Geoinformatica (2017) 21:797–827
DOI 10.1007/s10707-017-0293-2

Uncertain Voronoi cell computation based on space decomposition

Klaus Arthur Schmid[1] · Andreas Zufle[2] · Tobias Emrich[1] · Matthias Renz[2] · Reynold Cheng[3]

Received: 29 April 2016 / Revised: 2 November 2016 / Accepted: 10 February 2017 /
Published online: 28 February 2017

Abstract To facilitate (k)-Nearest Neighbor queries, the concept of Voronoi decomposition is widely used. In this work, we propose solutions to extend the concept of Voronoi-cells to uncertain data. Due to data uncertainty, the location, the shape and the extent of a Voronoi cell are random variables. To facilitate reliable query processing despite the presence of uncertainty, we employ the concept of *possible-Voronoi cells* and introduce the novel concept of *guaranteed-Voronoi cells*: The possible-Voronoi cell of an object U consists of all points in space that have a non-zero probability of having U as their nearest-neighbor; and the guaranteed-Voronoi cell, which consists of all points in space which must have U as their nearest-neighbor. Since exact computation of both types of Voronoi cells is computationally hard, we propose approximate solutions. Therefore, we employ hierarchical access methods for both data and object space. Our proposed algorithm descends both index structures simultaneously, constantly trying to prune branches in both trees by employing the concept of spatial domination. To support (k)-Nearest Neighbor queries having $k > 1$, this work further pioneers solutions towards the computation of higher-order possible and higher-order

✉ Klaus Arthur Schmid
schmid@dbs.ifi.lmu.de

Andreas Zufle
azufle@gmu.edu

Tobias Emrich
emrich@dbs.ifi.lmu.de

Matthias Renz
mrenz@gmu.edu

Reynold Cheng
ckcheng@cs.hku.hk

[1] Institute for Informatics, Ludwig-Maximilians-Universität München, München, Germany

[2] George Mason University, Fairfax, VA, USA

[3] Department of Computer Science, University of Hong Kong, Pok Fu Lam, Hong Kong

guaranteed Voronoi cells, which consist of all points in space which may (respectively must) have U as one of their k-nearest neighbors. For this purpose, we develop three algorithms to explore our index structures and show that the approach that descends both index structures in parallel yields the fastest query processing times. Our experiments show that we are able to approximate uncertain Voronoi cells of any order much more effectively than the state-of-the-art while improving run-time performance. Since our approach is the first to compute guaranteed-Voronoi cells and higher order (possible and guaranteed) Voronoi cells, we extend the existing state-of-the-art solutions to these concepts, in order to allow a fair experimental evaluation.

Keywords Uncertain data · kNN query · Nearest neighbor query · Voronoi cells · Voronoi decomposition

1 Introduction

The extensive use of social media, s.a. smartphones, and social networks produce a huge flood of geo-spatial and geo-spatio-temporal data. This data allows to assess information about the current positions of mobile entities, such as friends in social networks, unoccupied cabs in a taxi or ride-sharing application, or the current position of users in augmented reality games. However, our ability to unearth valuable knowledge from large sets of spatial and spatio-temporal data is often impaired by the quality of the data.

