkNN search. ASSP is unbiased, and its variance is at least as good as that of Monte Carlo sampling. We also generalize our heuristics to tackle bichromatic UG-RkNN retrieval. Extensive experiments using both real and synthetic graphs demonstrate the efficiency of our proposed algorithms. In the future, we plan to explore UG-RkNN search under the graph model with uncertain nodes.

Acknowledgements This work was supported in part by the 973 Program of China Grant Nos. 2013CB336500 and 2015C B352502, NSFC Grant Nos. 61522208, 61379033, and 61472 348, and the NSFC-Zhejiang Joint Fund Grant No. U1609217. We also would like to express our gratitude to some anonymous reviewers for their giving valuable and helpful comments to improve the technical quality and presentation of this paper.

Appendix

A Proof of Theorem 1

Proof Let t_i be the rate of the number of samples (N_i) to the number of population (L_i) for stratum i, as defined in Eq. 10. Based on the theorem of mathematical analysis [38], for stratum i, the variance of the sample S_i^2 and the variance of the simple random sampling $\mathrm{Var}(\hat{F}_i)$ are given in Eqs. 11 and 12, respectively.

$$t_i = \frac{N_i}{L_i} \tag{10}$$

$$S_i^2 = \frac{\sum_{j=1}^{L_i}(F_{ij} - \hat{F}_i)^2}{L_i - 1} \tag{11}$$

$$\mathrm{Var}(\hat{F}_i) = \frac{1 - t_i}{N_i} \cdot S_i^2 \tag{12}$$

Based on these three equation, let T represent the total number of strara in stratified sampling, and $\hat{F}_i$ denote the estimator of the true value F in the stratum i, where $\pi_i = L_i/L$. Then, we have

$$\begin{aligned}
\mathbb{E}(\hat{F}_{\mathrm{ASSP}}) &= \mathbb{E}\left(\sum_{i=1}^{T} \pi_i \hat{F}_i\right) \\
&= \sum_{i=1}^{T} \pi_i E(\hat{F}_i) \\
&= \sum_{i=1}^{T} \pi_i F_i \quad // \ F_i \text{ is unbiasd} \\
&= \sum_{i=1}^{T} \pi_i \frac{1}{L_i} \cdot \sum_{j=1}^{L_i} F_{ij} \\
&= \sum_{i=1}^{T} \frac{L_i}{L} \frac{1}{L_i} \cdot \sum_{j=1}^{L_i} F_{ij} \\
&= \frac{1}{L} \sum_{i=1}^{T} \sum_{j=1}^{L_i} F_{ij} \\
&= F.
\end{aligned}$$

□

B Proof of Lemma 10

Proof

$$\begin{aligned}
&\mathrm{Var}(\hat{F}_i) \\
&= \frac{1 - t_i}{N_i} \cdot S_i^2 \quad // \text{ based on Eq. 12} \\
&= \frac{L_i - N_i}{L_i - 1} \cdot \frac{1}{N_i} \cdot \frac{1}{L_i} \cdot \sum_{j=1}^{L_i} (F_{ij} - \hat{F}_i)^2 \\
&= \frac{L_i - N_i}{L_i - 1} \cdot \frac{1}{N_i} \cdot \frac{1}{L_i} \cdot \sum_{j=1}^{L_i} (F_{ij}^2 - 2F_{ij}\hat{F}_i + \hat{F}_i^{\,2}) \\
&= \frac{L_i - N_i}{L_i - 1} \cdot \frac{1}{N_i} \cdot \frac{1}{L_i} \\
&\quad \cdot \left(\sum_{j=1}^{L_i} F_{ij}^2 - 2\hat{F}_i \sum_{j=1}^{L_i} F_{ij} + L_i \hat{F}_i^2\right) \\
&= \frac{L_i - N_i}{L_i - 1} \cdot \frac{1}{N_i} \cdot \frac{1}{L_i} \cdot \left(\sum_{j=1}^{L_i} F_{ij}^2 - L_i \hat{F}_i^2\right) \\
&= \frac{L_i - N_i}{L_i - 1} \cdot \frac{1}{N_i} \cdot \left(\frac{1}{L_i} \sum_{j=1}^{L_i} F_{ij} - \hat{F}_i^2\right)
\end{aligned}$$

