The VLDB Journal (2017) 26:585–610
DOI 10.1007/s00778-017-0465-6

REGULAR PAPER

Indexing metric uncertain data for range queries and range joins

Lu Chen[1] · **Yunjun Gao**[1,2] · **Aoxiao Zhong**[1] · **Christian S. Jensen**[3] · **Gang Chen**[1,2] · **Baihua Zheng**[4]

Received: 29 August 2016 / Revised: 26 April 2017 / Accepted: 11 May 2017 / Published online: 24 May 2017

Abstract Range queries and range joins in metric spaces have applications in many areas, including GIS, computational biology, and data integration, where *metric uncertain data* exist in different forms, resulting from circumstances such as equipment limitations, high-throughput sequencing technologies, and privacy preservation. We represent metric uncertain data by using an object-level model and a bi-level model, respectively. Two novel indexes, the *uncertain pivot B^+-tree* (UPB-tree) and the *uncertain pivot B^+-forest* (UPB-forest), are proposed in order to support probabilistic range queries and range joins for a wide range of uncertain data types and similarity metrics. Both index structures use a *small* set of *effective* pivots chosen based on a newly defined criterion and employ the B^+-tree(s) as the underlying index. In addition, we present efficient metric probabilistic range query and metric probabilistic range join algorithms, which utilize validation and pruning techniques based on derived probability lower and upper bounds. Extensive experiments with both real and synthetic data sets demonstrate that, compared against existing state-of-the-art indexes for metric uncertain data, the UPB-tree and the UPB-forest incur much *lower* construction costs, consume *less* storage space, and can support *more efficient* metric probabilistic range queries and metric probabilistic range joins.

Keywords Range query · Range join · Uncertain data · Metric space · Index structure

✉ Yunjun Gao
gaoyj@zju.edu.cn

Lu Chen
luchen@zju.edu.cn

Aoxiao Zhong
axzhong@zju.edu.cn

Christian S. Jensen
csj@cs.aau.dk

Gang Chen
cg@zju.edu.cn

Baihua Zheng
bhzheng@smu.edu.sg

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 Department of Computer Science, Aalborg University, Aalborg, Denmark

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

1 Introduction

Given a distance threshold R, a range query returns the objects with distances to a specified query object q that are within R, and a range join returns all pairs of objects that are within distance R of each other. Consider, for example, Fig. 1, given a distance threshold R, a range query returns objects o_1 and o_3 for a query object q, and a range join returns object pairs $\langle q_1, o_1\rangle$, $\langle q_1, o_3\rangle$, $\langle q_2, o_4\rangle$ and $\langle q_3, o_6\rangle$. These operations are common in many areas of computer science, including GIS, computational biology, data integration, to name but a few. In such areas, *uncertainty* arises in different forms, resulting from the limitations of equipments and measurements, high-throughput sequencing technologies, privacy preservation, and so forth, which makes range queries and range joins inaccurate. Hence, we study probabilistic range queries and range joins over uncertain data, with three representative examples given as follows.

Application 1 (*GIS*) In a GIS application, the similarity between locations could be measured by the L_2-norm or the

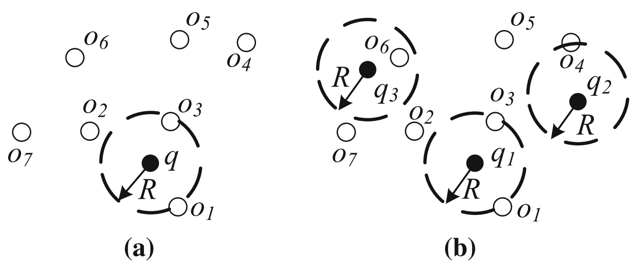

Fig. 1 Illustration of a range query and a range join. **a** Range query and **b** range join

shortest path distance. A range query is able to be utilized to find potential customers for a restaurant, and a range join can be employed to find residents near to each other in order to generate recommendations for carpooling or other social events. Due to privacy preservation, a location can be represented by using an uncertain region in which the resident can locate with a certain probability. To address the uncertainty of locations, probability thresholds are integrated into range queries (range joins), resulting in probabilistic range queries (probabilistic range joins).

Application 2 (*Multimedia retrieval*) In an information retrieval system, the similarity between terms or documents might edit distance, tf-idf, or other metrics. However, the limitation of extraction techniques or typos may incur uncertainty. For instance, due to the typos, "defoliati" can be represented by using two possible meanings "defoliation" and "defoliating," with each assigned a probability. Hence, probabilistic range queries can help to find similar terms to a given one, while probabilistic rang joins can be used to find similar term pairs for data integration.

Application 3 (*Computational biology*) In a protein interaction network, the distance between two proteins that represents how well the proteins interact with each other can be measured by the shortest path, maximum flow, or other metrics. However, the protein data are uncertain due to a number of aspects of the high-throughput sequencing techniques [18]. Therefore, probabilistic range queries can help to find the proteins that are more likely to interact with a specified query protein, and probabilistic range joins can help to find protein pairs that may interact with each other.

Although techniques can be developed for a specific data type, considering a wide range of data types in the aforementioned application scenarios (e.g., locations, strings, and protein sequences), a generic model (i.e., metric space) is desirable that is capable of accommodating not just a single data type, but a wide spectrum. In addition, distance metrics for quantifying the similarity between objects, such as edit distance used for strings, are no longer restricted to the Euclidean distance (i.e., L_2-norm). To accommodate a wide range of similarity notions, we investigate *probabilistic range queries and joins on metric uncertain data*, where *no* assumptions are made about the representations of uncertain objects and where any similarity notion satisfying the *triangle inequality* can be used.

Although many previous efforts exist on indexing uncertain data to accelerate probabilistic range queries and probabilistic range joins [23,26,38,43,44], they target mainly Euclidean space or, more generally, vector spaces. Unfortunately, these models of uncertain data and geometric properties are restricted to vector spaces, rendering them inapplicable to more complex data (e.g., sets) that cannot fit into Euclidean models. Hence, Angiulli and Fassetti [3] first study the indexing on metric uncertain data and present the UP-Index. While representing an important advance, this index can be further improved in several ways because (i) it processes metric uncertain data using the vector space uncertain model, which limits its applicability to complex metric uncertain data such as uncertain sets and uncertain graphs; (ii) it stores *unnecessary* information which leads to *redundancy*; (iii) it has to scan the *entire* index, for any range query, in order to prune away objects using probability upper bounds; and (iv) it cannot validate objects due to *no probability lower bounds*.

In order to design efficient external memory indexes on metric uncertain data, three challenging issues must be addressed. First, *how shall we model metric uncertain data in a more general manner than using the vector space uncertain model?* We propose two metric uncertain data models to represent the uncertainty in metric spaces. Second, *how can we support efficient metric probabilistic range queries and metric probabilistic range joins?* Here, efficiency is measured by both the number of distance computations (i.e., CPU cost) and the number of page accesses (i.e., I/O cost). We tackle this by identifying a small set of effective pivots to map uncertain objects from the metric space to the vector space, based on which probability lower and upper bounds are derived to validate and prune uncertain objects. Moreover, MBBs and a space-filling curve (e.g., Hilbert curve, *Z*-order curve) are utilized to cluster uncertain data to reduce the I/O cost. Third, *how do we achieve low costs of index storage*, *construction*, *and manipulation?* To save storage cost, our solution only stores necessary information and makes use of space-filling curve-based dimensionality reduction. To support efficient index construction and manipulation, we use the B^+-tree(s) as the underlying index.

The resulting proposals are called the *uncertain pivot* B^+*-tree* (UPB-tree) and the *uncertain pivot* B^+*-forest* (UPB-forest). These are based on two proposed metric uncertain data models. They do not depend on the detailed representations of objects and then can support all the distance metrics that satisfy the triangle inequality. To sum up, the key contributions of this paper are as follows:

- We develop two metric uncertain data models with corresponding disk-based index structures, the UPB-tree and the UPB-forest, which map uncertain objects from a metric space to a vector space using a set of pivots, utilize a space-filling curve for dimensionality reduction, and employ the B^+-tree(s) as the underlying index.
- We propose efficient algorithms for metric probabilistic range queries and joins that use the validation and pruning techniques based on derived lower and upper bounds of the probabilities.
- We present a pivot selection algorithm based on a new criterion to choose a small set of effective pivots.
- We conduct extensive experiments using real and synthetic data sets to show that the UPB-tree and the UPB-forest outperform the state-of-the-art competitors on metric uncertain data in terms of the number of distance computations and the number of page accesses.

The paper extends a preliminary study [8]. The extensions include support for the efficient processing of (i) two additional interesting queries, i.e., metric probabilistic range queries with uncertain query objects (Sects. 4.3, 5.3) and metric probabilistic range joins (Sects. 4.4, 5.4), based on the UPB-tree and the UPB-forest; (ii) an enhanced experimental evaluation that incorporates the new classes of queries (Sect. 7); and (iii) more comprehensive coverage of related work (Sect. 2). Also, we have revised the introduction and have contained additional theoretical analysis.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 formalizes the problem. Sections 4 and 5 elaborate the indexing structures with their corresponding metric probabilistic range query and metric probabilistic range join algorithms. Section 6 presents the pivot selection algorithm. Experimental findings are reported in Sect. 7. Finally, Sect. 8 concludes the paper and offers some directions for future work.

2 Related work

We proceed to review related work on indexing techniques for probabilistic range queries and range joins, and then survey metric range queries and range joins.

2.1 Probabilistic range queries and range joins

Given a set U of uncertain objects, a range region RR, and a probability threshold θ, a probabilistic range query returns the uncertain objects $u \in U$ with $Pr(u \in \text{RR}) \geq \theta$. Cheng et al. [10] first address probabilistic range queries on one-dimensional uncertain data. Then, R-tree-based approaches [27,36] are proposed, in which the MBB of an object serves as the summary of its PDF. Tao et al. [38] introduce the U-tree, as the PDF summary of an object is a finite set of probabilistically constrained regions. Next, range queries with constraints, where PDFs of uncertain data follow Gaussian or Uniform distributions, are studied [4,6,16]. Aggarwal and Yu [2] show how to construct an effective index structure in order to handle uncertain range queries in high dimensionality. Assuming that PDFs of objects are histograms, Agarwal et al. [1] provide a thorough theoretical analysis of various indexing schemes with linear or near-linear space and logarithmic query time. More recently, Zhang et al. [43] present the UI-tree, assuming that every uncertain object is a set of groups that are disjointed partitions of its PDF. Kimura et al. [23] propose a primary indexing technique named UPI to speed up query processing on uncertain data by clustering heap files. Zhang et al. [44] develop a U-Quadtree, which employs a Quadtree to organize multi-dimensional uncertain objects. Zhu et al. [45] present MRST and R-MRST, which are data structures that enable the effective and efficient indexing of uncertain objects. In addition, indexing techniques over one-dimensional uncertain data are explored [11,24], which divide the interval of an uncertain object into several disjoint subintervals.

Given two sets U and V of uncertain objects, a query distance R, and a probability threshold θ, a probabilistic range join (also termed as probabilistic similarity join) returns all uncertain object pairs $\langle u, v\rangle$ ($\in U \times V$) with $Pr(d(u, v) \leq R) \geq \theta$. Cheng et al. [9] and Silva et al. [34] deal with probabilistic range joins on one-dimensional uncertain data. Kriegel et al. [25] review all types of probabilistic joins. Kriegel et al. [26] propose an R-tree-based approach to accelerate probabilistic range joins.

The above approaches have shortcomings in relation to range queries and range joins on metric uncertain data. First, the uncertain models used are restricted to vector spaces, rendering the approaches inapplicable to complex data (e.g., sets, DNA sequences). Second, to accelerate query processing, they utilize *geometric properties* (e.g., MBB [26,27,36], grid [44]) that are *unavailable* in metric spaces.

To the best of our knowledge, only Angiulli and Fassetti [3] study the indexing of metric uncertain data. They present a pivot-based index, the UP-Index, that stores pre-computed histograms of the probability distribution for every uncertain object w.r.t. a set of pivots. The index enables pruning of uncertain objects during search using the upper bound of the probability derived by the reverse triangle inequality. Although the UP-Index supports any distance metric that satisfies the triangle inequality, it is still based on the vector space uncertain model, and it has to transform non-vector uncertain data into vectors before building the index. Section 7 reports on performance studies to show that the UP-Index is not competitive in terms of distance computations and page accesses. This is because: (i) UP-Index needs to store pre-computed histograms w.r.t. the whole distance range, resulting in *larger* storage cost; (ii) for any probabilistic range query, it needs to scan the whole index in order to prune uncertain objects using the probability upper bounds as it has no early termination conditions or batch pruning techniques; and (iii) it cannot validate objects since it has *no probability lower bounds*.

In addition, some efforts have been devoted to probabilistic range queries [13,22], similarity search [14,17], and similarity joins [21,28] for specific metric uncertain data (e.g., sets, data series, graphs). These works utilize uncertain models and algorithms particular to the specific metric uncertain data they target. Thus, they do not support efficiently probabilistic range queries and probabilistic range joins over metric uncertain data.

2.2 Metric range queries and range joins

Given an object set O, a query object q, and a search radius R in a metric space, a metric range query retrieves the objects in O with distances to q bounded by R. To accelerate metric range queries, a large number of metric indexing structures are proposed, which can be generally classified into two categories, compact partitioning methods [12,39] and pivot-based methods [7,30,37,40,41]. For two objects q and o, the distance $d(q, o)$ cannot be smaller than $|d(q, p) - d(o, p)|$ for any pivot p, due to the triangle inequality. Hence, it may be possible to prune an object o as a match for q using the lower bound $|d(q, p) - d(o, p)|$ instead of calculating $d(q, o)$. This capability enables pivot-based approaches to outperform compact partitioning methods in terms of the number of distance computations, a key performance criterion in metric spaces. Hence, in this paper, we adopt the pivot techniques.

Given two object sets Q and O, a metric range join (also termed as metric similarity join) retrieves pairs of objects from $Q \times O$ that are within distance R of each other. This operator has been explored in metric spaces, and efficient solutions exist [20]. Recently, Paredes and Reyes [31] handle similarity joins using LTC, which indexes two sets jointly. Fredriksson and Braithwaite [15] improve the Quickjoin algorithm [20] for similarity joins. Silva and Pearson [32,35] develop a nonblocking similarity join operator, DBSimJoin, and study index-based similarity joins. Also, metric range joins using map-reduce are investigated in [33,42].

The above efforts do not consider uncertainty, and therefore, they do not solve the problem we study.

3 Problem formulation

We review briefly properties of the metric space, present metric uncertain data models, and formalize the probabilistic range query and join in the metric space. Table 1 summarizes the notations used frequently throughout this paper.

Table 1 Symbols used frequently

Notation	Description
U, V	Sets of uncertain objects
u_i, v_j	Uncertain objects in U or V
u_{ij}	An instance object of an uncertain object u_i
m	The number of instances for each uncertain object
$\mathcal{G}_{p,u_i}$	The PDF of the distance between u_i and a pivot p
g_{p,u_i}	The approximation of $\mathcal{G}_{p,u_i}$
MPRQ(q, R, θ)	A metric probabilistic range query w.r.t. query object q, search radius R, probability threshold θ
MPRQU(uq, R, θ)	An MPRQ w.r.t. uncertain query object uq, search radius R, and probability threshold θ
MPRJ(U, V, R, θ)	A metric probabilistic range join w.r.t. U, V, search radius R, and probability threshold θ
P	A set of pivots p_t
$d()$	The distance function in a metric space
$D()$	The L_∞-norm metric in a mapped vector space
d^+	The maximum distance in a metric space
n, s	The number of slots or the intervals of each slot
R_q	A circular search region (in a metric space) centered at a query object q with radius R
$\phi_{p_t}(R_q), \phi(R_q)$	R_q in the mapped vector space using p_t or P
$\phi(q), \phi(u_i)$	q or u_i in the mapped vector space
$\pi(u_i, R_q)$	The probability of u_i contained in R_q
$l\pi(u_i, R_q)$	The lower bound of $\pi(u_i, R_q)$
$u\pi(u_i, R_q)$	The upper bound of $\pi(u_i, R_q)$
$\pi(u_i, v_j)$	The probability $Pr(d(u_i, v_j) \le R)$
$l\pi(u_i, v_j)$	The lower bound of $\pi(u_i, v_j)$
$u\pi(u_i, v_j)$	The upper bound of $\pi(u_i, v_j)$

3.1 Metric spaces

A metric space is given by a tuple (M, d), where M is an object domain and d is a distance function quantifying "similarity" between objects in M. The distance function d has four properties: (i) *symmetry*: $d(q, o) = d(o, q)$; (ii) *nonnegativity*: $d(q, o) \ge 0$; (iii) *identity*: $d(q, o) = 0$ iff $q = o$; and (iv) *triangle inequality*: $d(q, o) \le d(q, p) + d(p, o)$.

3.2 Metric uncertain data models

An uncertain object is typically represented in one of the two ways: (i) using all possible instances, each with an assigned probability, or (ii) using a probability density function (PDF). Hence, we present two metric uncertain data models.

Object-level model Every uncertain object u_i can be modeled by all the possible instances and their corresponding

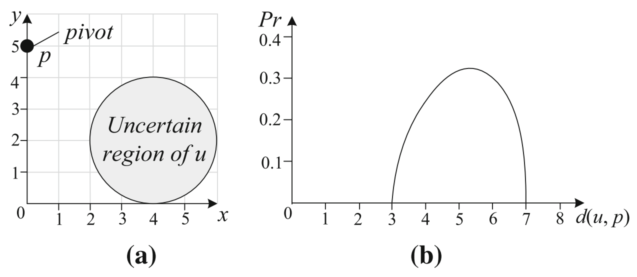

Fig. 2 Example of the bi-level model. **a** Uncertain object u and **b** continuous PDF of $d(u, p)$

nonnegative probabilities $\{\langle u_{i1}, Pr(u_{i1})\rangle, \langle u_{i2}, Pr(u_{i2})\rangle, \ldots, \langle u_{im}, Pr(u_{im})\rangle\}$ such that $\sum_{j=1}^{m} Pr(u_{ij}) = 1$, in which $Pr(u_{ij})$ denotes the nonnegative probability of u_i being u_{ij}.

As an example, an uncertain English word u can be modeled as $\{\langle$"defoliation," 0.2$\rangle$, $\langle$"defoliated," 0.1$\rangle$, $\langle$"defoliate," 0.3$\rangle$, $\langle$"defoliates," 0.4$\rangle\}$.

Bi-level model Each uncertain object u_i can utilize any other uncertainty model as the underlying model, based on which it can be further represented by using the PDF of its distance to a pivot p, satisfying that $\int_0^{d^+} \mathcal{G}_{p,u_i}(x)\mathrm{d}x = 1$ or $\sum_0^{d^+} \mathcal{G}_{p,u_i}(x) = 1$ for *continuous* or *discrete* PDFs, respectively. Here, x is the distance $d(u_i, p)$ from u_i to p, and d^+ is the maximum distance of $d(u_i, p)$.

To support query processing for a wide range of uncertain data types, the bi-level model can use any specific uncertain model (e.g., the vector space uncertain model [10,16,44] or the p-set model [17]) as its underlying model, based on which an upper model is built to represent its distance uncertainty. Consider the example of the bi-level model built on the vector space uncertain model shown in Fig. 2. The uncertain object u locates uniformly within the uncertain region, as illustrated in Fig. 2a, and the PDF of the distance $d(u, p)$ is depicted in Fig. 2b.

For the bi-level model, although a PDF might be both discrete and continuous, we concentrate on the continuous case for simplicity. However, for the object-level model, we only aim at the discrete case according to its definition.

3.3 Metric probabilistic range query

Given a query object q and a search radius R in a metric space, a *metric range query* finds the objects with distances to q within R, i.e., the objects located into a *circular search region* R_q (centered at q with radius R). Considering uncertainty, let $\pi(u_i, R_q)$ be the probability that a metric uncertain object u_i is contained in R_q, i.e., $\pi(u_i, R_q) = Pr(d(q, u_i) \le R)$. We formally define the probabilistic range query on metric uncertain data.

Definition 3.1 (*Metric probabilistic range query*) Given a set U of uncertain objects, a certain query object q, a search radius R, and a probability threshold θ $(0 < \theta \le 1)$ in a metric space, a metric probabilistic range query (MPRQ) finds the uncertain objects in U whose probabilities of being contained in R_q are no smaller than θ, i.e., MPRQ$(q, R, \theta) = \{u_i \mid u_i \in U \wedge \pi(u_i, R_q) \ge \theta\}$.

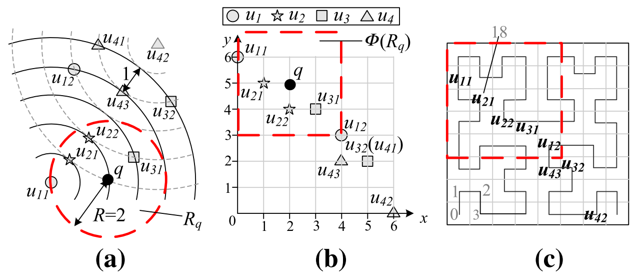

Fig. 3 Example of MPRQ using the object-level model. **a** Metric space, **b** vector space, **c** Hilbert curve

Consider, for example, the uncertain object set $U = \{u_1, u_2, u_3, u_4\}$, illustrated in Fig. 3a, with $u_1 = \{(u_{11}, 0.6), (u_{12}, 0.4)\}$, $u_2 = \{(u_{21}, 0.5), (u_{22}, 0.5)\}$, $u_3 = \{(u_{31}, 0.4), (u_{32}, 0.6)\}$, and $u_4 = \{(u_{41}, 0.2), (u_{42}, 0.3), (u_{43}, 0.5)\}$ using the object-level model. Assuming that the L_2-*norm* is used as the distance function, the result of MPRQ$(q, 2, 0.5)$ is $\{u_1, u_2\}$, as $\pi(u_1, R_q) = 0.6$ and $\pi(u_2, R_q) = 1$.