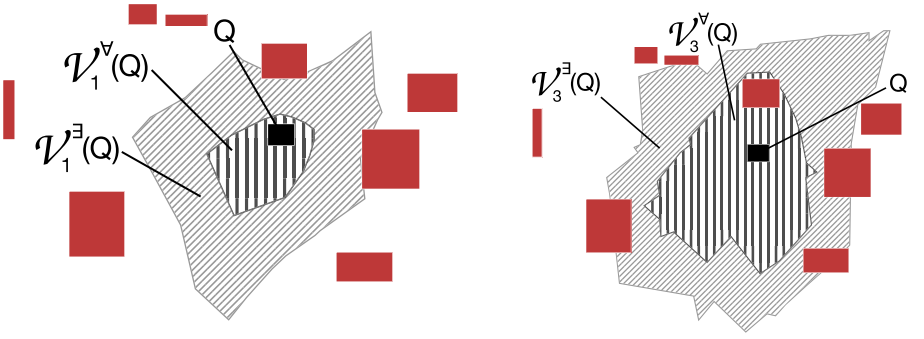
- Data may be imprecise, due to measurement errors, for instance in applications using sensor measurements such as location-based services.
- Data records can be obsolete. For example, ties of friendship bind and break over time, without necessarily reflecting such changes in a social network; in location-based services, users may update their location infrequently, due to bad connectivity or to preserve battery.
- Data can be obtained from unreliable sources, such as crowd-sourcing applications, where data is obtained from individual users, which may incur inaccurate or plain wrong data, deliberately or due to human error.
- To prevent privacy threats and to protect user anonymity, users often consent to relay just a cloaked indication of their whereabouts [1] abstracted as an *uncertainty region* enclosing (but apparently not centered at) their current position.

Simply ignoring these notions of imprecise, obsolete, unreliable and cloaked data, thus pretending that the data is accurate, current, reliable and correct is a common source of false decision making. The research challenge in handling uncertainty in spatial and spatio-temporal data is to obtain reliable results despite the presence of uncertainty. In this work, we revisit the problem of reliably answering nearest-neighbor queries in uncertain data. The problem of finding the closest uncertain object, which has applications such as ride-sharing, has gained much attention in recent years [2–5]. Following a common approach in uncertain data management, these approaches assume that uncertain objects are represented by rectangular or circular uncertainty regions, which are guaranteed to enclose the true (but unknown) position of the respective spatial objects. Following the approach of [6], we carry the concept of Voronoi cells to uncertain data. The idea of [6] is to approximate the *possible Voronoi cell* $\mathcal{V}^{\exists}(O)$ of an object O, which is defined as the space where a query point q can possibly have O as its nearest neighbor, i.e., there exists such a possible world. We leverage this work by defining the notion of the *guaranteed Voronoi cell* $\mathcal{V}^{\forall}(O)$, which we

define as the space for which we can guarantee, that any query point q in that space must have the uncertain object O as their nearest neighbor in any possible world. Applications for possible Voronoi cells include geo-location-based services, such as ride-sharing: In a ride sharing application like Uber and Lyft, we have customers and drivers. A customer can request a ride, and the closest drivers will be asked to accept this ride. From the perspective of a driver that is looking for a customer, the goal is to strategically position themselves in a way such that the area having this rider as one of their nearest-drivers is maximized. In this application, the possible Voronoi cell (guaranteed Voronoi cell) of an individual driver d covers the space of a city where customers may possibly (must certainly) have c as their nearest driver. Our solutions allow an individual driver to see these regions, and furthermore allow him to query nearby locations to maximize his influence by moving to locations having large Voronoi cells.

In such an application, as we see in taxi-GPS data sets such as the T-drive dataset [7, 8], the GPS position $c(t)$ of a cab c at a time t may be highly obsolete, due to infrequent GPS updates. Models to infer the uncertainty region of a mobile object on a road network given past observations have been given in the literature [9]. Furthermore, our approach is the first one to allow an efficient and effective approximation of a higher-order possible or guaranteed Voronoi cell. The k'th-order possible (guaranteed) Voronoi cell of an uncertain object O is defined as the subset of space where a query object q in that space may possible (must certainly) have O as one of it's k-nearest neighbors. Our concept of k'th-order possible (guaranteed) Voronoi cells, allows to leverage geoinformation systems that require the computation of k-nearest neighbors by allowing to efficiently find guaranteed and possible k-nearest neighbors in the presence of uncertain data. For example, applications to notify your k-nearest-friends based on obsolete and thus uncertain location data, will benefit from our approach by allowing to assess which of your friends are guaranteed to be in the result set, which of them are candidates, and which cannot be your nearest neighbors.