$$= \frac{L_i - N_i}{L_i - 1} \cdot \frac{\hat{F}_i - \hat{F}_i^2}{N_i}$$

Combining the equation that $\mathrm{Var}(\hat{F}) = \sum_{i=1}^{T} \pi_i^2 \mathrm{Var}(\hat{F}_i)$ [38], Thus, we have

$$\mathrm{Var}(\hat{F}_{\mathrm{ASSP}}) = \sum_{i=1}^{T} \pi_i^2 \frac{L_i - N_i}{L_i - 1} \cdot \frac{\hat{F}_{\mathrm{ASSP}}(1 - \hat{F}_{\mathrm{ASSP}})}{N_i}.$$

□

C Proof of Theorem 2

Proof On the one hand,

$$\begin{aligned}
\mathrm{Var}(\hat{F}_{\mathrm{ASSP}}) &= \sum_{i=1}^{T} \pi_i^2 \mathrm{Var}(\hat{F}_i) \\
&= \sum_{i=1}^{T} \pi_i^2 \left(\frac{1}{N_i} - \frac{1}{L_i}\right) S_i^2 \\
&= \sum_{i=1}^{T} \frac{\pi_i^2 S_i^2}{N_i} - \sum_{i=1}^{T} \frac{\pi_i^2 S_i^2}{L_i} \\
&= \frac{\sum_{i=1}^{T} \pi_i S_i^2}{N} - \frac{\sum_{i=1}^{T} \pi_i S_i^2}{L} \\
&= \left(\frac{1}{N} - \frac{1}{L}\right) \sum_{i=1}^{T} \pi_i S_i^2
\end{aligned}$$

On the other hand, $\mathrm{Var}(\hat{F}_{\mathrm{MC}}) = (\frac{1}{N} - \frac{1}{L}) S^2$,

$$\begin{aligned}
(L-1)S^2 &= \sum_{n=1}^{L} (F_n - \hat{F})^2 \\
&= \sum_{i=1}^{T} \sum_{n=1}^{N_i} (F_{in} - \hat{F})^2 \\
&= \sum_{i=1}^{T} \sum_{n=1}^{N_i} (F_{in} - \hat{F}_i)^2 + \sum_{i=1}^{T} L_i (\hat{F}_i - \hat{F})^2 \\
&= \sum_{i=1}^{T} (L_i - 1) S_i^2 + \sum_{i=1}^{T} L_i (\hat{F}_i - \hat{F})^2 \\
S^2 &= \sum_{i=1}^{T} \frac{(L_i - 1)}{L - 1} S_i^2 + \sum_{i=1}^{T} \frac{L_i}{L - 1} (\hat{F}_i - \hat{F})^2 \\
&\doteq \sum_{i=1}^{T} \pi S_i^2 + \sum_{i=1}^{T} \pi (\hat{F}_i - \hat{F})^2 \\
&\quad // \text{ when } L \text{ is rather large}
\end{aligned}$$

Hence, we have

$$\begin{aligned}
\mathrm{Var}(\hat{F}_{\mathrm{MC}}) &= \left(\frac{1}{N} - \frac{1}{L}\right) S^2 \\
&= \left(\frac{1}{N} - \frac{1}{L}\right) \left[\sum_{i=1}^{T} \pi S_i^2 + \sum_{i=1}^{T} \pi (\hat{F}_i - \hat{F})^2\right]
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\mathrm{Var}(\hat{F}_{\mathrm{MC}}) - \mathrm{Var}(\hat{F}_{\mathrm{ASSP}}) \\
&= \left(\frac{1}{N} - \frac{1}{L}\right) \left[\sum_{i=1}^{T} \pi S_i^2 \right.\\
&\quad \left. + \sum_{i=1}^{T} \pi (\hat{F}_i - \hat{F})^2\right] \\
&\quad - \left(\frac{1}{N} - \frac{1}{L}\right) \sum_{i=1}^{T} \pi_i S_i^2 \\
&= \left(\frac{1}{N} - \frac{1}{L}\right) \sum_{i=1}^{T} \pi (\hat{F}_i - \hat{F})^2 \\
&\geq 0
\end{aligned}$$