However, the query object q might be uncertain, due to the limitations of equipments and measurements, privacy preservation, and so forth. For example, in Application 2 stated in Sect. 1, the query object (i.e., a query term input by a user) could be uncertain due to typos. Thus, we also define the metric probabilistic range query with an uncertain query object.

Definition 3.2 (*Metric probabilistic range query with uncertain query object*) Given a set U of uncertain objects, an uncertain query object uq, a search radius R, and a probability threshold θ $(0 < \theta \le 1)$ in a metric space, a metric probabilistic range query with an uncertain query object (MPRQU) finds the uncertain objects in U whose probabilities of being contained in the search region R_{uq} are no smaller than θ, i.e., MPRQU$(uq, R, \theta) = \{u_i | u_i \in U \wedge \pi(u_i, R_{uq}) \ge \theta\}$.

Consider, for instance, the uncertain object set $U = \{u_2, u_3, u_4\}$ and $uq = u_1$ using the object-level model, as illustrated in Fig. 3a. Assume that the L_2-*norm* is used as the distance function, the result of MPRQ$(uq, 2, 0.5)$ is $\{u_2\}$, as $\pi(u_2, R_{uq}) = 0.6$.

Note that, R_{uq} for an uncertain query object uq could be m circular search regions R_{q_j} $(1 \le j \le m)$ if using the object-level model as discussed in Sect. 4.3, or one continuous search region if using the bi-level model as to be discussed in Sect. 5.3.

3.4 Metric probabilistic range join

The metric probabilistic range query only involves a single query object. In the following, we consider a set of uncertain query objects. For example, in Application 2 stated in Sect. 1, we need to find similar objects for each of the query objects during the data integration. Motivated by this, we also introduce the metric probabilistic range join as follows.

Definition 3.3 (*Metric probabilistic range join*) Given two sets U and V of uncertain objects, a search radius R, and a probability threshold θ ($0 < \theta \leq 1$) in a metric space, a metric probabilistic range join (MPRJ) finds the uncertain object pairs $\langle u_i, v_j \rangle$ in $U \times V$ whose probabilities of distances $d(u_i, v_j)$ being no larger than R are no smaller than θ, i.e., MPRJ$(U, V, R, \theta) = \{\langle u_i, v_j \rangle | u_i \in U \wedge v_j \in V \wedge \pi(u_i, v_j) \geq \theta\}$, where $\pi(u_i, v_j) = Pr(d(u_i, v_j) \leq R)$.

Consider, for example, two uncertain object sets $U = \{u_1, u_3\}$ and $V = \{u_2, u_4\}$ as depicted in Fig. 3a, with $u_1 = \{(u_{11}, 0.6), (u_{12}, 0.4)\}$, $u_2 = \{(u_{21}, 0.5), (u_{22}, 0.5)\}$, $u_3 = \{(u_{31}, 0.4), (u_{32}, 0.6)\}$, and $u_4 = \{(u_{41}, 0.2), (u_{42}, 0.3), (u_{43}, 0.5)\}$, using the object-level model. If the L_2-*norm* is used as the distance function, the result of a MPRJ$(U, V, 2, 0.5)$ is $\{\langle u_1, u_2 \rangle\}$, as $\pi(u_1, u_2) = 0.6$.

4 The UPB-tree

We propose a new index structure, the *UPB-tree*, to index uncertain objects based on the object-level model, and then, we present the corresponding range query and join algorithms. Also, we analyze query costs.

4.1 UPB-tree structure

UPB-tree construction is based on a two-stage mapping. In the first stage, we map all instances of metric uncertain objects to data points in a vector space, using selected pivots. We prefer the vector space to the metric space because it provides geometric and coordinate information that is not available in the metric space. In the second stage, we utilize a proximity-preserving *space-filling curve* (SFC) to map the data points in the vector space to integers in one-dimensional space. Finally, we use a B$^+$-tree with MBB information to index the metric uncertain objects.

Pivot mapping Given a pivot set $P = \{p_1, p_2, \ldots, p_{|P|}\}$, a metric space (M, d) can be mapped into a vector space $(R^{|P|}, L_\infty)$. Specifically, any instance object u_{ij} in the metric space can be represented as a point $\phi(u_{ij}) = \langle d(u_{ij}, p_1), d(u_{ij}, p_2), \ldots, d(u_{ij}, p_{|P|}) \rangle$ in the vector space. Consider Fig. 3, in which $U = \{u_1, u_2, u_3, u_4\}$. If $P = \{u_{11}, u_{42}\}$, U can be mapped to a two-dimensional vector space as depicted in

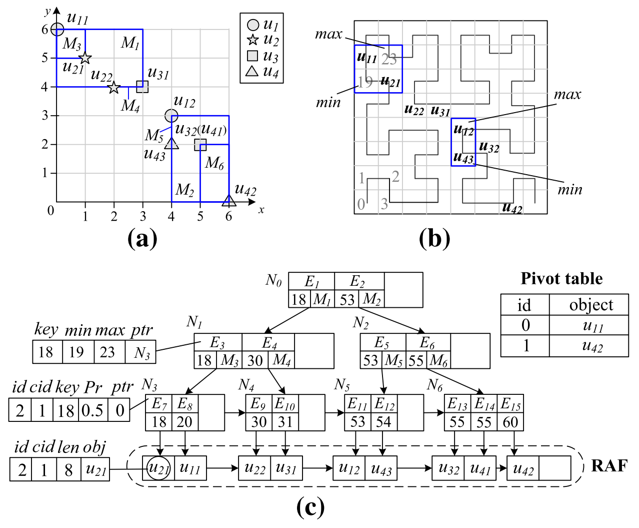

Fig. 4 Example of the UPB-tree for the dataset in Fig. 3. **a** MBBs of UPB-tree, **b** MBB representation, **c** the UPB-tree

Fig. 3b, where the x axis denotes $d(u_{ij}, u_{11})$ and the y axis represents $d(u_{ij}, u_{42})$.

Given a pivot set P, $d(u_{ij}, u'_{ij}) \geq \max\{|d(u_{ij}, p) - d(u'_{ij}, p)| \mid p \in P\} = D(\phi(u_{ij}), \phi(u'_{ij}))$ according to the triangle inequality, in which $D(\,)$ is an L_∞-norm metric. Clearly, the distance in the mapped vector space is a *lower bound* on that in the original metric space. For example, in Fig. 3, $d(u_{43}, u_{32}) \geq D(\phi(u_{43}), \phi(u_{32})) = 1$.

Space-filling curve mapping Given a vector $\phi(u_{ij})$ after the pivot mapping, an SFC maps $\phi(u_{ij})$ to an integer SFC$(\phi(u_{ij}))$. Considering the example shown in Fig. 3c, SFC$(\phi(u_{21})) = 18$. Although we employ the Hilbert curve in this paper, any SFC is applicable to the UPB-tree.

A UPB-tree used to index a metric uncertain object set based on the object-level model contains three parts, i.e., a pivot table, a *random access file* (RAF), and a B$^+$-tree. Figure 4 illustrates a UPB-tree that indexes $U = \{u_1, u_2, u_3, u_4\}$. The pivot table stores the pivots (u_{11} and u_{42}) that are used to map a metric space into a vector space. The RAF stores instance objects in ascending order of their SFC values as they appear in the B$^+$-tree, in order to enhance efficiency. Each RAF entry records (i) an uncertain object identifier *id*, (ii) an instance identifier *cid*, (iii) the storage overhead *len* (in bytes) of the instance object, and (iv) the real instance object *obj*. In Fig. 4c, the RAF entry associated with the object u_{21} records the uncertain object identifier 2, the instance identifier 1, the storage overhead 8 (bytes), and the real instance object u_{21}.

A B$^+$-tree is employed to index the SFC values of instance objects. The structure of a node entry varies, dependent on whether it is an entry in a leaf node or an entry in a nonleaf node. A leaf entry in the leaf node (e.g., N_3, N_4, N_5, and N_6) of the B$^+$-tree records (i) an uncertain object identifier *id*, (ii) an instance identifier *cid*, (iii) a SFC value *key*, (iv) a probability *Pr*, and (v) a pointer *ptr* to the real instance object, which is the address of the instance object kept in the RAF. As an example depicted in Fig. 4c, the leaf entry E_7 associated with u_{21} records the uncertain object identifier 2, the instance identifier 1, the Hilbert value 18, the probability 0.5, and the storage address 0 of u_{21} in the RAF. A nonleaf entry in the root or intermediate node (e.g., N_0, N_1, and N_2) of the B^+-tree records (i) the minimum SFC value *key* in its subtree, (ii) the pointer *ptr* to the root node of its subtree, and (iii) the SFC values *min* and *max* for $\langle a_1, a_2, \ldots, a_{|P|}\rangle$ and $\langle b_1, b_2, \ldots, b_{|P|}\rangle$, to represent the MBB M_i $(= \{[a_t, b_t] \mid t \in [1, |P|]\})$ of the root node N_i of its subtree. For instance, the nonleaf entry E_3 uses *min* (= 19) and *max* (= 23) to denote the M_3 of N_3.

Note that, while some existing index structures (e.g., the R-tree) can also be used to index the mapped vector space, the UPB-tree uses a B^+-tree to index the SFC values of instance objects after the pivot mapping. This is attractive because the use of an SFC can cluster objects into compact regions and meanwhile reduce the amount of storage needed for pre-computed distances, as to be verified by our experiments in Sect. 7.

4.2 UPB-tree-based MPRQ algorithm

Given a set U of uncertain objects, MPRQ finds the uncertain objects u_i in U with probabilities of being contained in R_q no smaller than θ, i.e., $\pi(u_i, R_q) \geq \theta$. Since an uncertain object u_i can be represented by all instance objects u_{ij} with associated nonnegative probabilities based on the object-level model, $\pi(u_i, R_q)$ can be calculated by summarizing all the probabilities of its instance objects u_{ij} $(\in R_q)$. Thus, to derive $\pi(u_i, R_q)$ of an uncertain object u_i, a naive solution is to check every instance object u_{ij} and determine whether u_{ij} is contained in R_q. However, as stated in Lemma 4.1, we only need to verify the instance objects u_{ij} with $\phi(u_{ij}) \in \phi(R_q)$, where $\phi(R_q)$ denotes the *mapped search region* in the vector space using a pivot set P. For example, the dashed circle in Fig. 3a represents R_q, and the dashed rectangle in Fig. 3b denotes $\phi(R_q)$ using $P = \{u_{11}, u_{42}\}$.

Lemma 4.1 *Given a pivot set P, if an instance object u_{ij} is enclosed in R_q, then $\phi(u_{ij})$ is contained in the mapped range region $\phi(R_q)$, where $\phi(R_q) = \{\langle x_1, x_2, \ldots, x_{|P|}\rangle | 1 \leq t \leq |P| \wedge x_t \geq 0 \wedge x_t \in [d(q, p_t) - R, d(q, p_t) + R]\}$.*

Proof Assume, to the contrary, that there exists an instance object $u_{ij} \in R_q$ but $\phi(u_{ij}) \notin \phi(R_q)$, i.e., $\exists p_t \in P$, $d(u_{ij}, p_t) > d(q, p_t) + R$ or $d(u_{ij}, p_t) < d(q, p_t) - R$. If $d(u_{ij}, p_t) > d(q, p_t) + R$ or $d(u_{ij}, p_t) < d(q, p_t) - R$, then $d(q, u_{ij}) \geq |d(u_{ij}, p_t) - d(q, p_t)| > R$ due to the triangle inequality. This contradicts our assumption that $u_{ij} \in R_q$. □

Based on Lemma 4.1, if the MBB of a node N does not overlap $\phi(R_q)$, we can prune N since the instance objects u_{ij} contained in N cannot contribute to the value of $\pi(u_i, R_q)$. Here, MBB can be obtained easily by using the SFC values *min* and *max* stored in the UPB-tree. In the MPRQ example shown in Figs. 3 and 4, N_6 can be discarded due to $M_6 \cap \phi(R_q) = \emptyset$.

When computing $\pi(u_i, R_q)$, Lemma 4.1 is used to avoid distance computations for instance objects u_{ij} having $\phi(u_{ij}) \notin \phi(R_q)$. Nonetheless, we still have to verify the instance objects u_{ij} with $\phi(u_{ij})$ enclosed in $\phi(R_q)$. Therefore, we develop Lemma 4.2 that enables us to further avoid unnecessary distance computations.

Lemma 4.2 *Given a pivot set P, for an instance object u_{ij}, if there exists a pivot p_t $(\in P)$ satisfying $d(u_{ij}, p_t) \leq R - d(q, p_t)$, then u_{ij} is included in R_q.*

Proof Given a query object q, an instance object u_{ij}, and a pivot p_t, we have $d(q, u_{ij}) \leq d(u_{ij}, p_t) + d(q, p_t)$ due to the triangle inequality. If $d(u_{ij}, p_t) \leq R - d(q, p_t)$, then $d(q, u_{ij}) \leq R - d(q, p_t) + d(q, p_t) = R$. Thus, u_{ij} is certainly contained in R_q. □

Consider the MPRQ example in Fig. 3. If $R = 3$, then for an object u_{21}, there exists a pivot u_{11} satisfying $d(u_{21}, u_{11}) = R - d(q, u_{11}) = 3 - 2 = 1$. Hence, u_{21} is guaranteed to be contained in R_q, and there is no need for any further distance computation of $d(q, u_{21})$.

To speed up MPRQ processing, we can also validate or prune uncertain objects u_i using the upper and lower bounds of $\pi(u_i, R_q)$, as stated in Theorem 4.1.

Theorem 4.1 *Given an MPRQ(q, R, θ), let $u\pi(u_i, R_q)$ and $l\pi(u_i, R_q)$ be the upper and lower bounds of $\pi(u_i, R_q)$, respectively. Then, (i) u_i can be safely pruned if $u\pi(u_i, R_q) < \theta$, and (ii) u_i can be validated if $l\pi(u_i, R_q) \geq \theta$.*

The proof of Theorem 4.1 is straightforward and thus omitted. To use Theorem 4.1, we have to derive *tight* upper bound $u\pi(u_i, R_q)$ and lower bound $l\pi(u_i, R_q)$. According to Lemma 4.1, by summarizing all the probabilities of instances u_{ij} for an uncertain object u_i with $\phi(u_{ij}) \in \phi(R_q)$, we can get the upper bound of $\pi(u_i, R_q)$ as follows.

Lemma 4.3 *Given an MPRQ(q, R, θ) and a pivot set P, $u\pi(u_i, R_q) = \sum_{\phi(u_{ij})\in\phi(R_q)} Pr(u_{ij})$.*

Proof Let $S_{u_i} = \{u_{ij} | u_{ij} \in R_q\}$ and $S_{\phi(u_i)} = \{u_{ij} | \phi(u_{ij}) \in \phi(R_q)\}$. We have $S_{u_i} \subseteq S_{\phi(u_i)}$ according to Lemma 4.1. Therefore, $\pi(u_i, R_q) = \sum_{u_{ij}\in S_{u_i}} Pr(u_{ij}) \leq \sum_{u_{ij}\in S_{\phi(ui)}} Pr(u_{ij}) = \sum_{\phi(u_{ij})\in\phi(R_q)} Pr(u_{ij})$. Consequently, $u\pi(u_i, R_q)$ can be set as $\sum_{\phi(u_{ij})\in\phi(R_q)} Pr(u_{ij})$. □

Consider the MPRQ(q, 2, 0.5) example in Fig. 3. We have $u\pi(u_1, R_q) = Pr(u_{11}) + Pr(u_{12}) = 1$ and $u\pi(u_3, R_q) =$

$Pr(u_{31}) = 0.4$, since $\phi(u_{11})$, $\phi(u_{12})$, and $\phi(u_{31})$ are inside $\phi(R_q)$. Hence, u_3 can be pruned by Theorem 4.1 because $u\pi(u_3, R_q) < \theta$. It is worth noting that, for u_i that cannot be filtered, when computing an accurate $\pi(u_i, R_q)$ during the verification phase, we can further tighten $u\pi(u_i, R_q)$ to improve the pruning power. Specifically, if $\phi(u_{ik}) \in \phi(R_q)$ but $d(q, u_{ik}) > R$, then $u\pi(u_i, R_q)$ can be tightened to $\sum_{\phi(u_{ij}) \in \phi(R_q)} Pr(u_{ij}) - Pr(u_{ik})$. As an example, in Fig. 3, $u\pi(u_1, R_q)$ can be reduced to 0.6 as $\phi(u_{12}) \in \phi(R_q)$ but $d(q, u_{12}) \geq 2$.

Next, we derive the lower bound of $\pi(u_i, R_q)$ using Lemma 4.2, by accumulating all the probabilities of the instance objects satisfying the condition of Lemma 4.2.

Lemma 4.4 *Given an MPRQ(q, R, θ) and a pivot set P,* $l\pi(u_i, R_q) = \sum_{\exists p_t \in P s.t. d(u_{ij}, p_t) \leq R - d(q, p_t)} Pr(u_{ij})$.

Proof Let $S_{u_i} = \{u_{ij} | \, u_{ij} \in R_q\}$ and $S_P = \{u_{ij} | \, \exists p_t (\in P)$ s.t. $d(u_{ij}, p_t) \leq R - d(q, p_t)\}$. Based on Lemma 4.2, we can get that $S_{u_i} \supseteq S_P$. Then, $\pi(u_i, R_q) = \sum_{u_{ij} \in S_{u_i}} Pr(u_{ij}) \geq \sum_{u_{ij} \in S_P} Pr(u_{ij}) = \sum_{\exists p_t \in P \text{ s.t. } d(u_{ij}, p_t) \leq R - d(q, p_t)} Pr(u_{ij})$. Thus, $l\pi(u_i, R_q)$ can be set to $\sum_{\exists p_t \in P \text{ s.t. } d(u_{ij}, p_t) \leq R - d(q, p_t)} Pr(u_{ij})$. □

Consider the MPRQ(q, 2, 0.5) example depicted in Fig. 3 again. We have $l\pi(u_1, R_q) = Pr(u_{11}) = 0.6$, as pivot u_{11} satisfies $d(u_{11}, u_{11}) \leq 2 - d(q, u_{11})$. Also, we can tighten $l\pi(u_i, R_q)$ during the verification phase. Specifically, if u_{ik} does not satisfy the condition of Lemma 4.2 but $u_{ik} \in R_q$, then $l\pi(u_i, R_q)$ can also be tightened to $\sum_{\exists p_t \in P \text{ s.t. } d(u_{ij}, p_t) \leq R - d(q, p_t)} Pr(u_{ij}) + Pr(u_{ik})$.

Algorithm 1 UPB-tree based MPRQ Algorithm (UtMA)

```
Input: $q$, $R$, $\theta$, an uncertain object set $U$ indexed by a UPB-tree
Output: the result set $S_r$ of $MPRQ(q, R, \theta)$
/* $S_c$: a set of candidate uncertain objects; $IS_c$: a candidate instance set. */
1: compute $\phi(q)$ and $\phi(R_q)$ using a pivot table $P$  // $\phi(q) = \langle d(q, p_t) \mid p_t \in P \rangle$
2: push the root node of the B$^+$-tree into a heap $H$
3: while $H \neq \emptyset$ do
4:    de-heap the top node $N$ from $H$
5:    if $N$ is a non-leaf node then
6:       for each entry $e$ in $N$ do
7:          if $MBB(e.ptr) \cap \phi(R_q) \neq \emptyset$ then
8:             push $e.ptr$ into $H$
9:    else  // $N$ is a leaf node
10:      if $MBB(N) \subseteq \phi(R_q)$ then
11:         for each entry $e$ in $N$ do
12:            if $e.id \notin S_c$ then insert $e.id$ into $S_c$
13:            update $u\pi(e.id, R_q)$ using Lemma 4.3
14:            update $l\pi(e.id, R_q)$ or insert $e.ptr$ into $IS_c$ using Lemma 4.4
15:      else
16:         for each entry $e$ in $N$ do
17:            if $\phi(e.ptr) \in \phi(R_q)$ then  // $\phi(e.ptr)$ is obtained by $e.key$
18-20:            these lines are identical to lines 12-14
21: for each $u_i$ in $S_c$ do
22:    if $l\pi(u_i, R_q) \geq \theta$ then  // Theorem 4.1
23:       insert $u_i$ into $S_r$ and remove all $u_{ik}$ from $IS_c$
24:    else if $u\pi(u_i, R_q) < \theta$ then remove all $u_{ik}$ form $IS_c$  // Theorem 4.1
25: for each $u_{ij}$ in $IS_c$ do
26:    if $d(q, u_{ij}) \leq R$ then update $l\pi(u_i, R_q)$
27:    else update $u\pi(u_i, R_q)$
28:    if $l\pi(u_i, R_q) \geq \theta$ then
29:       insert $u_i$ into $S_r$ and remove all $u_{ik}$ from $IS_c$
30:    else if $u\pi(u_i, R_q) < \theta$ then remove all $u_{ik}$ from $IS_c$
31: return $S_r$
```

Based on Theorem 4.1 and Lemmas 4.1 to 4.4, we develop a *UPB-tree-based MPRQ algorithm* (UtMA). The algorithm follows a filtering-and-refinement framework. The pseudo-code is presented in Algorithm 1. The algorithm takes as inputs q, R, θ, and an uncertain object set U indexed by a UPB-tree and outputs the result set S_r. In the filtering phase, UtMA first computes $\phi(q)$ and $\phi(R_q)$ using a pivot table P (line 1). Then, it traverses the B$^+$-tree in a depth-first manner (lines 2–20) to find candidate uncertain objects (maintained in a set S_c) and the uncertain instance objects needed further verification (preserved in a set IS_c) using Lemmas 4.1 and 4.2, and to update $u\pi(u_i, R_q)$ and $l\pi(u_i, R_q)$ for uncertain objects u_i in S_c via Lemmas 4.3 and 4.4. Next, UtMA validates or eliminates the uncertain objects u_i in S_c by Theorem 4.1, and removes corresponding uncertain instance objects u_{ik} from IS_c (lines 21–24). After that, during the refinement phase, UtMA first verifies every uncertain instance object u_{ij} in IS_c in order to tighten $u\pi(u_i, R_q)$ and $l\pi(u_i, R_q)$ for the corresponding uncertain object u_i (lines 26–27). Then, it validates or prunes u_i based on Theorem 4.1, and removes the corresponding uncertain instance objects u_{ik} from IS_c (lines 28–30). Finally, S_r is returned (line 31).