To illustrate our notion of uncertain Voronoi cells, consider Fig. 1, where rectangles correspond to the uncertainty regions of objects. For a query object Q, the large shaded region in Fig. 1a depicts the subspace $\mathcal{V}_1^{\exists}(Q)$ for which it holds that any point $p \in \mathcal{V}_1^{\exists}(Q)$ may possibly have object Q as its nearest neighbor. We define this region as the *possible Voronoi cell* of Q. In addition, Fig. 1a further shows the region $\mathcal{V}_1^{\forall}(Q)$, containing all the points in space for which we can guarantee that any such point $p \in \mathcal{V}_1^{\forall}(Q)$ must have Q as



(a) Inner and outer cell for $k = 1$. (b) Inner and outer cell for $k = 3$.

Fig. 1 Uncertain Voronoi cells

their nearest-neighbor, regardless of the exact position of Q, and regardless of the positions of all other uncertain objects. The latter region is called the *guaranteed Voronoi cell* of Q.

Additionally, Fig. 1b depicts the uncertain Voronoi cells of order three for the same object Q. More specifically, the large shaded region denoted by $\mathcal{V}_3^{\exists}(Q)$ corresponds to all points in space such that for any such point $p \in \mathcal{V}_3^{\exists}(Q)$ there must no more than two database objects which are certainly closer to p than to Q. In order words, but equivalently, point p may possibly have the uncertain object Q as one of its 3-nearest-neighbors, that is, there exists a possible world of object locations such that Q is a 3-nearest-neighbors of p. Analogously to these concepts, the region $\mathcal{V}_3^{\forall}(Q)$ contains all points p such that there must be less than two uncertain database objects closer to p than Q, thus guaranteeing that p has Q as one of its 3-nearest-neighbors.

Finding the regions $\mathcal{V}_k^{\exists}(Q)$ and $\mathcal{V}_k^{\forall}(Q)$ for an arbitrary value of k is the goal of this paper. This is not a trivial task: The shape of these regions is a non-convex region which is bounded by hyperbolic curves. As explained in [3, 6, 10], an exact construction of $\mathcal{V}(A)$ requires exponential time. For this reason, an approximate technique for deriving the possible Voronoi cell $\mathcal{V}_1^{\exists}(Q)$ was given in [6]. We propose a new solution for this problem, which extends the existing solution of [6] by the following aspects:

- Unlike previous solutions, our approach offers full index support, indexing the object space using an R^*-tree [11] and indexing the data space using a kd-trie [12].
- Rather than approximating the Voronoi cell $\mathcal{V}_1^{\exists}(Q)$ by a single rectangle ([6]), we use a set of kd-trie partitions, which allows much higher approximation quality. This gain in approximation quality not only improves query times, as our experiments show, but can also be used to gain a detailed visual exploration of possible Voronoi cells.
- In contrast to previous solutions presented in [13], we extend our solution to additionally compute the guaranteed Voronoi cell $\mathcal{V}_1^{\forall}(Q)$, a problem which has not been studied in previous literature.
- Another novel extension, we extend our algorithm to find possible Voronoi cells $\mathcal{V}_k^{\exists}(Q)$ and guaranteed Voronoi cells $\mathcal{V}_k^{\forall}(Q)$ of higher order, i.e., for values of k greater than one. This problem has not been studied before either.
- Our experiments further show that our provided index support for both data and space enables the scaling of uncertain Voronoi cell computation to large databases.