□

References

1. Abiteboul, S., Kanellakis, P., Grahne, G.: On the representation and querying of sets of possible worlds. In: SIGMOD, pp. 34–48 (1987)
2. Achtert, E., Böhm, C., Kröger, P., Kunath, P., Pryakhin, A., Renz, M.: Efficient reverse k-nearest neighbor estimation. Informatik-Forschung und Entwicklung **21**(3–4), 179–195 (2007)
3. Adar, E., Ré, C.: Managing uncertainty in social networks. IEEE Data Eng. Bull. **30**(2), 15–22 (2007)
4. Asthana, S., King, O.D., Gibbons, F.D., Roth, F.P.: Predicting protein complex membership using probabilistic network reliability. Genome Res. **14**(6), 1170–1175 (2004)
5. Bernecker, T., Emrich, T., Kriegel, H.P., Renz, M., Zankl, S., Züfle, A.: Efficient probabilistic reverse nearest neighbor query processing on uncertain data. PVLDB **4**(10), 669–680 (2011)
6. Cheema, M.A., Lin, X., Wang, W., Zhang, W., Pei, J.: Probabilistic reverse nearest neighbor queries on uncertain data. IEEE Trans. Knowl. Data Eng. **22**(4), 550–564 (2010)
7. Cheema, M.A., Zhang, W., Lin, X., Zhang, Y., Li, X.: Continuous reverse k nearest neighbors queries in Euclidean space and in spatial networks. VLDB J. **21**(1), 69–95 (2012)
8. Chen, L., Wang, C.: Continuous subgraph pattern search over certain and uncertain graph streams. IEEE Trans. Knowl. Data Eng. **22**(8), 1093–1109 (2010)
9. Choudhury, F.M., Culpepper, J.S., Sellis, T., Cao, X.: Maximizing bichromatic reverse spatial and textual k nearest neighbor queries. PVLDB **9**(6), 456–467 (2016)
10. Emrich, T., Kriegel, H.P., Niedermayer, J., Renz, M., Suhartha, A., Züfle, A.: Exploration of Monte-Carlo based probabilistic query processing in uncertain graphs. In: CIKM, pp. 2728–2730 (2012)