Example 1 Please refer to the conference paper [8] for details and thus omitted here. □

4.3 UPB-tree-based MPRQU algorithm

Since the query object q of an MPRQ might be uncertain (as discussed in Sect. 3.3), we extend MPRQ to take an uncertain query object as argument, obtaining a new query type called MPRQU. As an uncertain query object uq can be modeled as $\{\langle q_1, Pr(q_1)\rangle, \langle q_2, Pr(q_2)\rangle, \ldots, \langle q_m, Pr(q_m)\rangle\}$ based on the object-level model, the range region R_{uq} for uq consists of m circular range regions R_{q_x} $(1 \leq x \leq m)$, and thus, $\pi(u_i, R_{uq})$ can be calculated by summarizing all the probabilities of $\pi(u_i, R_{q_x}) \times Pr(q_x)$ $(q_x \in uq)$. In order to avoid the high cost of computing $\pi(u_i, R_{uq})$, we derive upper and lower bounds of $\pi(u_i, R_{uq})$ as follows, to prune and validate uncertain objects using Theorem 4.1.

Lemma 4.5 *Given an MPRQU(uq, R, θ) and a pivot set P,* $u\pi(u_i, R_{uq}) = \sum_{x=1}^{m} \sum_{\phi(u_{ij}) \in \phi(R_{q_x})} (Pr(u_{ij}) \times Pr(q_x))$.

Proof The proof is simple due to the proof of Lemma 4.3 and the definition of $\pi(u_i, R_{uq})$, and thus, it is omitted. □

Consider the MPRQU(uq, 2, 0.5) example depicted in Fig. 5 where $uq = \{\langle q_1, 0.6\rangle, \langle q_2, 0.4\rangle\}$. We have $u\pi(u_1, R_{uq}) = Pr(u_{11}) \times Pr(q_1) + Pr(u_{12}) \times Pr(q_1) + Pr(u_{12}) \times Pr(q_2) = 0.76$ and $u\pi(u_3, R_{uq}) = Pr(u_{31}) \times Pr(q_1) + Pr(u_{31}) \times Pr(q_2) = 0.4$, since $\phi(u_{11})$, $\phi(u_{12})$, and $\phi(u_{31})$ are inside $\phi(R_{q_1})$ while $\phi(u_{12})$ and $\phi(u_{31})$ are inside

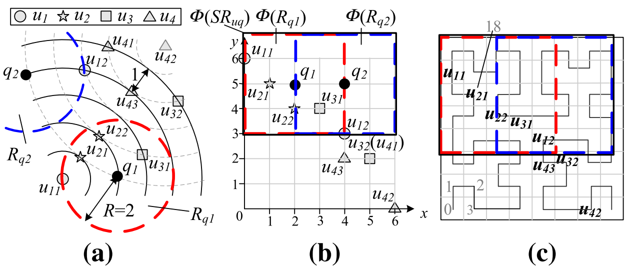

Fig. 5 Example of MPRQU using the object-level model. **a** Metric space, **b** vector space, **c** Hilbert curve

$\phi(R_{q_2})$. Hence, u_3 can be pruned based on Theorem 4.1. Also, $u\pi(u_i, R_{uq})$ can be tightened during verification in order to get stronger pruning power. To be more specific, if $\phi(u_{ik}) \in \phi(R_{q_y})$ but $d(q_y, u_{ik}) > R$, then $u\pi(u_i, R_{uq})$ can be tightened to $\sum_{x=1}^{m} \sum_{\phi(u_{ij}) \in \phi(R_{q_x})} (Pr(u_{ij}) \times Pr(q_x)) - Pr(u_{ik}) \times Pr(q_y)$.

Lemma 4.6 *Given an MPRQU(uq, R, θ) and a pivot set P,* $l\pi(u_i, R_{uq}) = \sum_{x=1}^{m} \sum_{\exists p_t \in P \text{ s.t. } d(u_{ij}, p_t) \le R - d(q_x, p_t)} (Pr(u_{ij}) \times Pr(q_x))$.

Proof The proof is simple due to the proof of Lemma 4.4 and the definition of $\pi(u_i, R_{uq})$, and thus, it is omitted. □

Consider the MPRQ$(uq, 2, 0.5)$ example in Fig. 5, where $uq = \{\langle q_1, 0.6\rangle, \langle q_2, 0.4\rangle\}$. We have $l\pi(u_1, R_{uq}) = Pr(u_{11}) \times Pr(q_1) = 0.3$, since pivot u_{11} satisfies $d(u_{11}, u_{11}) \le 2 - d(q, u_{11})$. Further, $l\pi(u_i, R_{uq})$ can be tightened during verification to improve the pruning power. Specifically, if u_{ik} does not satisfy the condition of Lemma 4.2 (i.e., $d(u_{ik}, p_t) > R - d(q_y, p_t)$) but $u_{ik} \in R_{q_y}$, then $l\pi(u_i, R_{uq})$ can be tighten to $\sum_{\exists p_t \in P \text{ s.t. } d(u_{ij}, p_t) \le R - d(q_x, p_t)} (Pr(u_{ij}) \times Pr(q_x)) + Pr(u_{ik}) \times Pr(q_y)$.

To reduce the computational costs of upper and lower bounds, a single range region $\phi(SR_{uq})$ that contains all subrange regions R_{q_x} $(1 \le x \le m)$ can be used to avoid unnecessary verifications. Consider the example illustrated in Fig. 5b, where $\phi(SR_{uq})$ denotes the minimum bounding box to contain $\phi(R_{q_1})$ and $\phi(R_{q_2})$.

Lemma 4.7 *Let* $\phi(SR_{uq})$ *be the minimum bounding box to contain all* $\phi(R_{qx})$ $(q_x \in uq)$*, if an instance object* u_{ij} *stratifies that* $\phi(u_{ij}) \notin \phi(SR_{uq})$*, then* u_{ij} *can be pruned safely for the uncertain query object uq.*

Proof According to the definition, we can get that $\phi(R_{qx}) \subset \phi(SR_{uq})$ for any $q_x \in uq$. Hence, if $\phi(u_{ij}) \notin \phi(SR_{uq})$, then $\phi(u_{ij}) \notin \phi(R_{qx})$ for any $q_x \in uq$. Due to Lemma 4.1, u_{ij} cannot locate in any R_{qx}, and thus, u_{ij} can be pruned safely for the uncertain query object uq. □

Consider the MPRQU$(uq, 2, 0.5)$ example depicted in Fig. 5 where $uq = \{\langle q_1, 0.6\rangle, \langle q_2, 0.4\rangle\}$. Instance objects u_{41},u_{42}, u_{43}, and u_{32} can be pruned using Lemma 4.7. According to Lemma 4.7, if the MBB of a node N does not overlap $\phi(SR_{uq})$, we can also prune N. However, we still have to verify instance objects that cannot be pruned by Lemma 4.7. Therefore, we develop Lemma 4.8 to further avoid unnecessary verifications.

Lemma 4.8 *Given a pivot set P, for an instance object* u_{ij}*, if there exists a pivot* p_t $(\in P)$ *satisfies that* $d(u_{ij}, p_t) \le R - max\{d(q_x, p_t) | q_x \in uq\}$*, then* u_{ij} *can be validated for the uncertain query object uq.*

Proof According to Lemma 4.2, if there exists a pivot p_t $(\in P)$ satisfies that $d(u_{ij}, p_t) \le R - max\{d(q_x, p_t) | q_x \in uq\}$, then u_{ij} locates in all R_{qx} $(q_x \in uq)$. Hence, u_{ij} can be validated for the uncertain query object uq. □

Consider again the MPRQU$(uq, 2, 0.5)$ example shown in Fig. 5 where $uq = \{\langle q_1, 0.6\rangle, \langle q_2, 0.4\rangle\}$. Due to Lemma 4.8, none of those instance objects can be validated for uq.

Algorithm 2 UPB-tree based MPRQU Algorithm (UtMUA)

```
Input: $uq$, $R$, $\theta$, an uncertain object set $U$ indexed by a UPB-tree
Output: the result set $S_r$ of $MPRQU(uq, R, \theta)$
/* $S_c$: a set of candidate uncertain objects; $IM_c$: a map to store candidate
   instance and vector pairs $\langle u_{ij}, v\rangle$, as vector $v$ stores qualified query
   instances $q_x \in uq$ with $\phi(u_{ij}) \in \phi(R_{qx})$. */
1: compute $\phi(q_x)$ and $\phi(R_{qx})$ for each $q_x \in uq$ using a pivot table $P$
2: compute the minimum bounding box $\phi(SR_{uq})$ to contain $\phi(R_{qx})$ $(q_x \in uq)$
3: push the root node of B$^+$-tree into a heap $H$
4: while $H \neq \emptyset$ do
5:   de-heap the top node $N$ from $H$
6:   if $N$ is a non-leaf node then
7:     for each entry $e$ in $N$ do
8:       if $MBB(e.ptr) \cap \phi(SR_{uq}) \neq \emptyset$ then   // Lemma 4.7
9:         for each query instance $q_x (1 \le x \le m)$ of $uq$ do
10:          if $MBB(e.ptr) \cap \phi(R_{qx}) \neq \emptyset$ then
11:            push $e.ptr$ into $H$ and break
12:  else   // $N$ is a leaf node
13:    for each entry $e$ in $N$ do
14:      if $MBB(e.ptr) \notin \phi(SR_{uq})$ then   // Lemma 4.7
15:        if Lemma 4.8 is satisfied then
16:          if $e.id \notin S_c$ then insert $e.id$ into $S_c$
17:          update $l\pi(e.id, R_{uq})$
18:        else
19:          for each query instance $q_x (1 \le x \le m)$ of $uq$ do
20:            if $\phi(e.ptr) \in \phi(R_{qx})$ then   // Lemma 4.1
21:              if $e.id \notin S_c$ then insert $e.id$ into $S_c$
22:              update $u\pi(e.id, R_{uq})$   // Lemma 4.5
23:              if Lemma 4.2 is satisfied then
24:                update $l\pi(e.id, R_{uq})$   // Lemma 4.6
25:              else if $e.ptr \notin IM_c$ then insert $\langle e.ptr, v = \{q_x\}\rangle$ into $IM_c$
26:              else push $q_x$ into $IM_c(e.ptr).v$
27: for each $u_i$ in $S_c$ do
28:   if $l\pi(u_i, R_{uq}) \ge \theta$ then   // Theorem 4.1
29:     insert $u_i$ into $S_r$ and remove all $u_{ik}$ from $IM_c$
30:   else if $u\pi(u_i, R_{uq}) < \theta$ then remove all $u_{ik}$ from $IM_c$   // Theorem 4.1
31: for each $u_{ij}$ in $IM_c$ do
32:   for each $q_x$ in $IM_c(u_{ij}).v$ do
33:     if $d(q_x, u_{ij}) \le R$ then update $l\pi(u_i, R_{uq})$
34:     else update $u\pi(u_i, R_{uq})$
35:     if $l\pi(u_i, R_{uq}) \ge \theta$ then
36:       insert $u_i$ into $S_r$, remove all $u_{ik}$ from $IM_c$, and break
37:     else if $u\pi(u_i, R_{uq}) < \theta$ then remove all $u_{ik}$ from $IM_c$ and break
38: return $S_r$
```

Based on Theorem 4.1 and Lemmas 4.5 to 4.8, we develop a *UPB-tree-based MPRQU algorithm* (UtMUA) that follows a filtering-and-refinement framework. The pseudo-code of

UtMUA is depicted in Algorithm 2, which takes as inputs uq, R, θ, and an uncertain object set U indexed by a UPB-tree, and outputs a result set S_r. In the filtering phase, UtMUA first computes $\phi(q_x)$ and $\phi(R_{q_x})$ for each query instance q_x ($1 \leq x \leq m$) of uq, using a pivot table P (line 1), and then, it computes the minimum bounding box $\phi(SR_{uq})$ to contain all $\phi(R_{q_x})$ (line 2). Next, similar as UtMA, it traverses the B$^+$-tree in a depth-first fashion (lines 3–26), to find candidate uncertain objects (maintained in a set S_c) and the uncertain instance objects needed further verification (kept in a map IM_c). The differences between UtMUA and UtMA are that (i) UtMUA visits nonleaf entries e in the B$^+$-tree if $MBB(e.ptr)$ intersects with any $\phi(R_{q_x})$ ($1 \leq x \leq m$) (lines 8–10), (ii) UtMUA prunes and validates the instance objects using Lemmas 4.7 and 4.8 (lines 14–17); (iii) UtMUA updates $u\pi(u_i, R_{uq})$ and $l\pi(u_i, R_{uq})$ using Lemmas 4.5 and 4.6 instead of Lemmas 4.3 and 4.4 (lines 22–24), and (iv) UtMUA utilizes the map IM_c to maintain each candidate instance object u_{ij} with a corresponding vector v that stores qualified query instances (lines 25–26). A query instance q_x is qualified for u_{ij} if the condition of Lemma 4.1 holds, i.e., $\phi(u_{ij}) \in \phi(R_{q_x})$. Then, the verification in UtMUA is also similar as that in UtMA (lines 27–37). The difference is that UtMUA needs to verify all the query instances in $IM_c(u_{ij}).v$ for each u_{ij} in IM_c (line 32). Finally, S_r is returned (line 38).

Example 2 We illustrate UtMUA using the MPRQU(uq, 2, 0.5) shown in Fig. 5 with the UPB-tree in Fig. 4. In the filtering phase, UtMUA computes $\phi(q_1) = (2, 5)$ and $\phi(q_2) = (4, 5)$ using $P = \{u_{11}, u_{42}\}$, and then, it gets $\phi(R_{q_1})$, $\phi(R_{q_2})$ and $\phi(SR_{uq})$. Next, it traverses the B$^+$-tree to obtain candidate uncertain objects ($S_c = \{u_1, u_2, u_3\}$) and instance objects (IM$_c = \{\langle u_{21}, \{q_1\}\rangle, \langle u_{22}, \{q_1, q_2\}\rangle, \langle u_{31}, \{q_1, q_2\}\rangle, \langle u_{12}, \{q_1, q_2\}\rangle\}$). Meanwhile, $l\pi(u_i, R_{uq})$ and u$\pi(u_i, R_{uq})$ ($u_i \in S_c$) are updated using Lemmas 4.5 and 4.6. After that, the algorithm validates or eliminates the uncertain objects in S_c. Specifically, u_3 is pruned as $u\pi(u_3, R_q) = 0.4 < 0.5$, and u$_{31}$ is removed from IM$_c$. Thereafter, the algorithm has S$_c = \{u_1, u_2\}$, IM$_c = \{\langle u_{21}, \{q_1\}\rangle, \langle u_{22}, \{q_1, q_2\}\rangle, \langle u_{12}, \{q_1, q_2\}\rangle\}$, and $S_r = \emptyset$. In the refinement phase, for u_{21} in IM$_c$, $l\pi(u_2, R_q)$ is tightened to 0.3 due to $d(q_1, u_{21}) < 2$. In the sequel, entries in IM$_c$ are verified similarly. Finally, the result set S$_r = \{u_1, u_2\}$ is returned. □

4.4 UPB-tree-based MPRJ algorithm

Metric probabilistic range queries only involve a single query object. In this section, we consider a set of uncertain query objects, and thus, we investigate the metric probabilistic range join (MPRJ) that finds all uncertain object pairs $\langle u_i, v_j\rangle$ with $\pi(u_i, v_j)$ exceeding the probability threshold θ. Consider the example using the object-level model illustrated in Fig. 6, where $U = \{u_1, u_2, u_3\}$ and $V = \{v_1, v_2\}$ with

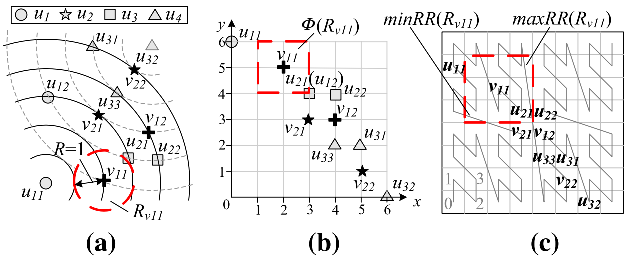

Fig. 6 Example of MPRJ using the object-level model. **a** Metric space, **b** vector space, **c** *Z*-order curve

$u_1 = \{(u_{11}, 0.6), (u_{12}, 0.4)\}$, $u_2 = \{(u_{21}, 0.5), (u_{22}, 0.5)\}$, $u_3 = \{(u_{31}, 0.2), (u_{32}, 0.3), (u_{33}, 0.5)\}$, $v_1 = \{(v_{11}, 0.4), (v_{12}, 0.6)\}$, and $v_2 = \{(v_{21}, 0.5), (v_{22}, 0.5)\}$. Assume that $R = 1$, $\theta = 0.5$, and that the L_2-norm is utilized. The result of MPRJ(U, V, 1, 0.5) is $\{\langle u_2, v_1\rangle, \langle u_3, v_2\rangle\}$.

A metric probabilistic range join MPRJ(U, V, R, θ) can be regarded as an MPRQU(u_i, R, θ) on V for each uncertain object u_i in U, i.e., MPRJ(U, V, R, θ) = $\cup_{u_i \in U} \{\langle u_i, v_j\rangle \mid v_j \in$ MPRQU$(u_i, R, \theta)\} = \cup_{u_i \in U}\{\langle u_i, v_j\rangle \mid \pi(v_j, R_{u_i}) \geq \theta\}$. Also, the metric probabilistic range join is *symmetric* in its two arguments U and V, i.e., MPRJ(U, V, R, θ) = MPRJ(V, U, R, θ). Hence, MPRJ(U, V, R, θ) = $\cup_{v_j \in V}\{\langle u_i, v_j\rangle \mid u_i \in$ MPRQU$(v_j, R, \theta)\} = \cup_{v_j \in V}\{\langle u_i, v_j\rangle \mid \pi(u_i, R_{v_j}) \geq \theta\}$.

A naive solution for the metric probabilistic range join MPRJ(U, V, R, θ) is to perform $|U|$ metric probabilistic range queries MPRQU(u_i, R, θ) ($u_i \in U$) on V or $|V|$ metric probabilistic range queries MPRQU(v_j, R, θ) ($v_j \in V$) on U. However, these are inefficient because they have to access the uncertain object set V or U multiple times. Thus, we propose an efficient algorithm, called *UPB-tree-based metric probabilistic range join algorithm* (UtMJA), which scans the uncertain object sets U and V only once.

As $\pi(u_i, v_j) = \pi(u_i, R_{v_j}) = \pi(v_j, R_{u_i})$ according to the definition of MPRJ, we get that $\pi(u_i, v_j) = \sum_{u_{ix} \in u_i} \sum_{v_{jy} \in v_j \wedge (v_{jy} \in R_{u_{ix}} \vee u_{ix} \in R_{v_{jy}})} (Pr(u_{ix}) \times Pr(v_{jy}))$. Hence, in order to find an MPRJ result pair $\langle u_i, v_j\rangle$ with $\pi(u_i, v_j) \geq \theta$, we need to find the instances $v_{jy} \in R_{u_{ix}}$ or the instances $u_{ix} \in R_{v_{jy}}$. Since MPRJ can be regarded as multiple MPRQU, Lemma 4.1 still holds. Thus, we can use Lemma 4.1 to avoid unnecessary verifications when finding instances $v_{jy} \in R_{u_{ix}}$ and instances $u_{ix} \in R_{v_{jy}}$. Also, the *Z*-order curve can be utilized with the pivot mapping to further accelerate MPRJ processing.

Here, *Z*-order curve rather than other SFC curves is utilized, because it has the property that, given two points $\phi(v_{jy}) = \langle s_1, \ldots, s_{|P|}\rangle$ and $\phi(v'_{jy}) = \langle s'_1, \ldots, s'_{|P|}\rangle$ in the vector space, if $s_t \leq s'_t$ for all $1 \leq t \leq |P|$, then SFC($\phi(v_{jy})$) $\leq$ SFC($\phi(v'_{jy})$). Let minRR($R_{u_{ix}}$) and maxRR($R_{u_{ix}}$) be the SFC values for the left lower and right upper points in $\phi(R_{u_{ix}})$, i.e., minRR($R_{u_{ix}}$) = SFC($\phi(\langle d(u_{ix}, p_1) - R, \ldots, d(u_{ix}, p_{|P|}) - R\rangle)$) and maxRR($R_{u_{ix}}$) = SFC($\phi(\langle d(u_{ix}, p_1)$

$+ R, \ldots, d(u_{ix}, p_{|P|}) + R\rangle)$). In order to find the instances $v_{jy} \in R_{u_{ix}}$, we only need to verify the uncertain instance objects v_{jy} whose SFC($\phi(v_{jy})$) are contained in the range [minRR($R_{u_{ix}}$), maxRR($R_{u_{ix}}$)], as elaborated.