2 Related work

The problem of answering nearest neighbor queries on uncertain data generally involves two steps: A filter approach and a refinement step. In the filter step, a (possibly small) set of objects is returned that contains all objects having a non-zero probability of being the result object. In the refinement step, the exact probability of each candidate object is computed. The refinement step is the main research topic of [14–16], showing how to compute exact probabilities of an object to be the nearest neighbor of a query object, given the probability density functions of objects. In contrast, other existing work focuses on the filter step, applying spatial filter steps in order to identify object that are guaranteed to have a zero probability to be the result object [5, 6, 17]. In all of these works, (reverse-) nearest neighbor queries on uncertain data are supported by evaluating spatial domination at query time, in order to identify objects that must (not) be part of the query result. Thus, these *lazy approaches*, have in common that the main computation is performed at query time. In this work, we propose

an *eager approach*: for each database object U, we precompute their possible-Voronoi cell (and their guaranteed-Voronoi cell), i.e., the regions in space which may (and must) have U as one of their k-nearest neighbor. Given these regions, we can reduce a kNN query on uncertain data to a simply polygon intersection test, vastly reducing query times.

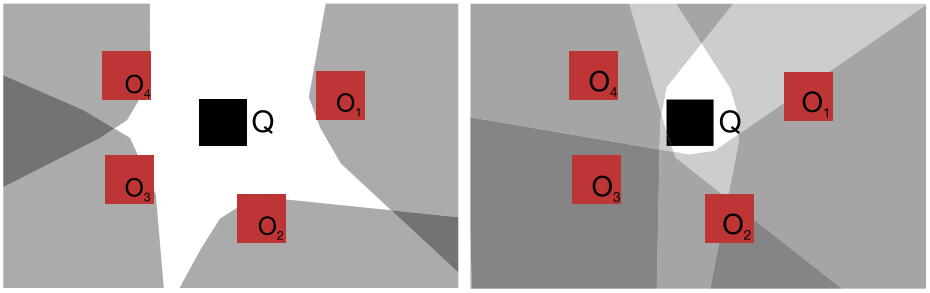
In this work, we focus on the filter step, i.e., the step of finding objects having a non-zero probability to be the nearest neighbor of an object using Voronoi-cells.

The idea of using Voronoi diagrams to answer nearest neighbor (NN) queries over points has been widely studied [18]. In this context, Voronoi diagrams have been used to support nearest neighbor queries in geo-spatial applications [19], location-based services [20, 21], in spatial data streams [22] and in distributed spatial environments [23] as well as in spatial network environments [24]. To support nearest neighbor queries on uncertain data, initial approaches have been presented in [2, 14]. However, in these work, only the database objects are assumed to be uncertain, whereas the query object is assumed to be a point. In [3] a solution to compute possible Voronoi-cells for the case of circular uncertainty regions has been presented. This exact approach has exponential construction and storage cost. Due to this computational drawback, an approximate solution was presented in [6]. The aim of this approach is to approximate the true (but unknown) possible Voronoi-cell $\mathcal{V}(O)$ of an uncertain object O using two rectangle: A single conservative rectangle $h(O)$ which is guaranteed to completely contain $\mathcal{V}(O)$, and a single progressive rectangle $l(O)$ which is guaranteed to be completely contained by $\mathcal{V}(O)$. These two approximation rectangles are obtained by iteratively expanding the progressive rectangle $l(O)$, and iteratively shrinking the conservative rectangle $h(O)$. However, considering examples such as shown in Fig. 1, it is evident that such approximations may be rather inaccurate. Thus, $h(O)$ may cover a large body of space not belonging to $\mathcal{V}(O)$, while $l(O)$ may miss a large body of $\mathcal{V}(O)$, even in the case where $h(O)$ is the smallest conservative bounding rectangle and $l(O)$ is the largest progressive bounded rectangle.[1] Furthermore, an approach for nearest neighbor search on moving uncertain objects has been presented in [4]. A problem common to [3] and [4] is that their solutions are customized for 2D data, making extensive use of intersection and rotation operations of 2D hyperbolic curves. Our approach, as well as the approach of [6] is applicable to arbitrary dimensionality. In comparison to [6], the main contribution of this work is that we can accurately approximate an arbitrarily shaped possible Voronoi-cell, rather than using a single rectangular approximation only. This allows to answer nearest-neighbor queries more efficiently, since less candidates have to be checked, and it allows to more precisely illustrate the Voronoi-region of an uncertain object. Finally, a first solution to compute the first-order possible Voronoi cell $\mathcal{V}_1^{\exists}(U)$ has been presented in [13]. This work extends the solution of [13] to the concept of guaranteed-Voronoi cells $\mathcal{V}_1^{\forall}(U)$. Further, this work extends both concepts $\mathcal{V}_1^{\exists}(U)$ and $\mathcal{V}_1^{\forall}(U)$ to higher-order Voronoi cells $\mathcal{V}_k^{\exists}(U)$ and $\mathcal{V}_k^{\forall}(U)$, thus allowing to employ these concepts for $k > 1$-nearest neighbor queries. These novel concepts and notions will be formally defined in the following section.