11. Gao, Y., Liu, Q., Miao, X., Yang, J.: Reverse k-nearest neighbor search in the presence of obstacles. Inf. Sci. **330**, 274–292 (2016)
12. Gao, Y., Zheng, B., Chen, G., Lee, W.C., Lee, K.C., Li, Q.: Visible reverse k-nearest neighbor query processing in spatial databases. IEEE Trans. Knowl. Data Eng. **21**(9), 1314–1327 (2009)
13. Gu, Y., Gao, C., Cong, G., Yu, G.: Effective and efficient clustering methods for correlated probabilistic graphs. IEEE Trans. Knowl. Data Eng. **26**(5), 1117–1130 (2014)
14. Hung, H.J., Yang, D.N., Lee, W.C.: Social influence-aware reverse nearest neighbor search. In: DSAA, pp. 223–229. IEEE (2014)
15. Jin, R., Liu, L., Aggarwal, C.C.: Discovering highly reliable subgraphs in uncertain graphs. In: SIGKDD, pp. 992–1000 (2011)
16. Jin, R., Liu, L., Ding, B., Wang, H.: Distance-constraint reachability computation in uncertain graphs. PVLDB **4**(9), 551–562 (2011)
17. Kollios, G., Potamias, M., Terzi, E.: Clustering large probabilistic graphs. IEEE Trans. Knowl. Data Eng. **25**(2), 325–336 (2013)
18. Korn, F., Muthukrishnan, S.: Influence sets based on reverse nearest neighbor queries. In: SIGMOD, pp. 201–212 (2000)
19. Krogan, N.J., Cagney, G., Yu, H., Zhong, G., Guo, X., Ignatchenko, A., Li, J., Pu, S., Datta, N., Tikuisis, A.P., et al.: Global landscape of protein complexes in the yeast saccharomyces cerevisiae. Nature **440**(7084), 637–643 (2006)
20. Lee, K.C., Zheng, B., Lee, W.C.: Ranked reverse nearest neighbor search. IEEE Trans. Knowl. Data Eng. **20**(7), 894–910 (2008)
21. Levin, R., Kanza, Y.: Stratified-sampling over social networks using mapreduce. In: SIGMOD, pp. 863–874 (2014)
22. Li, G., Li, Y., Li, J., LihChyun, S., Yang, F.: Continuous reverse k nearest neighbor monitoring on moving objects in road networks. Inf. Syst. **35**(8), 860–883 (2010)
23. Li, J., Zou, Z., Gao, H.: Mining frequent subgraphs over uncertain graph databases under probabilistic semantics. VLDB J. **21**(6), 753–777 (2012)
24. Li, R.H., Yu, J.X., Mao, R., Jin, T.: Efficient and accurate query evaluation on uncertain graphs via recursive stratified sampling. In: ICDE, pp. 892–903 (2014)
25. Li, R.H., Yu, J.X., Mao, R., Jin, T.: Recursive stratified sampling: a new framework for query evaluation on uncertain graphs. IEEE Trans. Knowl. Data Eng. **28**(2), 468–482 (2016)
26. Lian, X., Chen, L.: Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data. VLDB J. **18**(3), 787–808 (2009)
27. Lian, X., Chen, L., Huang, Z.: Keyword search over probabilistic RDF graphs. IEEE Trans. Knowl. Data Eng. **27**(5), 1246–1260 (2015)
28. Liu, G., Wong, L., Chua, H.N.: Complex discovery from weighted PPI networks. Bioinformatics **25**(15), 1891–1897 (2009)
29. Liu, Z., Wang, C., Wang, J.: Aggregate nearest neighbor queries in uncertain graphs. World Wide Web **17**(1), 161–188 (2014)
30. Melaniphy, J.C.: The restaurant location guidebook: a comprehensive guide to selecting restaurant & quick service food locations. International Real Estate Location Institute (2007)
31. Moustafa, W.E., Kimmig, A., Deshpande, A., Getoor, L.: Subgraph pattern matching over uncertain graphs with identity linkage uncertainty. In: ICDE, pp. 904–915 (2014)
32. Mukherjee, A.P., Xu, P., Tirthapura, S.: Mining maximal cliques from an uncertain graph. In: ICDE, pp. 243–254 (2015)
33. Ning, K., Ng, H.K., Srihari, S., Leong, H.W., Nesvizhskii, A.I.: Examination of the relationship between essential genes in PPI network and hub proteins in reverse nearest neighbor topology. BMC Bioinform. **11**(1), 1 (2010)