Lemma 4.9 *Assume that the Z-order curve and a pivot set P are used. If an uncertain instance v_{jy} is contained in $R_{u_{ix}}$, then SFC($\phi(v_{jy})$) $\in$ [minRR($R_{u_{ix}}$), maxRR($R_{u_{ix}}$)].*

Proof By definition, $\phi(R_{u_{ix}}) = \{\langle s_1, \ldots, s_{|P|}\rangle \mid d(u_{ix}, p_t) - R \le s_t \le d(u_{ix}, p_t) + R \wedge 1 \le t \le |P|\}$. Hence, for $\phi(v_{jy}) \in \phi(R_{u_{ix}})$, minRR($R_{u_{ix}}$) $\le$ SFC($\phi(v_{jy})$) $\le$ maxRR($R_{u_{ix}}$), according to the property of the *Z*-order curve. Based on Lemma 4.1, if an uncertain object instance $v_{jy} \in R_{u_{ix}}$, then $\phi(v_{jy})$ is certainly contained in $\phi(R_{u_{ix}})$. Thus, minRR($R_{u_{ix}}$) $\le$ SFC($\phi(v_{jy})$) $\le$ minRR($R_{u_{ix}}$). □

Due to the symmetry of MPRJ, we can also get that if an uncertain instance u_{ix} is contained in $R_{v_{jy}}$, then SFC($\phi(u_{ix})$) $\in$ [minRR($R_{v_{jy}}$), maxRR($R_{v_{jy}}$)]. In the example shown in Fig. 6, minRR($R_{v_{11}}$) and maxRR($R_{v_{11}}$) are equal to 18 and 30, respectively. According to Lemma 4.9, instance u_{22} can be pruned for v_{11} since SFC($\phi(u_{22})$) > maxRR($R_{v_{11}}$). Based on Lemma 4.9, we can stop evaluating instances v_{jy} for u_{ix}, when SFC($\phi(v_{jy})$) exceeds maxRR($R_{u_{ix}}$) if the instances v_{jy} are retrieved in ascending order of their SFC values, or when SFC($\phi(v_{jy})$) is smaller than minRR($R_{u_{ix}}$) if the instances v_{jy} are retrieved in descending order.

UPB-trees are assumed on the two uncertain object sets U and V, the leaf levels of which contain the uncertain object instances in ascending order of their *Z*-order values. A merge join is then performed on the leaf levels. Specifically, objects u_{ix} and v_{iy} stored in the two leaf levels of two UPB-trees are visited in ascending order of their *Z*-order values, and two lists are used to keep the instances u_{ix} or v_{jy} visited, respectively. When an instance u_{ix} is visited, UtMJA finds uncertain object instances v_{jy} ($\in R_{u_{ix}}$) among instances v_{jy} visited before u_{ix}, and updates $\pi(u_i, v_j)$. Likewise, when an instance v_{iy} is visited, UtMJA finds uncertain object instances u_{ix} ($\in R_{v_{jy}}$) among instances u_{ix} visited before v_{jy} and updates $\pi(u_i, v_j)$. Note that, Lemma 4.9 is used to eliminate unqualified u_{ix} and v_{jy} from the lists, and that Lemma 4.1 is utilized to avoid unnecessary verifications.

During the merge join, we can preserve the current occurrence probability and the failure probability for the uncertain object u_i or v_j, in order to avoid unnecessary verifications for uncertain object pairs. Let $Pr_c(u_i)$ (or $Pr_c(v_j)$) be the sum of the probabilities $Pr(u_{ix})$ (or $Pr(v_{jy})$) already visited, and $Pr_f(u_i)$ (or $Pr_f(v_j)$) be the sum of the probabilities $Pr(u_{ix})$ (or $Pr(v_{jy})$) already deleted from the list due to Lemma 4.9.

Lemma 4.10 *Let $\pi_c(u_i, v_j)$ be the current $\pi(u_i, v_j)$ during MPRJ processing. If $Pr_c(u_i) \times Pr_c(v_j) - \pi_c(u_i, v_j) + (1 - Pr_c(u_i)) \times Pr_f(v_j) + (1 - Pr_c(v_j)) \times Pr_f(u_i) > 1 - \theta$, then $\langle u_i, v_j\rangle$ can be pruned safely.*

Proof By definition, if $\langle u_i, v_j\rangle \in$ MPRJ(U, V, R, θ), then $\pi(u_i, v_j) \ge \theta$. Based on Lemma 4.9, $1 - \pi(u_i, v_j) \ge Pr_c(u_i) \times Pr_c(v_j) - \pi_c(u_i, v_j) + (1 - Pr_c(u_i)) \times Pr_f(v_j) + (1 - Pr_c(v_j)) \times Pr_f(u_i)$. Thus, if $Pr_c(u_i) \times Pr_c(v_j) - \pi_c(u_i, v_j) + (1 - Pr_c(u_i)) \times Pr_f(v_j) + (1 - Pr_c(v_j)) \times Pr_f(u_i) > 1 - \theta$, then $\pi(u_i, v_j) < \theta$. Hence, $\langle u_i, v_j\rangle$ can be pruned away without any further verification. □

Algorithm 3 UPB-tree based MPRJ Algorithm (UtMJA)

```
Input: UPB-trees $UT_U$ and $UT_V$ to index $U$ and $V$, $R$, $\theta$
Output: the result set $S_r$ of $MPRJ(U, V, R, \theta)$
1: $L_U \leftarrow \emptyset$, $L_V \leftarrow \emptyset$
2: traverse $UT_U$ and $UT_V$ to get the first leaf entries $E_U$ and $E_V$
3: while $E_U \ne \emptyset$ or $E_V \ne \emptyset$ do
4:   if $E_V = \emptyset$ or $E_U.key < E_V.key$ then   // all the entries in $UT_V$ are visited
                                    // or the SFC value of the current $E_U$ is smaller than that of $E_V$
5:     Verify($u_{ix}$, $L_V$)  // $u_{ix} = E_U.ptr$
6:     insert $u_{ix}$ into $L_U$ and update $Pr_c(u_i)$
7:     $E_U \leftarrow E_U.get\_next()$  // get the next leaf entry $E_U$ in $UT_U$
8:   else  // $E_U = \emptyset$ or $E_U.key \ge E_V.key$
9:     Verify($v_{jy}$, $L_U$)  // $v_{jy} = E_V.ptr$
10:    insert $v_{jy}$ into $L_V$ and update $Pr_c(v_j)$
11:    $E_V \leftarrow E_V.get\_next()$  // get the next leaf entry $E_V$ in $UT_V$
12: return $S_r$

Function: Verify($u_{ix}$, $L$)
13: $v_{jy} \leftarrow L.get\_last()$  // get the last entry in $L$
14: while $v_{jy} \ne \emptyset$ do
15:   if $\langle u_i, v_j\rangle \notin S_r$
16:     if $maxRR(R_{u_{ix}}) < SFC(\phi(v_{jy}))$ then  // Lemma 4.9
17:       delete $v_{jy}$ from $L$ and update $Pr_f(v_j)$
18:     else if $Pr_c(u_i) \times Pr_c(v_j) - \pi_c(u_i, v_j) + Pr_f(u_i) \times (1 - Pr_c(v_j)) + Pr_f(v_j) \times$
        $(1 - Pr_c(v_j)) \le 1 - \theta$ then  // Lemma 4.10
19:       if $\phi(v_{jy}) \in \phi(R_{u_{ix}})$ then  // Lemma 4.1
20:         if $d(u_{ix}, v_{jy}) \le R$ then
21:           update $\pi_c(u_i, v_j)$
22:           if $\pi_c(u_i, v_j) \ge \theta$ then
23:             insert $\langle u_i, v_j\rangle$ into $S_r$
24:   $v_{jy} \leftarrow v_{jy}.get\_pre()$  // get the previous entry in $L$
```

The pseudo-code of UtMJA is presented in Algorithm 3. First, UtMJA initializes two lists L_U and L_V to empty and gets the first leaf entries E_U and E_V of the UPB-trees UT_U and UT_V, respectively (lines 1–2). Then, the algorithm performs a *while loop* to visit the leaf entries in ascending order of SFC values (i.e., *key*s stored in the UPB-trees), until all leaf entries of UT_U and UT_V have been evaluated (i.e., E_U and E_V are empty) (lines 3–11). Each time, if all the leaf entries of UT_V are visited (i.e., $E_V = \emptyset$) or $E_U.key \le E_V.key$, the *Verify* function is invoked (line 5) to find the instances v_{iy} stored in L_V with $v_{iy} \in R_{u_{ix}}$ ($u_{ix} = E_U.ptr$), in order to update $\pi_c(u_i, v_j)$ and S_r if necessary (lines 18–23). The *Verify* function also deletes unqualified v_{jx} from L_V and updates corresponding $Pr_f(v_j)$ (lines 16–17). After that, UtMJA inserts u_{ix} into the list L_U, updates $Pr_c(u_i)$, and gets the next leaf entry E_U in UT_U (lines 6–7). Otherwise, if E_U is empty (i.e., all leaf entries in UT_U have been visited) or $E_U.key > E_V.key$, the algorithm invokes *Verify* (line 9) to find the instances u_{ix} stored in L_U satisfying $u_{ix} \in R_{v_{jy}}$ ($v_{jy} = E_V.ptr$) with $\pi_c(u_i, v_j)$ and S_r being updated if necessary, and to prune unqualified u_{ix} in L_U with the corresponding $Pr_f(u_i)$ being updated. Next, UtMJA inserts v_{jy} in the list L_V, updates $Pr_c(v_j)$, and gets the next leaf entry

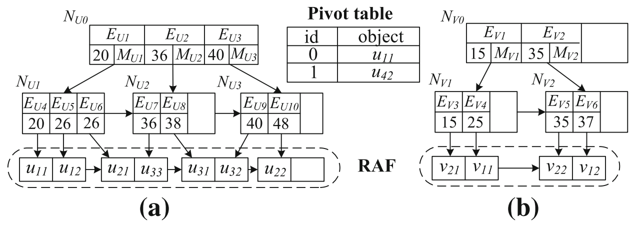

Fig. 7 UPB-trees built on U and V. **a** UT_U and **b** UT_V

E_V in UT_V (lines 10–11). Finally, the query result set S_r is returned (line 12).

Example 3 We illustrate UtMJA using MPRQ(U, V, 1, 0.5) as depicted in Fig. 6, with the corresponding UPB-trees shown in Fig. 7. We assume that $R = 1$ and $\theta = 0.5$. Initially, UtMJA sets L_U and L_V to empty, and traverses UT_U and UT_V to get the first leaf entries E_{U4} and E_{V3}. Then, it performs a while loop. In the first iteration, since $E_{U4}.key > E_{V3}.key$, it invokes Verify(v_{21}, L_U), inserts v_{21} (i.e., $E_{V3}.ptr$) into L_V ($= \{v_{21}\}$), updates $Pr_c(v_2)$ to 0.5, and gets the next leaf entry E_{V4} in UT_V. In the second iteration, as $E_{U4}.key > E_{V4}.key$, Verify(u_{11}, L_V) is invoked, where v_{21} can not be pruned due to Lemmas 4.1, 4.9, and 4.10, and $\pi_c(u_2, v_1)$ is updated to 0.25. Next, UtMJA inserts u_{21} into L_U ($= \{u_{12}\}$), updates $Pr_c(u_2)$ to 0.5, and gets the next leaf entry E_{U5} in UT_U. In the third iteration, as $E_{U5}.key < E_{V4}.key$, UtMJA calls Verify(u_{12}, L_V), where v_{21} is pruned for u_{12} as $d(u_{12}, v_{21}) > R$. Then, it inserts u_{21} into L_U ($= \{u_{12}, u_{21}\}$) and gets the next leaf entry E_{U6} in UT_U. UtMJA proceeds in the same manner until all the leaf entries in UT_U and UT_V are visited. Finally, it returns the result set $S_r = \{\langle u_2, v_1\rangle, \langle u_3, v_2\rangle\}$. □

4.5 Discussion

We proceed to discuss the SFC mapping, the impact of LRU buffer, and then present CPU and I/O cost analysis.

4.5.1 SFC mapping

During the SFC mapping, if the range of $d(\ )$ in an original metric space is *discrete* integers (e.g., edit distance), SFC can directly map $\phi(u_{ij})$ to an integer SFC($\phi(u_{ij})$). On the other hand, if the range of $d(\ )$ in a metric space covers a range of *continuous* real numbers (e.g., the L_∞*-norm*), the real numeric range can be partitioned into n slots, namely slot 0, slot 1, …, slot $n-1$ with corresponding intervals [0, s), [s, $2\times s$), …, [$(n-1)\times s$, d^+), in which $s = d^+/n$. Thus, the entire mapped vector space can be divided into cells. Then, $\phi(u_{ij})$ can be approximated as $\langle \lfloor d(u_{ij}, p_1)/s\rfloor, \lfloor d(u_{ij}, p_2)/s\rfloor, \ldots, \lfloor d(u_{ij}, p_{|P|})/s\rfloor\rangle$.

4.5.2 The impact of LRU buffer

If the LRU buffer size is much smaller than the index size, the I/O cost of our proposed MPRQ, MPRQU, and MPRJ algorithms based on UPB-trees is unaffected. This is because, UPB-trees are visited only once for every query. However, for a large LRU buffer (i.e., the whole tree can be load into the buffer in an extreme case), the I/O cost can be reduced. They also hold for the presented UPB-forest as follows.

4.5.3 CPU cost

The cost of distance computations is the dominant component of the CPU cost for search in the metric space, since distance computations are usually expensive (e.g., edit distance, Jaccard coefficient). Hence, the CPU cost can be estimated by the number of distance computations.

For UtMA, the number of distance computations includes the number of distance computations for computing $\phi(q)$ and that for verifying whether an instance object u_{ij} is contained in R_q. In the worst case, UtMA needs to verify all instance objects u_{ij} with $\phi(u_{ij}) \in \phi(R_q)$ according to Lemma 4.1. Thus, the CPU cost of UtMA in terms of distance computations can be calculated as:

$$\text{UtMA}_C = |P| + \sum_{u_{ij}\in U} I(\phi(u_{ij}), \phi(R_q)) \qquad (1)$$

where $I(a, b) = \begin{cases} 1 & a \in b \\ 0 & \text{otherwise} \end{cases}$.

Since UtMUA is similar to UtMA, with the query object q having replaced with the uncertain query object uq, the CPU cost of UtMUA in terms of distance computations can be calculated as:

$$\text{UtMUA}_C = |P| \times m + \sum_{q_x\in uq}\sum_{u_{ij}\in U} I(\phi(u_{ij}), \phi(R_{q_x})) \qquad (2)$$

For UtMJA, the number of distance computations can be estimated as the sum of distance computations needed when finding the instances v_{jy} in $R_{u_{ix}}$ for each instance u_{ix} using Lemmas 4.1 and 4.9. Therefore, the CPU cost of UtMJA in terms of distance computations can be calculated as:

$$\text{UtMJA}_C = \sum_{u_{ix}\in U}\sum_{v_{jy}\in V} I(\phi(v_{jy}), \phi(R_{u_{ix}})) \qquad (3)$$

4.5.4 I/O cost

The I/O cost of the query processing on a UPB-tree includes two parts, i.e., B$^+$-tree and RAF page accesses.

For UtMA, to obtain the number of B$^+$-tree page accesses, it is sufficient to sum all the nodes whose MBBs are inter-

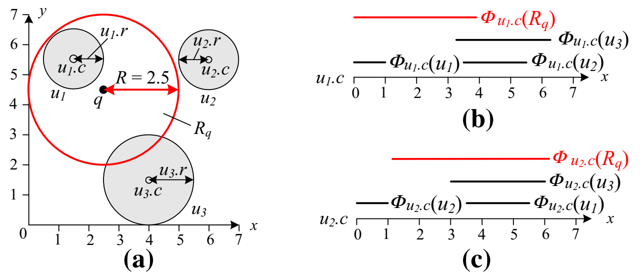

Fig. 8 Example of MPRQ using the UPB-forest. (a) Metric space, **b** the vector space after $u_1.c$ mapping, **c** the vector space after $u_2.c$ mapping

sected with the search region $\phi(R_q)$. Also, the number of RAF page accesses can be estimated as UtMA_C/f, because the instance objects accessed in the RAF are expected to be stored close to each other. Here, UtMA_C is used to estimate the total number of the instance objects visited, and f represents the average number of instance objects per RAF page. Hence, the I/O cost of UtMA in terms of page accesses can be calculated as:

$$\text{UtMA}_{IO} = \sum_{M_i \in B^+\text{-tree}} I'(M_i, \phi(R_q)) + \frac{\text{UtMA}_C}{f} \tag{4}$$

where $I'(a, b) = \begin{cases} 1 & a \text{ intersects with } b \\ 0 & \text{otherwise} \end{cases}$.

For UtMUA, similarly, the I/O cost of UtMUA in terms of page accesses can be calculated as:

$$\text{UtMUA}_{\text{IO}} = \sum_{M_i \in B^+\text{-tree}} I'(M_i, \phi(R_{uq})) + \frac{\text{UtMUA}_C}{f} \tag{5}$$

UtMJA traverses UPB-trees built on uncertain object sets U and V only once, and thus, it is sufficient to sum the number of B$^+$-tree leaf pages and RAF pages in the UPB-trees. Let $|UT_U|$ ($|UT_V|$) denote the total number of B$^+$-tree leaf pages of UT_U (UT_V), and let f_U (f_V) represent the average number of the objects per RAF page for UT_U (UT_V). The I/O cost of UtMJA in terms of page accesses can then be calculated as:

$$\text{UtMJA}_{IO} = |UT_U| + |UT_V| + \frac{|U| \times m}{f_U} + \frac{|V| \times m}{f_V} \tag{6}$$

5 The UPB-forest

We first present a UPB-forest structure based on the bi-level model (only considering the continuous case as stated in Sect. 3.2), and then, we provide corresponding MPRQ and MPRJ algorithms. Finally, we offer cost analysis.

5.1 UPB-forest structure

As does the UPB-tree, the UPB-forest maps a metric space into multiple one-dimensional vector spaces using a pivot set P, and then, it utilizes B$^+$-trees to index those mapped one-dimensional vector spaces. However, unlike the UPB-tree that only uses a single B$^+$-tree, the UPB-forest utilizes $|P|$ B$^+$-trees to index intervals (for continuous PDFs) or points (for discrete PDFs) in the mapped one-dimensional vector space. This is because, as discussed in Sect. 2.2, a metric uncertain object can be represented as $|P|$ PDFs w.r.t. each pivot in P using the bi-level model, but it is difficult to derive the multivariate PDF w.r.t the whole pivot set P. Therefore, one B$^+$-tree is needed for every $p \in P$.

Given a pivot set $P = \{p_1, p_2, \ldots, p_{|P|}\}$, a metric space (M, d) can be mapped into $|P|$ one-dimensional vector spaces (R, L_∞). Specifically, an uncertain object u_i (denoted as an uncertain region centered at $u_i.c$ with radius $u_i.r$) in the metric space, it can be represented as $|P|$ intervals $\phi_{p_t}(u_i) = [d(u_i.c, p_t) - u_i.r, d(u_i.c, p_t) + u_i.r]$ ($p_t \in P$) with associated continuous PDFs in the one-dimensional vector space. As an example, Fig. 8b, c illustrates the mapped one-dimensional vector spaces using pivots $u_1.c$ and $u_2.c$, respectively. According to the triangle inequality, given a query object q and a pivot p_t, $d(q, u_i) \geq |d(q, p_t) - d(u_i, p_t)| = D(\phi_{p_t}(q), \phi_{p_t}(u_i))$. Hence, we can conclude that the distance in the mapped vector space is a *lower bound* of that in the original metric space.

In order to index the intervals with continuous PDFs after the pivot mapping, the real numeric range is partitioned into n slots, i.e., slot 0, slot 1, …, slot $n-1$, with corresponding intervals $[0, s)$, $[s, 2 \times s)$, …, $[(n-1) \times s, d^+)$ where $s = d^+/n$. Thus, $\phi_{p_t}(u_i)$ can be partitioned into several subintervals, and every continuous $\mathcal{G}_{p_t,i}$ ($p_t \in P$) can be approximated as discrete g_{p_t,u_i}, in which $g_{p_t,u_i}(j)$ denotes the probability integration on the slot j (i.e., the interval $[j \times s, (j+1) \times s)$). Take Fig. 8 as an example. Suppose $n = 10$ and $s = 1$. Then, $\phi_{u_1.c}(u_3)$ can be partitioned into four slots (i.e., slots 3, 4, 5, and 6), and hence, $\mathcal{G}_{u_1.c,u_3}$ can be approximated as $g_{u_1.c,u_3}$, with $g_{u_1.c,u_3}(3) = \int_3^4 \mathcal{G}_{u_1.c,u_3}(x)\mathrm{d}x = 0.14$, $g_{u_1.c,u_3}(4) = \int_4^5 \mathcal{G}_{u_1.c,u_3}(x)\mathrm{d}x = 0.4$, $g_{u_1.c,u_3}(5) = 0.38$, and $g_{u_1.c,u_3}(6) = 0.08$.

An UPB-forest used to index a metric uncertain object set based on the bi-level model contains three parts, viz., a pivot table, a RAF, and $|P|$ B$^+$-trees. Figure 9 shows a UPB-forest that indexes the uncertain object set $U = \{u_1, u_2, u_3\}$ depicted in Fig. 8. The pivot table stores selected pivots P ($u_1.c$ and $u_2.c$ in this case) used to map a metric space into $|P|$ one-dimensional vector spaces. The RAF is utilized to keep uncertain objects in ascending order of their identifiers, which improves the efficiency of managing complex uncertain objects. Each RAF entry records an uncertain

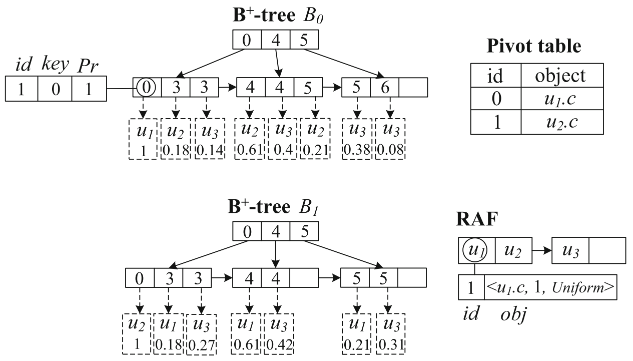

Fig. 9 Example of UPB-forest for the dataset in Fig. 8

object identifier *id* and the real uncertain object *obj*. Continuing the above example, in Fig. 9, the RAF entry associated with uncertain object u_1 records its uncertain object identifier 1 and its real uncertain object $\langle u_1.c, 1, Uniform\rangle$.