3 Problem definition

Figure 2a shows how the possible Voronoi cell $\mathcal{V}_1^{\exists}(Q)$ of an uncertain object Q is defined. For each of the anonymous database objects of Fig. 2, each shaded region corresponds to the

[1] The later case can not be guaranteed by the approach of [6] due to the numeric nature of their approach.



(a) $V_1^{\exists}$ Pruning Regions (b) $V_1^{\forall}$ Pruning Regions

Fig. 2 Spatially dominated regions of objects surrounding Q

pruning region $S_O(Q)$, i.e., the smallest region such that for any $p \in S_O(Q)$, the corresponding object O must be closer to p than Q. Formally,

Definition 1 (**Nearest Neighbor Pruning Region**) Let $\mathcal{D} = \{O_1, ..., O_N\}$ be an uncertain database where each object $O_i \in \mathcal{D}$ is represented by a rectangular uncertainty region in $\mathcal{R}^d$. Let $dist(.,.)$ denote any L_p norm.[2] For any $A, B \in \mathcal{D}$, we define the nearest neighbor pruning region where any point must be closer to A than to B as follows:

$$S_A(B) := \{q \in \mathcal{R}^d : \text{maxDist}(q, A) < \text{minDist}(q, B)\},$$

where maxDist(q, A) and minDist(q, B) denote the maximum and minimum distance between a point q and a rectangle A or B, respectively, as defined in [25].

Figure 2a shows four nearest neighbor pruning regions $S_{O_1}(Q), ..., S_{O_4}(Q)$ as shaded regions. Any region implies, the the respective object O_1 must be closer to this region than Q. Using Definition 1, we can now define the possible Voronoi cell $\mathcal{V}_1^{\exists}(Q)$ of an object Q as the space that does not intersect any nearest neighbor pruning region $S_O(Q)$ of a database object O. Formally:

Definition 2 (**Possible Voronoi Cell**) Let $Q \in \mathcal{D}$ be an uncertain object. Then the possible Voronoi cell $\mathcal{V}_1^{\exists}(Q)$ is defined as

$$\mathcal{V}_1^{\exists}(Q) = \mathcal{R}^d \setminus \bigcup_{O \in \mathcal{D} \setminus \{Q\}} S_O(Q).$$

In Fig. 2, the white (i.e., non-shaded) region corresponds to the Voronoi cell $\mathcal{V}_1^{\exists}(Q)$.

Figure 2b depicts shades regions corresponding to the four nearest neighbor pruning regions $S_Q(O_1), ..., S_Q(O_4)$ as shaded regions. Again following Definition 1, any such region $S_Q(O)$ corresponds to the region that is guaranteed to be closer to Q than to O. This observation allows us to define the guaranteed Voronoi cell $\mathcal{V}_1^{\forall}(Q)$ as the space intersection of all these regions.

[2] We use Euclidean distance in all examples and illustrations, but any L_p norm can be applied.