34. Parchas, P., Gullo, F., Papadias, D., Bonchi, F.: The pursuit of a good possible world: extracting representative instances of uncertain graphs. In: SIGMOD, pp. 967–978 (2014)
35. Parchas, P., Gullo, F., Papadias, D., Bonchi, F.: Uncertain graph processing through representative instances. ACM Trans. Database Syst. **40**(3), 20 (2015)
36. Potamias, M., Bonchi, F., Gionis, A., Kollios, G.: K-nearest neighbors in uncertain graphs. PVLDB **3**(1), 997–1008 (2010)
37. Radovanovic, M., Nanopoulos, A., Ivanovic, M.: Reverse nearest neighbors in unsupervised distance-based outlier detection. IEEE Trans. Knowl. Data Eng. **27**(5), 1369–1382 (2015)
38. Rice, J.: Mathematical statistics and data analysis. Cengage Learning (2006)
39. Safar, M., Ibrahimi, D., Taniar, D.: Voronoi-based reverse nearest neighbor query processing on spatial networks. Multimedia Syst. **15**(5), 295–308 (2009)
40. Sen, P., Deshpande, A., Getoor, L.: PrDB: managing and exploiting rich correlations in probabilistic databases. VLDB J. **18**(5), 1065–1090 (2009)
41. Stanoi, I., Agrawal, D., El Abbadi, A.: Reverse nearest neighbor queries for dynamic databases. In: SIGMOD, pp. 44–53 (2000)
42. Suratanee, A., Plaimas, K.: Identification of inflammatory bowel disease-related proteins using a reverse k-nearest neighbor search. J. Bioinf. Comput. Biol. **12**(04), 1450017 (2014)
43. Tao, Y., Papadias, D., Lian, X.: Reverse kNN search in arbitrary dimensionality. In: VLDB, pp. 744–755 (2004)
44. Tao, Y., Yiu, M.L., Mamoulis, N.: Reverse nearest neighbor search in metric spaces. IEEE Trans. Knowl. Data Eng. **18**(9), 1239–1252 (2006)
45. Wackerly, D., Mendenhall, W., Scheaffer, R.: Mathematical statistics with applications. Nelson Education (2007)
46. Wang, S., Cheema, M.A., Lin, X.: Efficiently monitoring reverse k-nearest neighbors in spatial networks. Comput. J. **58**(1), 40–56 (2015)
47. Wang, S., Cheema, M.A., Lin, X., Zhang, Y., Liu, D.: Efficiently computing reverse k furthest neighbors. In: ICDE, pp. 1110–1121 (2016)
48. Wu, W., Yang, F., Chan, C.Y., Tan, K.L.: Finch: Evaluating reverse k-nearest-neighbor queries on location data. PVLDB **1**(1), 1056–1067 (2008)
49. Xu, C., Gu, Y., Chen, L., Qiao, J., Yu, G.: Interval reverse nearest neighbor queries on uncertain data with markov correlations. In: ICDE, pp. 170–181 (2013)
50. Yang, S., Cheema, M.A., Lin, X., Wang, W.: Reverse k nearest neighbors query processing: experiments and analysis. PVLDB **8**(5), 605–616 (2015)
51. Yang, S., Cheema, M.A., Lin, X., Zhang, Y.: Slice: Reviving regions-based pruning for reverse k nearest neighbors queries. In: ICDE, pp. 760–771 (2014)
52. Yiu, M.L., Papadias, D., Mamoulis, N., Tao, Y.: Reverse nearest neighbors in large graphs. IEEE Trans. Knowl. Data Eng. **18**(4), 540–553 (2006)
53. Yuan, Y., Wang, G., Chen, L., Wang, H.: Efficient subgraph similarity search on large probabilistic graph databases. PVLDB **5**(9), 800–811 (2012)
54. Yuan, Y., Wang, G., Chen, L., Wang, H.: Efficient keyword search on uncertain graph data. IEEE Trans. Knowl. Data Eng. **25**(12), 2767–2779 (2013)
55. Yuan, Y., Wang, G., Chen, L., Wang, H.: Graph similarity search on large uncertain graph databases. VLDB J. **24**(2), 271–296 (2015)
56. Yuan, Y., Wang, G., Wang, H., Chen, L.: Efficient subgraph search over large uncertain graphs. PVLDB **4**(11), 876–886 (2011)
57. Zhang, W., Lin, X., Zhang, Y., Zhu, K., Zhu, G.: Efficient probabilistic supergraph search. IEEE Trans. Knowl. Data Eng. **28**(4), 965–978 (2016)
58. Zou, Z., Li, J., Gao, H., Zhang, S.: Finding top-k maximal cliques in an uncertain graph. In: ICDE, pp. 649–652 (2010)
59. 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)