The B$^+$-trees B_t $(0 \le t < |P|)$ for each pivot p_t in the pivot table are employed to index the intervals $\phi_{p_t}(u_i)$ after the pivot mapping. Every leaf entry in the leaf node of B_t (0 $\le t < |P|$) records an uncertain object identifier *id*, the slot number *key* $(0 \le key < n)$, and the probability integration *Pr* of $\mathcal{G}_{p_t,u_{id}}$ on the corresponding slot (i.e., $g_{p_t,u_{id}}(key)$). As an example, in Fig. 9, the first leaf entry of B_0 records the uncertain object identifier $id = 1$, the slot number $key =$ 0, and the probability integration $Pr = 1$ equivalent to the numeric integration of $\mathcal{G}_{u_1.c,u_1}$ on slot 0.

Note that a UPB-forest works iff the continuous PDF representing the uncertain objects can be integrated. In addition, unlike the UP-Index [3], which needs to store the pre-computed histograms w.r.t. the *whole* distance range for every uncertain object u_i, the UPB-forest only stores *necessary* information. To be more specific, for each u_i, the UPB-forest only records nonzero values $g_{p_t,u_i}(j)$ on slot j instead of the entire slot set. In addition, B$^+$-trees of the UPB-forest preserve entries in ascending order of their corresponding slot values, which improves I/O efficiency as the entries to be visited are stored close to each other.

5.2 UPB-forest-based MPRQ algorithm

In what follows, we employ a running example to better explain MPRQ processing. Specifically, consider the uncertain object set $U = \{u_1, u_2, u_3\}$ illustrated in Fig. 8, with $u_1 = \langle u_1.c, 1, Uniform\rangle$ (indicating that u_1 is located uniformly within the uncertain region centered at $u_1.c$ having radius 1), $u_2 = \langle u_2.c, 1, Uniform\rangle$, and $u_3 = \langle u_3.c, 1.5, Uniform\rangle$. Assuming that the L_2-*norm* is used to measure similarity, the result of a MPRQ$(q, 2.5, 0.15)$ is the uncertain object u_1.

To accelerate MPRQ processing, we first derive an upper bound $u\pi(u_i, R_q)$ and a lower bound $l\pi(u_i, R_q)$ on $\pi(u_i, R_q)$ based on the UPB-forest. These allow us to filter out and validate uncertain objects according to Theorem 4.1 without any need for further verification.

Lemma 5.1 *Given a pivot set P and a query object q, for an uncertain object u_i, $u\pi(u_i, R_q) = min_{p_t \in P} \int_{d(q,p_t)-R}^{d(q,p_t)+R} \mathcal{G}_{p_t,u_i}(x)\mathrm{d}x$ is an upper bound on $\pi(u_i, R_q)$.*

Proof Based on the triangle inequality, given *a* pivot p_t, we have $d(q, p_t) - R \le d(u_i, p_t) \le d(q, p_t) + R$ if $d(q, u_i) \le R$, while the reverse is not true. Hence, $\pi(u_i, R_q) = Pr(d(q, u_i) \le R) \le Pr(d(q, p_t) - R \le d(u_i, p_t) \le d(q, p_t) + R) = \int_{d(q,p_t)-R}^{d(q,p_t)+R} \mathcal{G}_{p_t,u_i}(x)\mathrm{d}x$, and $u\pi(u_i, R_q)$ can be set as $\min_{p_t \in P} \int_{d(q,p_t)-R}^{d(q,p_t)+R} \mathcal{G}_{p_t,u_i}(x)\mathrm{d}x$. The proof completes. □

Consider MPRQ$(q, 2.5, 0.15)$ depicted in Fig. 8 as an example. Given $P = \{u_1.c, u_2.c\}, u\pi(u_3, R_q) = \int_{d(q,u_1.c)-R}^{d(q,u_1.c)+R} \mathcal{G}_{u_1.c,u_3}(x)\mathrm{d}x \approx 0.14 < \theta$, and thus, u_3 can be pruned.

As mentioned in Sect. 5.1, $\mathcal{G}_{p_t,u_i}$ is approximated as g_{p_t,u_i} when building the UPB-forest. To utilize Lemma 5.1, $u\pi(u_i, R_q)$ can be calculated as $\min_{p_t \in P} \sum_{\lfloor (d(q,p_t)-R)/s \rfloor}^{\lceil (d(q,p_t)+R)/s \rceil -1} g_{p_t,u_i}(x)$. Considering again Fig. 8 as an example, $u\pi(u_3, R_q) = g_{u_1.c,u_3}(3) = 0.14$.

Note that, according to Lemma 5.1, if $\phi_{p_t}(u)$ does not intersect with $\phi_{p_t}(R_q)$ $(p_t \in P)$, u_i can be safely pruned by Theorem 4.1 as $u\pi(u_i, R_q) = 0 < \theta$. Thus, we only need to verify uncertain objects u_i with $\phi_{p_t}(u_i)$ crossing $\phi_{p_t}(R_q)$ for every $p_t \in P$, i.e., $\phi(u_i) \cap \phi(R_q) \ne \emptyset$, since $\phi(u_i)$ and $\phi(R_q)$ represent the hypercubes in the mapped multi-dimensional vector space using the whole pivot set P.

Lemma 5.2 *Given a pivot set P and a query object q, for an uncertain object u_i, $l\pi(u_i, R_q) = max_{p_t \in P} \int_0^{R-d(q,p_t)} \mathcal{G}_{p_t,u_i}(x)\mathrm{d}x$ is a lower bound on $\pi(u_i, R_q)$.*

Proof According to the triangle inequality, given *a* pivot p_t, if $d(u_i, p_t) \le R - d(q, p_t)$, then $d(q, u_i) \le R$. Hence, $\pi(u_i, R_q) = Pr(d(q, u_i) \le R) \ge Pr(d(u_i, p_t) \le R - d(q, p_t)) = \int_0^{R-d(q,p_t)} \mathcal{G}_{p_t,u_i}(x)\mathrm{d}x$, and thus, $l\pi(u_i, R_q)$ can be set as $\max_{p_t \in P} \int_0^{R-d(q,p_t)} \mathcal{G}_{p_t,u_i}(x)\mathrm{d}x$. The proof completes. □

Consider again the MPRQ$(q, 2.5, 0.15)$ depicted in Fig. 8. Here, $l\pi(u_1, R_q) = \int_0^{R-d(q,u_1.c)} \mathcal{G}_{u_1.c,u_1}(x)\mathrm{d}x = 1 > \theta$. Therefore, u_1 can be validated without further verification. Similarly, to use Lemma 5.2, $l\pi(u_i, R_q)$ can be calculated as $\max_{p_t \in P} \sum_0^{\lfloor (R-d(q,p_t))/s \rfloor -1} g_{p_t,i}(x)$. For instance, in Fig. 8, $l\pi(u_3, R_q) = g_{u_1.c,u_1}(0) = 1$.

Algorithm 4 UPB-forest based MPRQ Algorithm (UfMA)

```
Input: $q$, $R$, $\theta$, an uncertain object set $U$ indexed by a UPB-forest
Output: the result set $S_r$ of $MPRQ(q, R, \theta)$
1: compute $\phi_{pt}(R_q) \leftarrow [d(q, p_t) - R, d(q, p_t) + R]$, $S_{pt} \leftarrow \{x \mid x \in [\lfloor(d(q, p_t) - R)/s\rfloor, \lceil(d(q, p_t) + R)/s\rceil - 1]\}$, and $S'_{pt} \leftarrow \{x \mid x \in [0, \lfloor(R - d(q, p_t))/s\rfloor - 1]\}$ for every $p_t \in P$   // $s = d^+/n$, and $S_{pt}$ is a set of slots $x$ ($\in \phi_{pt}(R_q)$)
2: for each B$^+$-tree $B_t$ ($0 \le t < |P|$) of the UPB-forest do
3:   for each leaf entry $\langle u_i, key, pr\rangle$ in $B_t$ with the $key \in S_{pt} \cup S'_{pt}$ do
4:     update $u\pi(u_i, R_q)$ and $l\pi(u_i, R_q)$ using Lemmas 5.1 and 5.2
5:     if $u\pi(u_i, R_q) \ge \theta$ and $u_i \notin S_c$ then   // Theorem 4.1
6:       insert $u_i$ into $S_c$
7: for every $u_i \in S_c$ do
8:   if $l\pi(u_i, R_q) \ge \theta$ then   // Theorem 4.1
9:     insert $u_i$ into $S_r$
10:  else if $u\pi(u_i, R_q) < \theta$ then   // Theorem 4.1
11:    prune $u_i$
12:  else if $\pi(u_i, R_q) \ge \theta$ then
13:    insert $u_i$ into $S_r$
14: return $S_r$
```

Based on these upper and lower bounds, we present the *UPB-forest-based MPRQ algorithm* (UfMA), which follows a filtering-and-refinement framework. The pseudo-code of UfMA is shown in Algorithm 4. It takes as inputs q, R, θ, and an uncertain object set U indexed by a UPB-forest, and it outputs the result set S_r of MPRQ(q, R, θ). In the filtering phase, UfMA first computes the set S_{p_t} of slots contained in the mapped search region $\phi_{p_t}(R_q)$ and the set S'_{p_t} for each $p_t \in P$ (line 1). It then traverses each B$^+$-tree B_t of the UPB-forest to retrieve leaf entries $\langle u_i, key, pr\rangle$ with $keys \in S_{p_t}$ or $keys \in S'_{p_t}$ (line 3), updates $u\pi(u_i, R_q)$ or $l\pi(u_i, R_q)$ based on Lemma 5.1 or 5.2 (line 4), and inserts all qualified candidates u_i (i.e., $u\pi(u_i, R_q) \ge \theta$ and $u_i \notin S_c$) into S_c (lines 5–6). In the refinement phase, for every $u_i \in S_c$, the algorithm adds u_i to S_r if $l\pi(u_i, R_q) \ge \theta$ (lines 8–9) and filters u_i if $u\pi(u_i, R_q) < \theta$ (lines 10–11). Otherwise, it computes $\pi(u_i, R_q)$ using numerical integration and inserts u_i into S_r if $\pi(u_i, R_q) \ge \theta$ (lines 12–13). Note that, $\pi(u_i, R_q)$ is derived by using GNU Scientific Library with the error bound set to 10^{-5}. Finally, the result set S_r is returned (line 14).

Example 4 Please refer to the conference paper [8] for details and thus omitted here. □

5.3 UPB-forest-based MPRQU algorithm

To explain MPRQU processing, we also use a running example. Specifically, consider the uncertain object set $U = \{u_1, u_2, u_3\}$ illustrated in Fig. 10, with $u_1 = \langle u_1.c, 1, \textit{Uniform}\rangle$ (meaning that u_1 is located uniformly within the uncertain region centered at $u_1.c$ having radius 1), $u_2 = \langle u_2.c, 1, \textit{Uniform}\rangle$, and $u_3 = \langle u_3.c, 1.5, \textit{Uniform}\rangle$. Assume that the L_2*-norm* is used to measure similarity and $uq = \langle uq.c, 1, \textit{Uniform}\rangle$, the result of a MPRQU($uq$, 1.5, 0.15) is the uncertain object u_1.

Since the query object q is replaced with the uncertain query object uq (an uncertain region centered at $uq.c$ with the radius $uq.r$ in our running example) for MPRQU, the derivation of the upper bound $u\pi(u_i, R_{uq})$ and lower bound

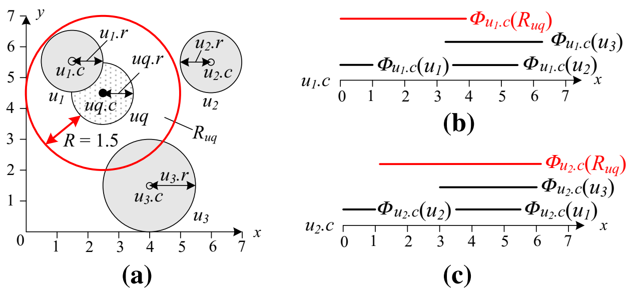

Fig. 10 Example of MPRQU using the UPB-forest. **a** Metric space, **b** the vector space after $u_1.c$ mapping, **c** the vector space after $u_2.c$ mapping

$l\pi(u_i, R_{uq})$ based on the UPB-forest needs to take into account the uncertainty of uq. The range region R_{uq} in the mapped vector space w.r.t. a specific pivot p_t (i.e., $\phi_{p_t}(R_{uq})$) can be represented as an interval $[d(uq.c, p_t) - uq.r - R, d(uq.c, p_t) + uq.r + R]$. Let $F_{p_t,uq}(x) = \int_{x-R}^{x+R} \mathcal{G}_{p_t,uq}(x)\mathrm{d}x$, we can derive the upper bound below.

Lemma 5.3 *Given a pivot set P and a query uncertain object uq, for an uncertain object u_i, $u\pi(u_i, R_{uq}) = min_{p_t \in P} \int_{d(uq.c,p_t)-uq.r-R}^{d(uq.c,p_t)+uq.r+R} \mathcal{G}_{p_t,u_i}(x) \times F_{p_t,uq}(x)\mathrm{d}x$ is a upper bound on $\pi(u_i, R_{uq})$.*

Proof The proof follows straightforwardly from the definition of MPRQU and the proof of Lemma 5.1. □

Since we approximate a continuous $\mathcal{G}_{p_t,u_i}$ ($p_t \in P$) by a discrete g_{p_t,u_i}, in which $g_{p_t,u_i}(j)$ denotes the probability integration on the slot j (i.e., the interval $[j \times s, (j+1) \times s)$), we can also approximate $F_{p_t,uq}$ as a discrete $uf_{p_t,uq}$, where $uf_{p_t,uq}(j)$ represents the maximum probability of $F_{p_t,uq}$ on the slot j. Hence, $u\pi(u_i, R_{uq})$ can be calculated as $\min_{p_t \in P} \sum_{\lfloor(d(uq.c,p_t)-uq.r-R/s)\rfloor}^{\lceil(d(uq.c,p_t)+uq.r+R)/s\rceil-1} (g_{p_t,u_i}(x) \times uf_{p_t,uq}(x))$. For example, in Fig. 10, given $P = \{u_1.c, u_2.c\}$, $u\pi(u_3, R_{uq}) = g_{u_1.c,u_3}(3) \times uf_{u_1.c,uq}(3) = 0.0261 < \theta$. Thus, u_3 can be pruned.

Let $F'_{p_t,uq}(x) = \int_0^{R-x} \mathcal{G}_{p_t,uq}(x)\mathrm{d}x$, we can derive the lower bound as follows.

Lemma 5.4 *Given a pivot set P and a query uncertain object uq, for an uncertain object u_i, $l\pi(u_i, R_{uq}) = max_{p_t \in P} \int_0^{R-d(q,p_t)+uq.r} \mathcal{G}_{p_t,u_i}(x)\mathrm{d}x \times F'_{p_t,uq}(x)dx$ is a lower bound on $\pi(u_i, R_{uq})$.*

Proof The proof follows straightforwardly from the definition of MPRQU and the proof of Lemma 5.2. □

Similarly, we can also approximate $F'_{p_t,uq}$ as the discrete $lf_{p_t,uq}$, in which $lf_{p_t,uq}(j)$ denotes the minimum probability of $F'_{p_t,uq}$ on the slot j. Hence, $l\pi(u_i, R_{uq})$ can be calculated as $\max_{p_t \in P} \sum_0^{\lceil(R-d(uq.c,p_t)+uq.r)/s\rceil-1} (g_{p_t,u_i}(x) \times lf_{p_t,uq}(x))$. Using the example in Fig. 10, given $P =$

$\{u_1.c, u_2.c\}$, $l\pi(u_1, R_{uq}) = g_{u_1.c,u_1}(1) \times lf_{u_1.c,uq}(1) = 0.0086 < \theta$. Thus, u_1 cannot be validated and a further verification is needed.

Algorithm 5 UPB-forest based MPRQU Algorithm (UfMUA)

```
Input: $uq$, $R$, $\theta$, an uncertain object set $U$ indexed by a UPB-forest
Output: the result set $S_r$ of $MPRQU(uq, R, \theta)$
1: compute $uf_{pt,uq}$ on $S_{pt} \leftarrow \{x \mid x \in [\lfloor(d(uq, p_t) - uq.r - R)/s\rfloor, \lceil(d(uq, p_t) + uq.r + R)/s\rceil - 1]\}$   // $s = d^+/n$, and $S_{pt}$ is a set of slots $x$ ($\in \phi_{pt}(R_q)$)
2: compute $lf_{pt,uq}$ on $S'_{pt} \leftarrow \{x \mid x \in [0, \lceil(R - d(uq, p_t) + uq.r)/s\rceil - 1]\}$
3: for each B$^+$-tree $B_t$ ($0 \le t < |P|$) of the UPB-forest do
4:    for each leaf entry $\langle u_i, key, pr\rangle$ in $B_t$ with the $key \in S_{pt}$ do
5:       update $u\pi(u_i, R_{uq})$ and $l\pi(u_i, R_{uq})$ using Lemmas 5.3 and 5.4
6:       if $u\pi(u_i, R_{uq}) \ge \theta$ and $u_i \notin S_c$ then   // Theorem 4.1
7:          insert $u_i$ into $S_c$
8: for every $u_i \in S_c$ do
9:    if $l\pi(u_i, R_{uq}) \ge \theta$ then   // Theorem 4.1
10:      insert $u_i$ into $S_r$
11:   else if $u\pi(u_i, R_{uq}) < \theta$ then   // Theorem 4.1
12:      prune $u_i$
13:   else if $\pi(u_i, R_{uq}) \ge \theta$ then
14:      insert $u_i$ into $S_r$
15: return $S_r$
```

Based on the upper and lower bounds derived above, we present the *UPB-forest-based MPRQU algorithm* (UfMUA), which follows a filtering-and-refinement framework. The pseudo-code of UfMUA is shown in Algorithm 5. It takes as inputs uq, R, θ, and an uncertain object set U indexed by a UPB-forest, and it outputs the result set S_r of MPRQU(uq, R, θ). In the filtering phase, UfMUA first computes $uf_{p_t,uq}$ on the set S_{p_t} of slots contained in the mapped search region $\phi_{p_t}(R_{uq})$ and $lf_{p_t,uq}$ on the set S'_{p_t} for each p_t in P (lines 1–2). It then traverses each B$^+$-tree B_t of the UPB-forest to search leaf entries $\langle u_i, key, pr\rangle$ with $keys \in S_{p_t}$ or $keys \in S'_{p_t}$ (line 4), updates $u\pi(u_i, R_{uq})$ or $l\pi(u_i, R_{uq})$ based on Lemma 5.3 or 5.4 (line 5), and inserts all qualified candidates u_i (i.e., $u\pi(u_i, R_{uq}) \ge \theta$ and $u_i \notin S_c$) into S_c (lines 6–7). In the refinement phase, for every $u_i \in S_c$, the algorithm adds u_i to S_r if $l\pi(u_i, R_{uq}) \ge \theta$ (lines 10–11), and it filters u_i if $u\pi(u_i, R_{uq}) < \theta$ (lines 9–10). Otherwise, it computes $\pi(u_i, R_{uq})$ using the numerical integration and inserts u_i into S_r if $\pi(u_i, R_{uq}) \ge \theta$ (lines 13–14). Finally, the result set S_r is returned (line 15).

Example 5 We illustrate UfMUA using MPRQU(uq, 1.5, 0.15) as depicted in Fig. 10 with the UPB-forest shown in Fig. 9, and we let n = 10 and s = 1. UfMUA computes $uf_{u_1.c,uq}$ on $S_{u_1.c} = \{x \mid x \in [-2, 3]\}$, $uf_{u_2.c,uq}$ on $S_{u_2.c} = \{x \mid x \in [1, 6]\}$, $lf_{u_1.c,uq}$ on $S'_{u_1.c} = \{x \mid x \in [0, 1]\}$, and $lf_{u_2.c,uq}$ on $S'_{u_2.c} = \emptyset$. Then, it traverses the B$^+$-tree B_0 to find the leaf entries with their keys contained in S_{p_t} or S'_{p_t}, i.e., $\langle u_1, 0, 1\rangle$, $\langle u_2, 3, 0.18\rangle$, and $\langle u_3, 3, 0.14\rangle$; it updates $u\pi(u_1, R_{uq}) = 1$, $u\pi(u_2, R_{uq}) = 0.0934$, $u\pi(u_3, R_{uq}) = 0.0261$, and $l\pi(u_1, R_{uq}) = 0.0086$ based on Lemmas 5.3 and 5.4; and it adds u_1 to S_c. Next, B_1 is traversed similarly, after which $u\pi(u_2, R_{uq})$ is updated to 0. In the sequel, UfMUA subsequently checks every uncertain object in S_c. It inserts u_1 into S_r as $\pi(u_1, R_{uq}) > \theta$. Finally, the result set $S_r = \{u_1\}$ is returned. □

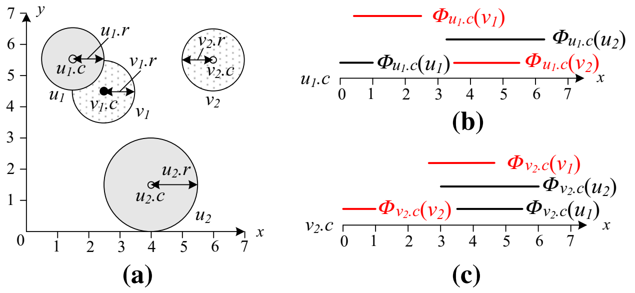

Fig. 11 Example of MPRQU using the UPB-forest. **a** Metric space, **b** the vector space after $u_1.c$ mapping, (c) the vector space after $v_2.c$ mapping

5.4 UPB-forest-based MPRJ algorithm

We also utilize a running example better explain MPRJ processing. Specifically, consider two uncertain object sets $U = \{u_1, u_2\}$ and $V = \{v_1, v_2\}$ illustrated in Fig. 11, with $u_1 = \langle u_1.c, 1, Uniform\rangle$ (meaning that u_1 is located uniformly within the uncertain region centered at $u_1.c$ having radius 1), $u_2 = \langle u_2.c, 1.5, Uniform\rangle$, $v_1 = \langle v_1.c, 1, Uniform\rangle$, and $v_2 = \langle v_2.c, 1, Uniform\rangle$. Assuming that the L_2-*norm* is used to measure similarity, the result of MPRJ(U, V, 1.5, 0.5) is $\{\langle u_1, v_1\rangle\}$.