Definition 3 (**Guaranteed Voronoi Cell**) Let $Q \in \mathcal{D}$ be an uncertain object. Then the guaranteed Voronoi cell $\mathcal{V}_1^{\forall}(Q)$ is defined as

$$\mathcal{V}_1^{\forall}(Q) = \bigcap_{O \in \mathcal{D} \setminus \{Q\}} S_Q(O).$$

The challenge of this work is to accurately approximate both uncertain Voronoi regions $\mathcal{V}_1^{\forall}(Q)$ and $\mathcal{V}_1^{\exists}(Q)$ efficiently.

4 Spatial domination on rectangles revisited

The concept of spatial domination and efficient techniques to verify it were introduced in [26]. Spatial domination describes the spatial relation of three rectangles to each other. Since the spatial domination can also be utilized for the computation of uncertain Voronoi cells, we briefly want to review the concept. Notations used throughout this paper are explained in Table 1.

Definition 4 (**Spatial Domination**) Let $A, B, R \subseteq \mathcal{R}^d$ be rectangles in a d-dimensional space and $dist()$ be a distance function defined on that space. The rectangle A dominates B w.r.t. R iff for all points $r \in R$ it holds that every point $a \in A$ is closer to r than any point $b \in B$, i.e.

$$Dom(A, B, R) \Leftrightarrow \forall r \in R, \forall a \in A, \forall b \in B : dist(a, r) < dist(b, r)$$

Informally speaking, $Dom(A, B, R)$ is thus true if A is "certainly" closer to R than B. In addition the concept of partial spatial domination was introduced.

Definition 5 (**Partial Spatial Domination**) Let $A, B, R \subseteq \mathcal{R}^d$ be rectangles in a d-dimensional space and $dist()$ be a distance function defined on that space. The rectangle A

Table 1 Table of notations

Notation	Meaning	Notation	Meaning
$\mathcal{D}$	Uncertain Object Database	$O, Q, U \in \mathcal{D}$	uncertain objects
$\mathcal{I}_{\mathcal{D}}$	Hierarchical Data Index	$\mathcal{I}_{\mathcal{S}}$	Hierarchical Space Index
$\mathcal{G}$	d-dimensional grid	$g_i \in \mathcal{G}$	Rectangular Grid Cell
$\mathcal{V}_k^{\exists}(U)$	k-th order possible Voronoi cell of U		
$\mathcal{V}_k^{\forall}(U)$	k-th order guaranteed Voronoi cell of U		
$S_A(B) \subseteq \mathcal{R}^d$	The region where object A dominates object B		
$Dom(A, B, R)$	Predicate that is true iff rectangle R is fully contained $S_A(B)$.		
	Can be evaluated efficiently [26].		
$PDom(A, B, R)$	Predicate that is true iff rectangle R intersects $S_A(B)$.		
	Can be evaluated efficiently [26].		
$h \subseteq \mathcal{R}^d$	Rectangular Space Index Entry obtained from $\mathcal{I}_{\mathcal{S}}$:		
	Partition of Space for which we want to decide if it belongs to $\mathcal{V}(U)$		
$e \subseteq \mathcal{R}^d$	Rectangular Data Index Entry obtained from $\mathcal{I}_{\mathcal{D}}$:		
	Spatial approximation of a set of data objects if e is non-leaf entry,		
	Uncertainty region of a single data object if e is a leaf entry.		

dominates B partially w.r.t. R , denoted by PDom(A, B, R) if A dominates B for some, but not all $r \in R$, i.e.

$$PDom(A, B, R) \Leftrightarrow (\exists r \in R : \forall a \in A, \forall b \in B : dist(a, r) < dist(b, r)) \wedge$$
$$(\exists r \in R : (\exists a \in A, \exists b \in B : dist(a, r) \leq dist(b, r)) \wedge$$
$$(\exists a \in A, \exists b \in B : dist(a, r) \geq dist(b, r))).$$