As discussed in Sect. 4.4, a metric probabilistic range join can be regarded as multiple MPRQU. Hence, a naive solution for MPRJ based on the bi-level model is to perform UfMUA on V for each $u_i \in U$ or to perform UfMUA on U for each $v_j \in V$. However, these are inefficient because they have to scan the uncertain object set V or U multiple times. In view of this, we develop an efficient algorithm, called *UPB-forest-based metric probabilistic range join algorithm* (UfMJA), which scans U and V only once. To accelerate MPRJ, we validate or prune uncertain object pairs $\langle u_i, v_j\rangle$ using upper and lower bounds of $\pi(u_i, v_j)$ without any further verification for computing $\pi(u_i, v_j)$, as stated in Theorem 5.1 below.

Theorem 5.1 *Given an MPRJ(U, V, R, θ), let $u\pi(u_i, v_j)$ and $l\pi(u_i, v_j)$ be the upper bound and lower bound on $\pi(u_i, v_j)$. Then (i) $\langle u_i, v_j\rangle$ can be safely pruned if $u\pi(u_i, v_j) < \theta$, and (ii) $\langle u_i, v_j\rangle$ can be validated if $l\pi(u_i, v_j) \ge \theta$.*

The proof of Theorem 5.1 is straightforward and thus omitted. In order to employ Theorem 5.1, we need to derive bounds $u\pi(u_i, v_j)$ and $l\pi(u_i, v_j)$.

As defined in Sect. 5.3, $F_{p_t,v_j}(x) = \int_{x-R}^{x+R} \mathcal{G}_{p_t,v_j}(x)\mathrm{d}x$. Then, we can approximate F_{p_t,v_j} as f_{p_t,v_j} using g_{p_t,v_j} stored in the UPB-forest UF_V built on U. In particular, $f_{p_t,v_j}(k)$ on slot k can be calculated as $\sum_{k-\lceil R/s\rceil}^{k+1+\lceil R/s\rceil} g_{p_t,v_j}(y)$. Hence, we can get that for $F_{p_t,v_j}(x)$ ($x \in [k \times s, (k+1) \times s)$) $\le \int_{k\times s-R}^{(k+1)\times s+R} \mathcal{G}_{p_t,v_j}(x)\mathrm{d}x \le f_{p_t,v_j}(k)$.

Lemma 5.5 *Given a pivot set P, and two uncertain objects u_i and v_j, $u\pi(u_i, v_j) = min_{p_t \in P} \sum_{\lfloor (d(u_i.c, p_t) - u_i.r)/s \rfloor}^{\lceil (d(u_i.c, p_t) + u_i.r)/s \rceil - 1} (g_{p_t,u_i}(k) \times f_{p_t,v_j}(k))$ is an upper bound on $\pi(u_i, v_j)$.*

Proof According to Lemma 5.3, $u\pi(u_i, v_j) = \min_{p_t \in P} \int_{d(v_j.c,p_t)-v_j.r-R}^{d(v_j.c,p_t)+v_j.r+R} (g_{p_t,u_i}(x) \times F_{p_t,v_j}(x))$. Due to the pivot mapping, we can get that $\phi_{p_t}(u_i) = [d(u_i.c, p_t) - u_i.r, d(u_i.c, p_t) + u_i.r]$, and thus, g_{p_t,u_i} has nonzero values for the slots $k \in [\lfloor (d(u_i.c, p_t) - u_i.r)/s \rfloor, \lceil (d(u_i.c, p_t) + u_i.r)/s \rceil - 1]$. Since $F_{p_t,v_j}(x) \le f_{p_t,v_j}(k)$ $(x \in [k \times s, (k+1) \times s))$, $u\pi(u_i, v_j) \le \min_{p_t \in P} \sum_{\lfloor (d(u_i.c,p_t)-u_i.r)/s \rfloor}^{\lceil (d(u_i.c,p_t)+u_i.r)/s \rceil - 1} (g_{p_t,u_i}(k) \times f_{p_t,v_j}(k))$. The proof completes. □

Note that, since MPRJ is symmetric, $u\pi(u_i, v_j)$ can be calculated as $\min_{p_t \in P} \sum_{\lfloor (d(u_i.c,p_t)-u_i.r)/s \rfloor}^{\lceil (d(u_i.c,p_t)+u_i.r)/s \rceil - 1} (g_{p_t,v_j}(k) \times f_{p_t,u_i}(k))$. Consider MPRJ$(U, V, 1.5, 0.5)$ in Fig. 11. Assume that $P = \{u_1.c, v_2.c\}$, we can get that $u\pi(u_1, v_1) = g_{u_1.c,u_1}(0) \times \sum_{-2}^{3} g_{u_1.c,v_1}(k) = 1$ and $u\pi(u_1, v_2) = g_{u_1.c,u_1}(0) \times \sum_{-2}^{3} g_{u_1.c,v_2}(k) = 0$. Therefore, we can prune the uncertain object pair $\langle u_1, v_2 \rangle$ due to $u\pi(u_1, v_2) < \theta$ according to Theorem 5.1.

As defined in Sect. 5.3, $F'_{p_t,v_j}(x) = \int_0^{R-x} \mathcal{G}_{p_t,v_j}(x)\mathrm{d}x$. Then, we can also approximate F'_{p_t,v_j} as f'_{p_t,v_j} using g_{p_t,v_j} stored in the UPB-forest UF_V over V. In particular, $f'_{p_t,v_j}(k)$ on slot k can be calculated as $\sum_0^{\lfloor R/s \rfloor - k - 2} g_{p_t,v_j}(y)$. Hence, we can get that for $F'_{p_t,v_j}(x)$ $(x \in [k \times s, (k+1) \times s)) \ge \int_0^{R-(k+1)\times s} \mathcal{G}_{p_t,v_j}(x)\mathrm{d}x \ge f'_{p_t,v_j}(k)$.

Lemma 5.6 *Given a pivot set P, and two uncertain objects u_i and v_j, $l\pi(u_i, v_j) = max_{p_t \in P} \sum_{\lfloor (d(u_i.c,p_t)-u_i.r)/s \rfloor}^{\lceil (d(u_i.c,p_t)+u_i.r)/s \rceil - 1} (g_{p_t,u_i}(k) \times f'_{p_t,v_j}(k))$ is an lower bound on $\pi(u_i, v_j)$.*

Proof According to Lemma 5.3, $u\pi(u_i, v_j) = \min_{p_t \in P} \int_{d(v_j.c,p_t)-u_i.r-R}^{R-d(v_j.c,p_t)+u_i.r} (g_{p_t,i}(x) \times F'_{p_t,v_j}(x))$. Due to the pivot mapping, $\phi_{p_t}(u_i) = [d(u_i.c, p_t) - u_i.r, d(u_i.c, p_t) + u_i.r]$, and thus, g_{p_t,u_i} has nonzero values on slots $k \in [\lfloor (d(u_i.c, p_t) - u_i.r)/s \rfloor, \lceil (d(u_i.c, p_t) + u_i.r)/s \rceil - 1]$. Since $F'_{p_t,v_j}(x) \le f'_{p_t,v_j}(k)$ $(x \in [k \times s, (k+1) \times s)$, $l\pi(u_i, v_j) \ge \max_{p_t \in P} \sum_{\lfloor (d(u_i.c,p_t)-u_i.r)/s \rfloor}^{\lceil (d(u_i.c,p_t)+u_i.r)/s \rceil - 1} (g_{p_t,u_i}(k) \times f'_{p_t,v_j}(k))$. The proof completes. □

Note that, since MPRJ is symmetric, $l\pi(u_i, v_j)$ can be calculated as $\max_{p_t \in P} \sum_{\lfloor (d(v_j.c,p_t)-v_j.r)/s \rfloor}^{\lceil (d(v_j.c,p_t)+v_j.r)/s \rceil - 1} (g_{p_t,v_j}(k) \times f'_{p_t,u_i}(k))$. Consider MPRJ$(U, V, 1.5, 0.5)$ again in Fig. 11. Assuming that $P = \{u_1.c, v_2.c\}$, we can get that $l\pi(u_1, v_1) = g_{u_1.c,u_1}(0) \times \sum_0^{-1} g_{u_1.c,v_1}(k) = 0$. Thus, we cannot validate the uncertain object pair $\langle u_1, v_1 \rangle$ by Theorem 5.1, as $l\pi(u_1, v_1) < \theta$.

Algorithm 6 UPB-forest based MPRJ Algorithm (UfMJA)

```
Input: UPB-forests $UF_U$ and $UF_V$ to index $U$ and $V$, $R$, $\theta$
Output: the result set $S_r$ of $MPRJ(U, V, R, \theta)$
1: for each B$^+$-tree $B_{Ut} \in UT_U$ and $B_{Vt} \in UT_V$ $(0 \le t < |P|)$ do
2:   $L_U \leftarrow \emptyset$, $L_V \leftarrow \emptyset$
3:   traverse $B_{Ut}$ and $B_{Vt}$ to get the first leaf entries $E_U$ and $E_V$
4:   while $E_U \ne \emptyset$ or $E_V \ne \emptyset$ do
5:     if $E_V = \emptyset$ or $E_U.key < E_V.key$ then   // all the entries in $UT_V$ are visited
                                   // or the key value of the current $E_U$ is smaller than that of $E_V$
6:       Verify($E_U$, $L_O$)
7:       insert $E_U$ into $L_Q$
8:       $E_U \leftarrow E_U.get\_next()$   // get the next leaf entry $E_U$ in $B_{Ut}$
9:     else   // $E_U = \emptyset$ or $E_U.key \ge E_V.key$
10:      Verify($E_V$, $L_Q$)
11:      insert $E_V$ into $L_O$
12:      $E_V \leftarrow E_V.get\_next()$   // get the next leaf entry $E_V$ in $B_{Vt}$
13: for each $\langle u_i, v_j \rangle$ in $U \times V$ do
14:   if $l\pi(u_i, v_j) \ge \theta$ then   // $l\pi(u_i, v_j) = max\{l\pi_t(u_i, v_j) \mid 0 \le t < |P|\}$
15:     insert $\langle u_i, v_j \rangle$ into $S_r$   // Theorem 5.1
16:   else if $u\pi(u_i, v_j) < \theta$ then   // $u\pi(u_i, v_j) = min\{u\pi_t(u_i, v_j) \mid 0 \le t < |P|\}$
17:     prune $\langle u_i, v_j \rangle$   // Theorem 5.1
18:   else if $\pi(u_i, v_j) \ge \theta$
19:     insert $\langle u_i, v_j \rangle$ into $S_r$
20: return $S_r$

Function: Verify($E_U$, $L$)
21: $E_V \leftarrow L.get\_last()$   // get the last entry in $L$
22: while $v_{jy} \ne \emptyset$ do
23:   if $E_V.key \le \lfloor R/s \rfloor - E_U.key - 2$ then   // Lemma 5.6
24:     update $l\pi_t(u_i, v_j)$   // $i = E_U.id, j = E_V.id$
25:   if $E_V.key < E_U.key - \lceil R/s \rceil$ then   // Lemma 5.5
26:     delete $E_V$ from $L$
27:   else
28:     update $u\pi_t(u_i, v_j)$
29:   $E_V \leftarrow E_V.get\_pre()$   // get the previous entry in $L$
```

Based on Lemmas 5.5 and 5.6, and Theorem 5.1, UfMJA adopts a filtering-and-refinement framework. The pseudo-code of UfMJA is presented in Algorithm 6. The algorithm takes as inputs UPB-forests UF_U and UF_V built on U and V, respectively, a search radius R, and a probability threshold θ, and outputs the result S_r of MPRJ(U, V, R, θ). In the filtering phase, for the B$^+$-trees B_{Ut} and B_{Vt} $(0 \le t < |P|)$ of UF_U and UF_V, UfMJA first initializes two lists L_U and L_V to empty and gets the first leaf entries E_U and E_V of the B$^+$-trees B_{Ut} and B_{Vt}, respectively (lines 2–3). Then, the algorithm performs a *while loop* to visit the leaf entries in ascending order of their *keys* values, until all leaf entries of both B_{Ut} and B_{Vt} are evaluated (i.e., E_U and E_V are empty) (lines 4–12). Each time, if all the leaf entries of B_{Vt} are visited (i.e., $E_V = \emptyset$) or $E_U.key \le E_V.key$, the *Verify* function is invoked (line 5) to update $u_t\pi(u_i, v_j)$ and $l_t\pi(u_i, v_j)$ $(i = E_U.id,\ j = E_V.id \in L_V)$ and to delete unqualified E_V from L_V using Lemmas 5.5 and 5.6. Note that, $u_t\pi(u_i, v_j)$ and $l_t\pi(u_i, v_j)$ are equal to $u\pi(u_i, v_j)$ and $l\pi(u_i, v_j)$ as computed using a specific pivot p_t. After that, UfMJA inserts E_U into the list L_U and gets the next leaf entry E_U in B_{Ut} (lines 7–8). Otherwise, if E_U is empty (i.e., all leaf entries in B_{Ut} have been visited) or $E_U.key > E_V.key$, the algorithm invokes *Verify* to update $u_t\pi(u_i, v_j)$ and $l_t\pi(u_i, v_j)$ $(i = E_U.id \in$

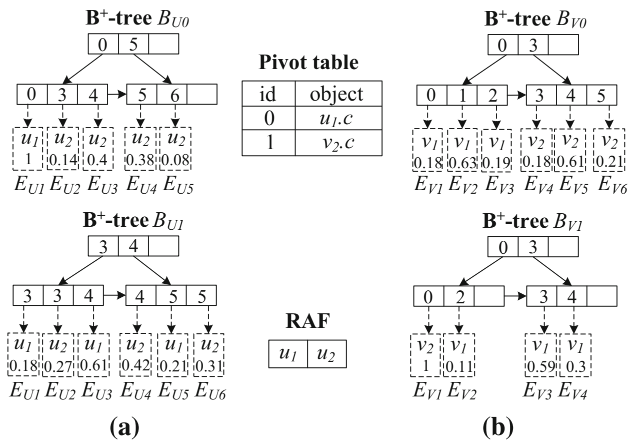

Fig. 12 UPB-forests built on U and V. **a** UF_U and **b** UF_V

$L_U, j = E_V.id$), and to delete unqualified E_U from L_U using Lemmas 5.5 and 5.6. Thereafter, UfMJA inserts E_V into the list L_V and gets the next leaf entry E_V in B_{Vt} (lines 11–12). In the refinement phase, for each uncertain object pair $\langle u_i, v_j\rangle$ in $U \times V$, the algorithm inserts $\langle u_i, v_j\rangle$ into S_r if $l\pi(u_i, v_j) \geq \theta$ (lines 14–15), and it filters $\langle u_i, v_j\rangle$ if $u\pi(u_i, v_j) < \theta$ (lines 16–17). Otherwise, it computes $\pi(u_i, v_j)$ and inserts $\langle u_i, v_j\rangle$ into S_r if $\pi(u_i, v_j) \geq \theta$. Finally, the result set S_r is returned (line 20).

Example 6 We illustrate the working of the UfMJA using the MPRJ(U, V, 1.5, 0.5) example shown in Fig. 11, with UPB-forests built on U and V as depicted in Fig. 12. For B$^+$-trees B_{U0} and B_{V0}, UfMJA sets L_U and L_V to empty and traverses the two trees to get the first leaf entries E_{U1} and E_{V1}. Then, it performs a while loop. In the first iteration, since $E_{U1}.key = E_{V1}.key$, UfMJA calls *Verfiy*(E_{V1}, L_U), inserts E_{V1} into L_V, and gets the next leaf entry E_{V2} in B_{V0}. In the second iteration, as $E_{U1}.key < E_{V2}.key$, *Verify*(E_{U1}, L_V) is invoked, where $u_0\pi(u_1, v_1)$ is updated to 0.18 due to Lemmas 5.5. Then, UfMJA inserts E_{U1} into L_U and gets the next leaf entry E_{U2} in B_{U0}. In the third iteration, as $E_{U2}.key > E_{V2}.key$, the algorithm calls *Verify*(E_{V2}, L_U), in which $u_0\pi(u_1, v_1)$ is updated to 0.81. Then, UtMJA inserts E_{V2} into L_V and gets the next leaf entry E_{V3} in B_{V0}. The algorithm proceeds in the same manner until all leaf entries in B_{U0} and B_{V0} are visited, with $u\pi_0(u_1, v_1) = 1$, $u\pi_0(u_1, v_2) = 0$, $u\pi_0(u_2, v_1) = 0.4748$, and $u\pi_0(u_2, v_2) = 0.9856$. Next, B_{U1} and B_{V1} are traversed similarly, with $u\pi_1(u_1, v_1) = 1$, $u\pi_1(u_1, v_2) = 0$, $u\pi_1(u_2, v_1) = 0.9659$, and $u\pi_1(u_2, v_2) = 0$. Subsequently, $\langle u_1, v_1\rangle$ is inserted into S_r due to $\pi(u_1, v_1) > 0.5$. Other uncertain object pairs $\langle u_i, v_j\rangle$ are pruned due to $u\pi(u_i, v_j) < 0.5$ using Theorem 5.1. Finally, the algorithm returns the result set $S_r = \{\langle u_1, v_1\rangle\}$. □

5.5 Discussion

We proceed to analyze the CPU and I/O costs of query processing based on the UPB-forest.

5.5.1 CPU cost

As for UfMA, the CPU cost of UfMA can be estimated by using the number of distance computations, which includes distance computations for computing $\phi_{p_t}(R_q)$ w.r.t. each $p_t \in P$, and that for verifying whether an uncertain object u_i is a real answer object. According to Lemma 5.1, in the worst case, UfMA needs to verify all the uncertain objects u_i with $\phi_{p_t}(u_i) \cap \phi_{p_t}(R_q) \neq \emptyset$ for every $p_t \in P$, i.e., $\phi(u_i) \cap \phi(R_q) \neq \emptyset$. Hence, the CPU cost of UfMA in terms of distance computations can be calculated as:

$$\text{UfMA}_C = |P| + \sum_{u_i \in U} I'(\phi(u_i), \phi(R_q)) \tag{7}$$

where $I'(a, b) = \begin{cases} 1 & a \text{ intersects with } b \\ 0 & \text{otherwise} \end{cases}$.

Similarly, the CPU cost of UfMUA in terms of distance computations can be calculated as:

$$\text{UfMUA}_C = |P| + \sum_{u_i \in U} I'(\phi(u_i), \phi(R_{uq})) \tag{8}$$

For UfMJA, the number of distance computations can be estimated by using the sum of distance computations needed when finding the instances v_j intersecting with R_{u_i} for each instance u_i. Thus, the CPU cost of UfMJA in terms of distance computations can be calculated as:

$$\text{UfMJA}_C = \sum_{u_i \in U} \sum_{v_j \in V} I'(\phi(u_i), \phi(R_{v_j})) \tag{9}$$

5.5.2 I/O cost

The I/O cost of UfMA on the UPB-forest contains two parts, i.e., B$^+$-tree and RAF page accesses. To obtain the number of B$^+$-tree page accesses, it is sufficient to sum all the B$^+$-tree B_t ($0 \leq t < |P|$) nodes whose MBBs intersect with the search region $\phi_{p_t}(R_q)$. In addition, the number of RAF page accesses can be estimated by using UfMA$_C/f$, where UfMA$_C$ is used to estimate the total number of the uncertain objects accessed, and f represents the average number of the uncertain objects per RAF page because the uncertain objects visited in RAF are stored close to each other. Hence, the I/O cost of UfMA in terms of page accesses can be calculated as:

$$\text{UfMA}_{IO} = \sum_{M_i \in B_t (0 \le t < |P|)} I'(M_i, \phi_{p_t}(R_q)) + \frac{\text{UfMA}_C}{f} \quad (10)$$

The I/O cost of UfMUA in terms of page accesses can be calculated similarly:

$$\text{UfMUA}_{IO} = \sum_{M_i \in B_t (0 \le t < |P|)} I'(M_i, \phi_{p_t}(R_{uq})) + \frac{\text{UfMUA}_C}{f} \quad (11)$$

As UfMJA traverses UPB-forests built on two uncertain object sets U and V only once, it is sufficient to sum the number of B$^+$-tree leaf pages and RAF pages in the UPB-forests. Let $|B_{Ut}| \in UF_U$ ($|B_{Vt}| \in UF_V$) denote the total number of B$^+$-tree leaf pages of B_{Ut} (B_{Vt}), and let f_U (f_V) represent the average number of the objects per RAF page for UF_U (UF_V). In brief, the I/O cost of UfMJA in terms of page accesses can be calculated as:

$$\text{UfMJA}_{IO} = \sum_{0 \le t < |P|} |B_{Ut}| + \sum_{0 \le t < |P|} |B_{Vt}| + \frac{|U|}{f_U} + \frac{|V|}{f_V} \quad (12)$$

6 Pivot selection

In this section, we introduce a quality criterion for pivot sets and then propose an efficient pivot selection algorithm to obtain a high-quality pivot set based on the newly defined quality criterion.

According to the cost analysis in Sects. 4.5 and 5.5, the costs of MPRQ, MPRQU, and MPRJ using the UPB-tree or the UPB-forest depend significantly on the pivot set P. Hence, it makes sense to define a criterion to measure the quality of pivots. Since our algorithms follow the filtering-and-refinement framework, they prune an uncertain object u using its probability upper bound $Pr(D(\phi(q), \phi(u)) \le R)$, to avoid unnecessary computation of $Pr(d(q, u) \le R)$ (as $D(\ )$ is the lower bound of $d(\ )$). In order to achieve high performance, i.e., fewer computations of $Pr(d(q, u) \le R)$, the lower bound distances should be close to the actual distances, i.e., the mapping to the vector space should preserve the proximity in the metric space.

Let $\epsilon(x, y)$ $(x, y \in U)$ denote the random variable defined as $D(\phi(x), \phi(y))/d(x, y)$. Note that, $0 \le \epsilon(x, y) \le 1$. The expected value of $\epsilon(x, y)$ can be calculated as:

$$E[\epsilon(x, y)] = \int_0^1 z \times Pr(D(\phi(x), \phi(y))/\mathrm{d}(x, y) = z)\mathrm{d}z \quad (13)$$

To obtain an optimal pivot set P, we need to pick pivots that maximize $E[\epsilon(x, y)]$. However, it is costly to obtain the distribution of $\epsilon(x, y)$ for every uncertain object pair. Therefore, random sampling is exploited to estimate the value of $E[\epsilon(x, y)]$, denoted as $\hat{E}[\epsilon(x, y)]$, for achieving the following goal:

$$Pr(|E[\epsilon(x, y)] - \hat{E}[\epsilon(x, y)]| < \xi) > 1 - \delta \quad (14)$$

According to the Hoeffding's inequality [19],

$$Pr(|E[\epsilon(x, y)] - \hat{E}[\epsilon(x, y)]| < \xi) > 1 - 2 \cdot \exp(-2T\xi^2) \quad (15)$$

Here, T represents the sampled size. Hence, to reach the target stated in Eq. 14, $T \ge \frac{1}{2\xi^2}\log(\frac{2}{\delta})$.

Given a set U_S ($|U_S| \ge \frac{1}{2\xi^2}\log(\frac{2}{\delta})$) of uncertain instance pairs sampled from an uncertain object set U, the uncertain instance object u_{ij} with higher probability $Pr(u_{ij})$ contributes more to derive constrictive upper bound $u\pi(u_i, R_q)$. Hence, taking the probability of the uncertain instance object into consideration, we offer a criterion to measure the quality of a pivot set P based on the sampled metric uncertain data as follows.

Definition 6.1 Given a sample U_S of instance object pairs on the metric uncertain data, the quality of a pivot set P is defined as

$$\epsilon_P = \frac{1}{|U_S|} \sum_{<u_{ij}, u_{mn}> \in U_S} \left(\frac{D(\phi(u_{ij}), \phi(u_{mn}))}{d(u_{ij}, u_{mn})} \times Pr(u_{ij}) \times Pr(u_{mn}) \right)$$

Algorithm 7 Incremental Pivot Selection Algorithm (IPS)

```
Input: a set U of uncertain objects, the number t of pivots requested
Output: a set P of t pivots (i.e., |P| = t)
1: $P_c \leftarrow$ HF(cert(U), cp_scale)  // get a candidate pivot set $P_c$ with
                                          // $|P_c|$ = cp_scale
2: $P \leftarrow \varnothing$
3: while $|P| < t$ do
4:    select $p$ from $P_c$ with the maximum $\epsilon_{P \cup \{p\}}$
5:    $P \leftarrow P \cup \{p\}$ and $P_c \leftarrow P_c - \{p\}$
6: return P
```

The more the pivots, the better the pruning capability. But the cost of using transformed objects will also be higher (to be studied in Sect. 7). To be more specific, if P contains more pivots, we can expect a larger $D(\phi(u_{ij}), \phi(u_{mn}))$; then, $D(\phi(u_{ij}), \phi(u_{mn}))$ approaches $d(u_{ij}, u_{mn})$ and hence approaches 1. Therefore, we can prune more uncertain objects using a larger pivot set. On the other hand, the cost (e.g., $D(\phi(u_{ij}), \phi(u_{mn}))$ computation cost) to prune unqualified uncertain objects increases as well.

Table 2 Statistics of the datasets used

Dataset	Cardinality	Dim.	Ins.	Measurement
English	60 K	1–22	4.9	*Edit distance*
Color	56 K	8	2.9	L_5-*norm*
SF	176 K	2	6.9	L_2-*norm*
DNA	200 K	108	14.8	*Cosine similarity under tri-gram counting space*
Synthetic (set)	[50 K, 250 K]	20	4.76	*Jaccrad coefficient*

As shown in the literature [3], the problem of determining whether a pivot set P ($|P| \geq 1$) exists that makes the penalty function of a defined criterion no larger than a nonnegative real number is *NP-hard*. Thus, it is appropriate to apply a greedy pivot selection algorithm for picking a high-quality pivot set P (from U) of a fixed size to maximize ϵ_P. However, the time complexity O($|P| \times |U|$) still remains high, especially for a large uncertain object set. Note that, since pivots are certain objects, to select P ($\subseteq U$), we need to replace the uncertain objects in U with a set of representative certain objects, denoted as $cert(U)$.

To further reduce time complexity, we present an *incremental pivot selection algorithm* (IPS), with the pseudo-code depicted in Algorithm 7. IPS first employs HF algorithm [40] to obtain outliers as candidate pivots P_c, and then, it selects effective pivots from P_c incrementally. As pointed out in the literature [5], good pivots are usually outliers, but outliers are not always good pivots. The time complexity of IPS is reduced to O($|P| \times |P_c|$), in which the cardinality of P_c (i.e., $|P_c|$) is small, and is only related to the distribution of the object set. In this paper, we fix $|P_c|$ to 40 (as elsewhere [29]), which is enough to find all the outliers in our experiments.

7 Experimental evaluation

In this section, we experimentally evaluate the performance of the UPB-tree and the UPB-forest. First, we study the effectiveness of our pivot selection algorithm. Then, we compare the UPB-tree and the UPB-forest with state-of-the-art indexes. Finally, MPRQU and MPRJ performance are explored. We implemented the UPB-tree, the UPB-forest, the MPRQ, the MPRQU, and the MPRJ algorithms in C++. All experiments were conducted on an Intel Core 2 Duo 2.93GHz PC with 8GB RAM.

We employ four real datasets, namely *English*, *Color*, *SF*, and *DNA*, as depicted in Table 2. *English*[1] contains the words extracted from the English language dictionary, and edit distance is employed to compute the distance between two words. *Color*[2] denotes the first eight dimensions of color histograms extracted from an image database, and the L_5-norm is utilized to compare the color image features. *SF*[3] consists of the locations in San Francisco, and the L_2-norm is used to measure their similarity. *DNA*[4] includes DNA data, and hamming distance is utilized to measure similarity. In our experimental settings, *English* and *Color* use the *object-level model* to represent the uncertainty. Specifically, every uncertain object is represented using m ($100 \leq m \leq 300$) instances with their distances to the word or image feature in *English* or *Color* bounded by Υ, resulting in a total of 12M instances for the default *English* and 11.2M instances for the default *Color*. The associated instance probability follows two popular distributions, namely *Normal* and *Arbitrary*, for *English* and *Color*, respectively. *SF* and *DNA* utilize the *bi-level model* to represent the uncertainty. For *SF*, the uncertain object u is denoted as an uncertain region centered at every location in *SF* with the radius bounded by Υ, and the probability for u located in the uncertain region follows the *Uniform* distribution. For *DNA*, each uncertain string uses the character element model [21] as the underlying uncertain model. *Synthetic* (*set*) datasets based on both the object-level and bi-level models are also generated, in which every set in *Synthetic* has 30 to 60 elements, and the Jaccard coefficient is utilized to measure similarity. Note that, for the bi-level model, the uncertain *set* uses the p-set model [17] as its underlying uncertain model. Table 2 lists the statistics of the datasets used in our experiments. All index structures are configured to use a fixed disk page size of 4KB.

We investigate the efficiency of MPRQ, MPRQU, and MPRJ algorithms using the UPB-tree and the UPB-forest when varying different parameters, as shown in Table 3. In each experiment, we change one parameter and fix the others to their default values. The main performance metrics include the number of page accesses (*PA*), the number of distance computations (*compdists*), and the CPU time. Each measurement we report is an average of 50 random queries.

[1] Available at http://www.sisap.org/Metric_Space_Library.html.

[2] Available at http://www.sisap.org/Metric_Space_Library.html.

[3] Available at http://www.dbs.informatik.uni-muenchen.de/~seidl.

[4] Available at http://www.ncbi.nlm.nih.gov/genome.

Table 3 Parameter ranges and default values

Parameter	Setting	Default
The number of pivots $\|P\|$	1, 3, 5, 7, 9	5
Uncertain region size γ (% of d^+)	4, 6, 8, 10, 12	8
The number m of instances	100, 150, 200, 250, 300	200
The number n of slots	200, 400, 600, 800, 1000	1000
Search radius R (% of d^+)	2, 4, 8, 16, 32	8
Probability threshold θ	0.1, 0.3, 0.5, 0.7, 0.9	0.5
Cardinality	50 K, 100 K, 150 K, 200 K, 250 K	150 K

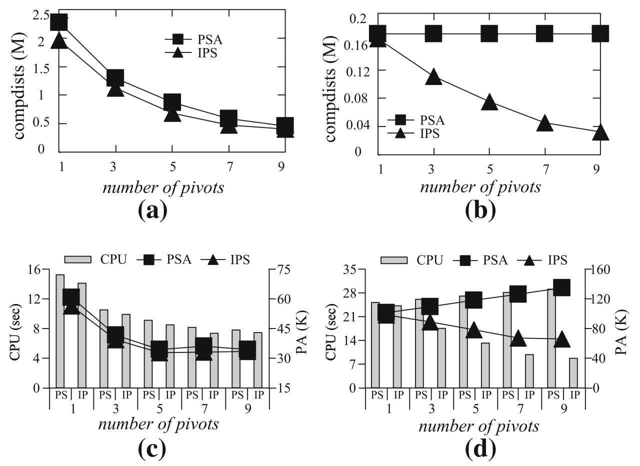

Fig. 13 Efficiency of pivot selection methods versus $|P|$. **a** *English*, **b** *DNA*, **c** *English*, **d** *DNA*

7.1 Pivot selection effectiveness

The first set of experiments compares the effectiveness of our pivot selection algorithm IPS with that of the existing pivot selection algorithm PSA [3]. We utilize the MPRQ on the UPB-tree and the UPB-forest to investigate the effectiveness of the pivot selection algorithms. Figure 13 depicts the results using datasets *English* and *DNA*, where abbreviations of pivot selection algorithms (**IP** for IPS, **PS** for PSA) are shown at the bottom of each column. As observed, IPS performs better than PSA. The reason is that MPRQ performance is highly related to ϵ_P as defined in Definition 6.1, and IPS tries to maximize ϵ_P. The second observation is that the number of distance computations decreases as the number of pivots grows. This is because, using more pivots, the query efficiency improves as ϵ_P becomes larger, incurring fewer distance computations. The third observation is that PA and CPU time first drop and then stay stable or increase when the number of pivots increases. However, PA and CPU time of PSA on *DNA* increase with the growth of the number of pivots. The reason is that the cost of filtering unqualified uncertain objects grows as the number of pivots ascends. Hence, MPRQ achieves better performance for all performance metrics, when the number of pivots approaches the *intrinsic dimensionality* (i.e., Ins. Dim. for short in Table 2) of the dataset [40].

7.2 Comparisons with the UP-Index

The second set of experiments compares the efficiency of the UPB-tree and the UPB-forest with that of state-of-the-art UP-Index [3] using MPRQ.

Construction cost First, Tables 4 and 5 show the construction costs and storage sizes for the UP-Index, the UPB-tree, and the UPB-forest, using real datasets. Clearly, the UPB-tree and the UPB-forest have much lower construction costs in terms of the number of page accesses (i.e., *PA*) and the storage sizes (denoted as *Storage*) in most cases. The reason is that our UPB-tree and UPB-forest only store necessary information, while the UP-Index needs to preserve the histograms w.r.t. the whole distance range for every uncertain object. Notice that, on *English*, the UP-Index has smaller *PA* than the UPB-tree. This is because, *English* has small distance range (i.e., [0, 22]), resulting in low redundant storage for UP-Index, while UPB-tree needs to store additional information (e.g., SFC values and MBB information) to achieve search

Table 4 Construction cost of the UP-Index and the UPB-tree

	UP-Index		UPB-tree	
	English	*Color*	*English*	*Color*
PA	127,696	704,279	523,830	564,418
compdists	68,105,000	56,000,000	68,105,000	56,000,000
Time (s)	1970	293	553	552
Storage (KB)	510,784	2,817,116	489,669	707,909

Table 5 Construction cost of the UP-Index and the UPB-forest

	UP-Index		UPB-forest	
	SF	*DNA*	*SF*	*DNA*
PA	1,709,708	280,702	454,293	17,700
compdists	874,780	1,000,000	874,780	1,000,000
Time (s)	1509	105	795	333
Storage (KB)	6,838,832	1,122,808	608,037	506,306

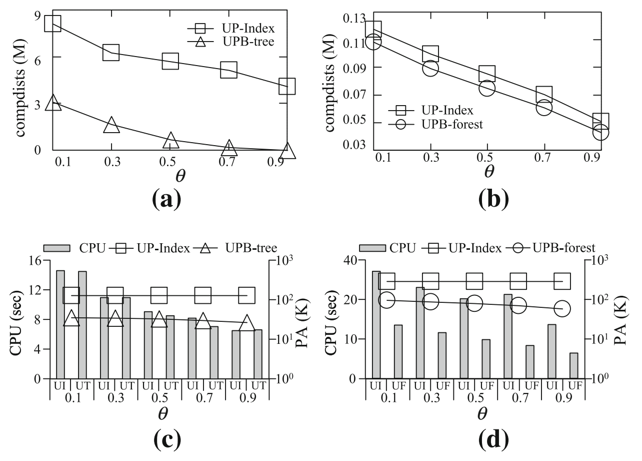

Fig. 14 MPRQ performance versus θ. **a** *English*, **b** *DNA*, **c** *English*, **d** *DNA*

efficiency. However, the construction CPU time (denoted as *Time*) of UPB-tree and UPB-forest can be higher than that of UP-Index. This is because UPB-tree and UPB-forest need additional CPU cost to build the B$^+$-tree(s), while UP-Index stores the histograms as a table.

Effect of θ Figure 14 illustrates the influence of θ on the efficiency of MPRQ algorithms using *English* and *DNA*. Abbreviations of indexes (**UI** for UP-Index, **UT** for UPB-tree, **UF** for UPB-forest) are shown at the bottom of each column. The first observation is that the UPB-tree- and the UPB-forest-based MPRQ algorithms are better in terms of *compdists*. The reason is that our pivot selection algorithm selects effective pivots to avoid a large number of distance computations, and the MPRQ algorithms can avoid unnecessary distance computations by validating or pruning uncertain objects based on the probability lower and upper bounds. In contrast, the UP-Index cannot validate uncertain objects because it has no probability lower bounds. The second observation is that the UPB-tree- and the UPB-forest-based MPRQ algorithms perform much better in terms of I/O cost. This is because the UP-Index has to scan the entire index to prune uncertain objects. Nevertheless, for the UPB-tree and the UPB-forest, they achieve I/O efficiency since only qualified B$^+$-tree entries and uncertain objects are visited based on Lemmas 4.1 and 4.4, 5.1 and 5.2,

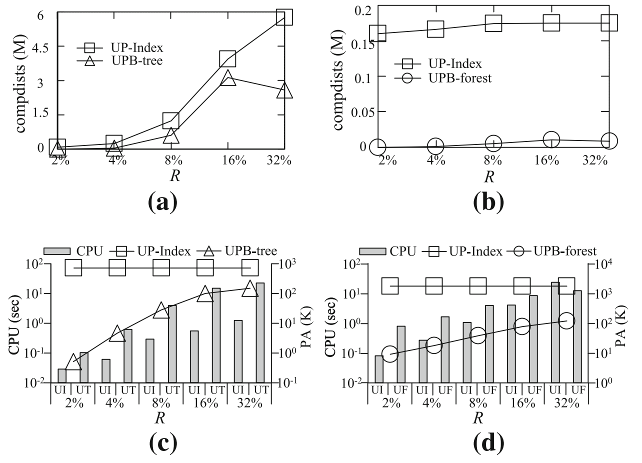

Fig. 15 MPRQ performance versus R. **a** *Color*, **b** *SF*, **c** *Color*, **d** *SF*

and Theorem 4.1. In addition, the MPRQ cost drops with the growth of θ because the pruning power increases with θ.

Effect of R Figure 15 depicts the performance of MPRQ when varying R, using *Color* and *SF*. As expected, UPB-tree- and UPB-forest-based MPRQ algorithms perform better in terms of *compdists* and *PA*. However, the CPU time of UPB-tree and UPB-forest is larger than that of UP-Index. This is because, UPB-tree and UPB-forest need additional CPU cost to compute the probability lower and upper bounds by traversing the indexes. The query costs (including *compdists*, *PA*, and the CPU time) increase with R, since the search space grows with R. In contrast, due to Lemma 4.4, the validation power of the UPB-tree increases with R, resulting in fewer distance computations, which makes it possible that the *compdists* drops when R reaches 32% on *Color* as well as also makes it possible that the CPU time of UPB-forest is less than that of UP-Index on *SF*.

Effect of n In order to observe the impact of the number n of slots on the efficiency of the indexes, we employ *Color* and *SF* datasets since the ranges of their distance functions are real numeric. Figure 16 plots the results with respect to various n values. As observed, *compdists* slightly drops with the growth of n. The reason is that, as n increases, the discrete case approaches to the continuous case, which helps tighten the probability upper and lower bounds calculations, hence reducing the number of distance computations. However, for the UP-Index and the UPB-forest, the number of page accesses increases with n, since their storage costs grow with n in order to preserve the pre-computed histogram or approximated discrete PDF for every uncertain object, incurring higher I/O cost.

Effect of γ **and** m Figures 17 and 18 show the performance of MPRQ when varying γ and m. We see that the query costs (including *compdists*, *PA*, and the CPU time) increase with the growth of m. The reason is that the search space grows with m. Notice that the query costs are not sensitive

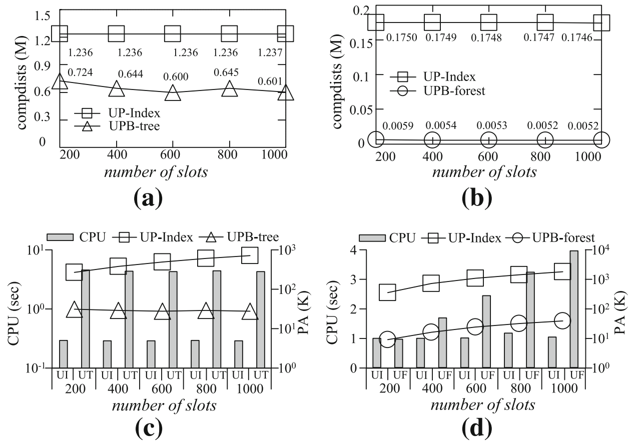

Fig. 16 MPRQ performance versus n. **a** *Color*, **b** *SF*, **c** *Color*, **d** *SF*

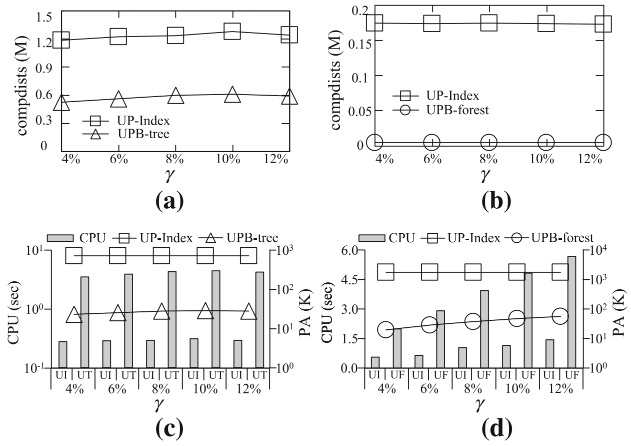

Fig. 17 MPRQ performance versus γ. **a** *Color*, **b** *SF*, **c** *Color*, **d** *SF*

to γ, and they drop slightly when γ reaches 12% on *Color*. This is because, for the uncertain objects with larger uncertain regions, the total number of qualifying answer objects drops with fixed search radius.

Effect of *cardinality* In order to study the scalability of the UPB-tree and the UPB-forest, we employ *Synthetic* (*set*) datasets. Figure 19 plots the MPRQ performance as a function of cardinality. Clearly, *compdists*, *PA*, and the CPU time grow linearly with the cardinality. This occurs because the search space grows as the dataset cardinality increases. Since both object-level and bi-level models are used on *Synthetic* (*set*) datasets, a comparison between UPB-tree and UPB-forest is also investigated. As observed, *compdists* of UPB-tree is much larger than that of UPB-forest, but the CPU time of UPB-tree is smaller. This is because one distance computation computes the distance between uncertain instances for the UPB-tree, but computes the distance between uncertain objects for the UPB-forest. Note that, the distance computational cost for uncertain objects is much more costly than that for uncertain instances on the same dataset. In addition, the I/O cost of UPB-forest is slightly

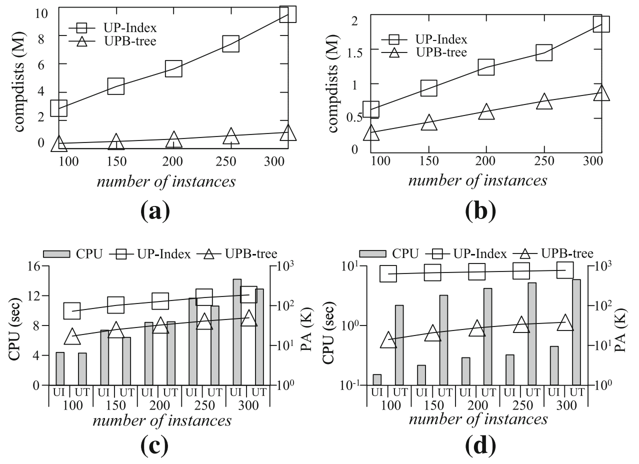

Fig. 18 MPRQ performance versus m. **a** *English*, **b** *Color*, **c** *English*, **d** *Color*

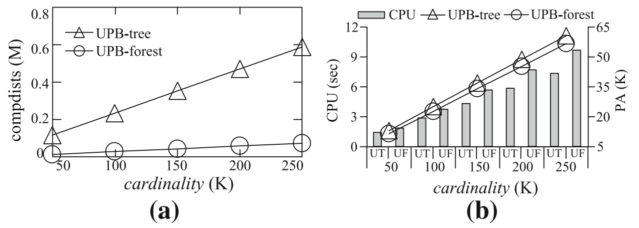

Fig. 19 MPRQ performance versus cardinality. **a** Synthetic (set) and **b** synthetic (set)

smaller than that of UPB-tree, because the bi-level model needs fewer storage size than the object-level model.

7.3 Comparison with the U-Quadtree, UPR-tree, and CSQ

To further explore the efficiency of the UPB-tree and the UP-forest, we also compare them with the state-of-the-art U-Quadtree [44] and CSQ [17] for specific uncertain metric spaces using MPRQ. In addition, UPR-tree that uses a R-tree instead of a B$^+$-tree is also developed for comparison. Table 6 shows the experimental results on *Color* and *Synthetic* datasets, where abbreviations (UQ for U-Quadtree, UT for UPB-tree, UR for UPR-tree, and UF for UPB-forest) are used. Here, there is no value of *compdists* for U-Quadtree, because the verification of the U-Quadtree does not need any distance computation. Also, there is no value for *PA* for CSQ, since CSQ is an in-memory method without any I/O cost. It is observed that the I/O cost of UPB-tree is fewer than that of U-Quadtree and UPR-tree. This is because SFC utilized by the UPB-tree can reduce the storage cost of pre-computed distances and meanwhile preserve the locality. However, *compdists* of UPB-tree is slightly larger than that of UPR-tree, since the UPB-tree uses approximated discrete distances to perform the SFC mapping for continuous distance func-

Table 6 Comparisons with U-Quadtree, UPR-tree, and CSQ

	Color			Synthetic		
	UQ	*UR*	*UT*	*CSQ*	*UF*	*CSQ+UF*
PA	166,879	54,870	27,289	–	33,824	33,769
compdists	–	548,720	599,674	28,746	43,757	19,260
Time (s)	0.9	1.7	3.8	111.35	5.645	4.461

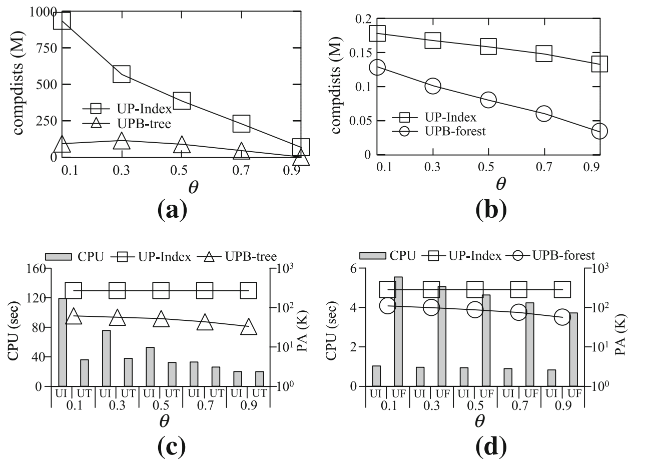

Fig. 20 MPRQU performance versus θ. **a** *Color*, **b** *DNA*, **c** *Color*, **d** *DNA*

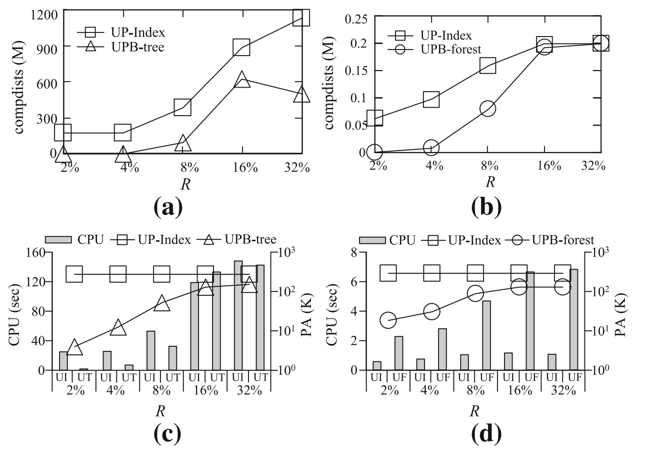

Fig. 21 MPRQU performance versus R. **a** *Color*, **b** *DNA*, **c** *Color*, **d** *DNA*

tions, resulting in weaker pruning ability of Lemma 4.1. Note that, U-Quadtree achieves the smallest CPU time, followed by UPR-tree and then UPB-tree. The reason is that UPR-tree and UPB-tree need additional CPU cost to prune and validate data using the triangle inequality for simple distance computation (e.g., L_2-*norm*), and UPB-tree needs additional CPU cost to perform SFC mapping. The second observation is that CSQ has fewer *compdists* but takes longer than the UPB-forest. The reason is that CSQ utilizes the characteristics of *set* data to prune uncertain sets, leading to fewer distance computations. Nonetheless, the individual and batch pruning techniques used by CSQ can also be integrated easily into our algorithm (denoted as CSQ + UPB), to further improve the query efficiency on uncertain set data, as depicted in Table 6.

7.4 MPRQU performance

The fourth set of experiments verifies the MPRQU performance of the UPB-tree and the UPB-forest, compared with the state-of-the-art UP-Index.

Effect of θ Figure 20 shows the MPRQU performance when varying θ, using *Color* and *DNA*. The first observation is that the UPB-tree- and the UPB-forest-based MPRQU algorithms perform better than the UP-Index-based MPRQU algorithm in terms of *compdists* and *PA*. This is because the UPB-tree- and the UPB-forest-based MPRQU algorithms utilize Lemmas 4.5, 4.6, 5.3, and 5.4 to derive the probability upper and lower bounds, and thus are able to prune and validate uncertain objects without further verifications. However, the CPU time of UP-Index can be smaller than that of our methods. The reason is that the additional CPU cost is used to derive the probability upper and lower bounds. In addition, the query costs (including *compdists*, *PA* and the CPU time) of MPRQU drop with the growth of θ since the pruning power increases with θ.

Effect of R Figure 21 depicts the MPRQU performance under various R values, using *Color* and *DNA*. As expected, the *compdists*, *PA* and the CPU time increase with the growth of R, because the search space grows with R. However, due to Lemma 4.6, the validation power of the UPB-tree increases with R, resulting in fewer distance computations, which makes it possible that the *compdists* drops when R reaches 32% on *Color* and also makes it possible that the CPU time of UPB-tree is less than that of UP-Index on *Color*.

7.5 MPRJ performance

The last set of experiments target the MPRJ performance based on the UPB-tree and the UPB-forest, compared with the baseline algorithms as discussed in Sects. 4.4 and 5.4. Recall that the baseline algorithms perform multiple MPRQU.

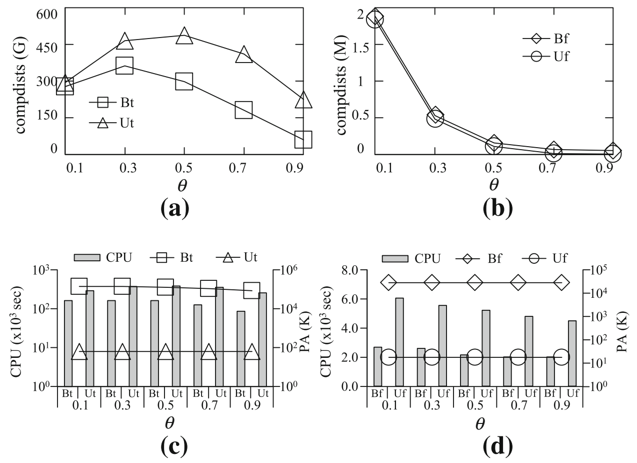

Fig. 22 MPRQJ performance versus θ. **a** *Color*, **b** *SF*, **c** *Color*, **d** *SF*

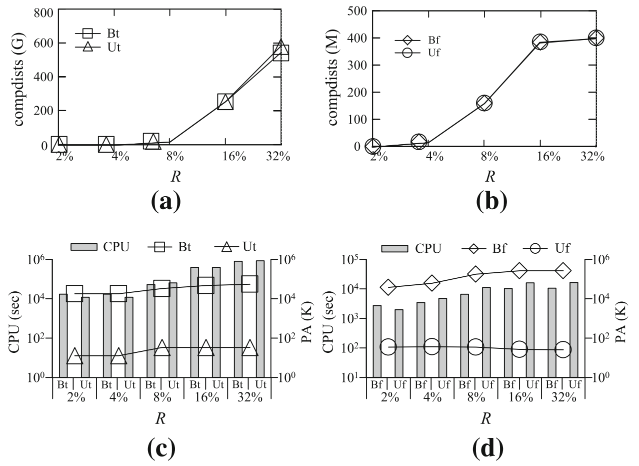

Fig. 23 MPRJ performance versus R. **a** *English*, **b** *DNA*, **c** *English*, **d** *DNA*

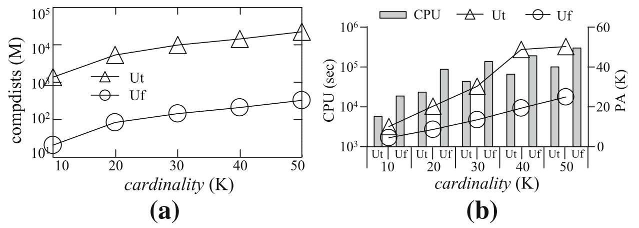

Fig. 24 MPRJ performance versus cardinality. **a** Synthetic (set) and **b** synthetic (set)

Effect of θ Figure 22 plots the MPRJ performance when changing θ, using *Color* and *SF*. As MPRJ is costly and it involves two uncertain datasets, we divide *Color* (20K) and *SF* (20K) into two parts with equal size. Abbreviations (**Ut** for UtMJA, **Bt** for UPB-tree-based baseline algorithm, **Uf** for UfMJA, and **Bf** for UPB-forest-based baseline algorithm) are depicted at the bottom of each column. The first observation is that UtMJA and UfMJA are several orders of magnitude better than the baseline algorithms in terms of page accesses. The reason is that the baseline algorithms have to traverse the datasets multiple times, while UtMJA and UfMJA need to traverse the datasets only once. In addition, Lemmas 4.9, 4.10, 5.5, and 5.6, and Theorem 5.1 are utilized by UtMJA and UfMJA to further accelerate the MPRJ. The second observation is that the query costs of MPRJ drop with the growth of θ on *SF* dataset, while the query costs first increase and then drop on *Color* dataset. This is because the validating power decreases but the pruning power increases with θ. It is worth noting that the *compdists* of UtMJA is larger than that of the baseline algorithm. The reason is that *Z*-order curve used for UtMJA to employ Lemma 4.9 has the weaker space locality performance than Hilbert curve used by the baseline algorithm.

Effect of R Figure 23 shows the MPRJ performance under various R values, using *English* (20K) and *DNA* (40K). As expected, UtMJA and UfMJA are several orders of magnitude better than the baseline algorithms in terms of *PA*. In addition, the query costs increase with R, because the search space grows with R.

Effect of *cardinality* Figure 24 depicts the MPRJ performance as a function of cardinality. Clearly, the query costs grow linearly with the cardinality as the search increases. In addition, the number of distance computations of UPB-tree-based algorithm is much larger than that of UPB-forest-based algorithm, but the CPU time of UPB-tree-based algorithm is smaller, as already discussed in Sect. 7.2.

8 Conclusions

To address the uncertainty in various applications, we introduce an object-level model and a bi-level model, and we propose corresponding indexing structures, i.e., the UPB-tree and the UPB-forest, and present efficient metric probabilistic range query and range join algorithms using these indexes. The resulting findings via extensive experiments on both real and synthetic data sets are summarized as follows.

- Our pivot selection method PSA can select more effective pivots than existing method IPS.
- Compared with the state-of-the-art method UP-Index, our indexes UPB-tree and UPB-forest have *lower* construction costs and metric probabilistic range query costs in terms of distance computations and page accesses, but have *larger* CPU time in some cases. Thus, our indexes perform better than UP-Index when the I/O cost is the dominance cost or the distance function is costly.
- UPB-forest has smaller I/O cost yet higher CPU cost than UPB-tree using the same datasets.
- Our metric probabilistic range join algorithms UtMJA and UfMJA are several orders of magnitude better than baseline algorithms in terms of I/O cost, whereas they achieve similar CPU performance. Hence, our methods

perform better than the baseline algorithms when the I/O cost is the dominance cost.

In the future, it is interesting to extend our indexes to distributed environments. Another promising direction for future work concerns other types of metric probabilistic queries, such as metric probabilistic nearest neighbor search.

Acknowledgements This work was supported in part by the 973 Program of China No. 2015CB352502, the NSFC Grant Nos. 61522208, 61379033, and 61472348, the NSFC-Zhejiang Joint Fund Grant No. U1609217, and a grant from the Obel Family Foundation.

References

1. Agarwal, P.K., Cheng, S.W., Tao, Y., Yi, K.: Indexing uncertain data. In: PODS, pp. 137–146 (2009)
2. Aggarwal, C., Yu, P.: On high dimensional indexing of uncertain data. In: ICDE, pp. 1460–1461 (2008)
3. Angiulli, F., Fassetti, F.: Indexing uncertain data in general metric space. IEEE Trans. Knowl. Data Eng. **24**(9), 1640–1657 (2012)
4. Bohm, C., Kunath, P., Schubert, M.: The Gauss-tree: efficient object identification of probabilistic feature vectors. In: ICDE, article 9 (2006)
5. Bustos, B., Navarro, G., Chavez, E.: Pivot selection techniques for proximity searching in metric spaces. Pattern Recognit. Lett. **24**(14), 2357–2366 (2003)
6. Chen, J., Cheng, R.: Efficient evaluation of imprecise location-dependent queries. In: ICDE, pp. 586–595 (2007)
7. Chen, L., Gao, Y., Li, X., Jensen, C.S., Chen, G.: Efficient metric indexing for similarity search. In: ICDE, pp. 591–602 (2015)
8. Chen, L., Gao, Y., Li, X., Jensen, C.S., Chen, G., Zheng, B.: Indexing metric uncertain data for range queries. In: SIGMOD, pp. 951–965 (2015)
9. Cheng, R., Singh, S., Prabhakar, S., Shah, R., Vitter, J.S., Xia, Y.: Efficient join processing over uncertain data. In: CIKM, pp. 738–747 (2006)
10. Cheng, R., Xia, Y., Prabhakar, S., Shah, R., Vitter, J.S.: Efficient indexing methods for probabilistic threshold queries over uncertain data. In: VLDB, pp. 876–887 (2004)
11. Chung, C.W., Pan, C.H., Liu, C.M.: An effective index for uncertain data. In: IS3C, pp. 482–485 (2014)
12. Ciaccia, P., Patella, M., Zezula, P.: M-tree: an efficient access method for similarity search in metric spaces. In: VLDB, pp. 426–435 (1997)
13. Dai, D., Xie, J., Zhang, H., Dong, J.: Efficient range queries over uncertain strings. In: SSDBM, pp. 75–95 (2012)
14. Dallachiesa, M., Palpanas, T., Ilyas, I.F.: Top-*k* nearest neighbor search in uncertain data series. PVLDB **8**(1), 13–24 (2014)
15. Fredriksson, K., Braithwaite, B.: Quicker similarity joins in metric spaces. In: SISAP, pp. 127–140 (2013)
16. Frentzos, E., Gratsias, K., Theodoridis, Y.: On the effect of location uncertainty in spatial querying. IEEE Trans. Knowl. Data Eng. **21**(3), 366–383 (2008)
17. Gao, M., Jin, C., Wang, W., Lin, X., Zhou, A.: Similarity query processing for probabilistic sets. In: ICDE, pp. 913–924 (2013)
18. Ge, T., Li, Z.: Approximate substring matching over uncertain strings. In: PVLDB vol. 4(11), pp. 772–782 (2011)
19. Hoeffding, W.: Probability inequalities for sums of bounded random variables. J. Am. Stat. Assoc. **58**(301), 13–30 (1963)
20. Jacox, E.H., Samet, H.: Metric space similarity joins. ACM Trans. Database Syst. **33**(2), 7:1–7:38 (2008)
21. Jestes, J., Li, F., Yan, Z., Yi, K.: Probabilistic string similarity joins. In: SIGMOD, pp. 327–338 (2010)
22. Jin, R., Liu, L., Ding, B., Wang, H.: Distance constraint reachability computation in uncertain graphs. In: PVLDB vol. 4(9), pp. 511–562 (2011)
23. Kimura, H., Madden, S., Zdonik, S.B.: UPI: a primary index for uncertain databases. In: PVLDB vol. 3(1), pp. 630–637 (2010)
24. Knight, A., Yu, Q., Rege, M.: Efficient range query processing on complicated uncertain data. In: Ozyer, T., Kianmehr, K., Tan, M., Zeng, J. (eds.) Information Reuse and Integration in Academia and Industry, pp. 51–72. Springer, Vienna (2013)
25. Kriegel, H.P., Bernecker, T., Renz, M., Zuefle, A.: Probabilistic join queries in uncertain databases. In: Aggarwal, C. C. (ed.) Managing and Mining Uncertain Data, pp. 257–298. Springer, New York (2009)
26. Kriegel, H.P., Kunath, P., Pfeifle, M., Renz, M.: Probabilistic similarity join on uncertain data. In: DASFAA, pp. 295–309 (2006)
27. Lian, X., Chen, L.: A generic framework for handling uncertain data with local correlations. In: PVLDB, vol. 4(1), pp. 12–21 (2010)
28. Lian, X., Chen, L.: Set similarity join on probabilistic data. In: PVLDB, vol. 3(1), pp. 650–659 (2010)
29. Mao, R., Mirankerb, W.L., Mirankerc, D.P.: Pivot selection: dimension reduction for distance-based indexing. J. Discrete Algorithms **13**, 32–46 (2012)
30. Novak, D., Batko, M., Zezula, P.: Metric index: an efficient and scalable solution for precise and approximate similarity search. Inf. Syst. **36**(4), 721–723 (2011)
31. Paredes, R., Reyes, N.: Solving similarity joins and range queries in metric spaces with the list of twin clusters. J. Discrete Algorithms **7**(1), 18–35 (2009)
32. Pearson, S.S., Silva, Y.N.: Index-based R-S similarity joins. In: SISAP, pp. 106–112 (2014)
33. Sarma, A.D., He, Y., Chaudhuri, S.: Clusterjoin: a similarity joins framework using map-reduce. In: PVLDB, vol. 7(12), pp. 1059–1070 (2014)
34. Silva, Y.N., Aref, W.G., Ali, M.H.: The similarity join database operator. In: ICDE, pp. 892–903 (2010)
35. Silva, Y.N., Pearson, S.: Exploiting database similarity joins for metric spaces. In: PVLDB, vol. 5(12), pp. 1922–1925 (2012)
36. Singh, S., Mayfield, C., Prabhakar, S., Shah, R., Hambrusch, S.E.: Indexing uncertain categorical data. In: ICDE, pp. 616–625 (2007)
37. Skopal, T., Pokorny, J., Snasel, V.: PM-tree: pivoting metric tree for similarity search in multimedia databases. In: ADBIS, pp. 803–815 (2004)
38. Tao, Y., Xiao, X., Cheng, R.: Range search on multidimensional uncertain data. ACM Trans. Database Syst. **32**(3), 15:1–15:54 (2007)
39. Traina Jr, C., Traina, A.J.M., Seeger, B., Faloutsos, C.: Slim-trees: high performance metric trees minimizing overlap between nodes. In: ICDE, pp. 51–65 (2000)
40. Traina Jr, C., Filho, R.F.S., Traina, A.J.M., Vieira, M.R., Faloutsos, C.: The omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient. VLDB J. **16**(4), 483–505 (2007)
41. Vidal, E.: An algorithm for finding nearest neighbors in (approximately) constant average time. Pattern Recognit. Lett. **4**(3), 145–157 (1986)
42. Wang, Y., Metwally, A., Parthasarathy, S.: Scalable all-pairs similarity search in metric spaces. In: KDD, pp. 829–837 (2013)
43. Zhang, Y., Lin, X., Zhang, W., Wang, J., Lin, Q.: Effectively indexing the uncertain space. IEEE Trans. Knowl. Data Eng. **22**(9), 1247–1261 (2010)
44. Zhang, Y., Zhang, W., Lin, Q., Lin, X.: Effectively indexing the multi-dimensional uncertain objects for range searching. In: EDBT, pp. 504–515 (2012)
45. Zhu, R., Wang, B., Wang, G.: Indexing uncertain data for supporting range queries. In: WAIM, pp. 72–83 (2014)