In [5] it was shown that spatial domination can be utilized when the rectangles conservatively approximate uncertain objects. In this case Dom(A, B, R) means P("R is closer to A than to B") = 1 and PDom(A, B, R) means $0 \leq$ P("R is closer to A than to B") ≤ 1. Using the Dom()- and the PDom()-function it is thus possible to decide the location of a rectangle w.r.t. the uncertain bisector of two uncertain objects. The uncertain bisector between two uncertain objects A and B (conservatively approximated by rectangles) defines three spaces: In $S_A(B) = \{s \in \mathcal{S} : Dom(A, B, \{s\})\}$ all objects are certainly closer to A than to B, in $S_B(A) = \{s \in \mathcal{S} : Dom(B, A, \{s\})\}$ object are certainly closer to B than to A and in the space in between no certain decision can be made. This relation is shown in Fig. 3. We are thus able to decide where the rectangle R is located w.r.t. the bisector $S_B(A)$ and $S_A(B)$ of A and B respectively by performing the Dom() and the PDom() function [26]. The following six cases are defined using a function DomCase(A, B, R) as follows.

Definition 6 (**Domination Cases**) Let A and B be rectangles. For any rectangle R, one of the following cases holds:

Case 1: R is fully contained in $S_A(B)$ iff $Dom(A, B, R)$;
Case 2: R intersects $S_A(B)$ but not $S_B(A)$ iff $PDom(A, B, R) \wedge \neg PDom(B, A, R)$;
Case 3: R intersects neither $S_A(B)$ nor $S_B(A)$ iff $\neg Dom(A, B, R) \wedge \neg PDom(A, B, R) \wedge \neg PDom(B, A, R) \neg Dom(B, A, R)$;
Case 4: R intersects $S(B)$ but not $S(A)$ iff $\neg PDom(A, B, R) \wedge PDom(B, A, R)$;
Case 5: R is fully contained in $S(B)$ iff $Dom(B, A, R)$;
Case 6: R intersects both $S(A)$ and $S(B)$ iff $PDom(A, B, R) \wedge PDom(B, A, R)$;

Figure 3 depicts all possible cases. Here, each rectangle R_i corresponds to Case i in Definition 6. Note that the materialization of the pruning regions $S_A(B)$ and $S_B(A)$ is a hard problem [6]. Nevertheless, the function DomCase(A, B, R) allows to efficiently decide

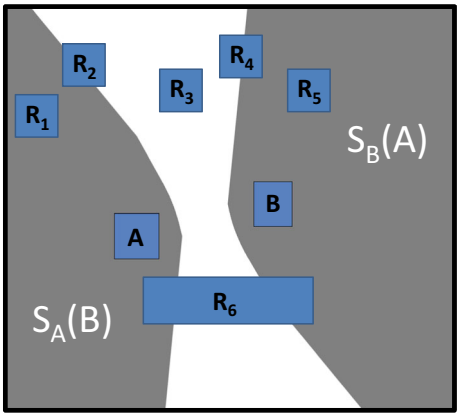


Fig. 3 Domination relation

between the six possible domination cases defined above. In the next section we will show how to use these relations in order to compute uncertain Voronoi cells.

5 Possible and guaranteed Voronoi cell approximation

5.1 Naive solution

Computing uncertain Voronoi cells is a daunting task for two reasons: First, it is challenging to find the objects in the database that have an effect on its shape. Second, the representation of the cell is hard since it consists of many linear and parabolic parts that grow exponentially with the dimensionality. This section will present four algorithms that apply the concept of spatial domination to efficiently approximate the possible-Voronoi cell $\mathcal{V}_k^{\exists}(U)$ and the guaranteed-Voronoi cell $\mathcal{V}_k^{\forall}(U)$ of an object U as tight as possible. The first, naive, algorithm divides the space into equi-distant grid cells and labels the cells according to their membership to the possible-Voronoi cell. The second algorithm, additionally uses an R*-tree to index the data objects to avoid exploration of irrelevant objects. The third algorithm uses a kd-trie to index the grid cells, in order to identify large regions of space for which their relation to $\mathcal{V}_k^{\exists}(U)$ and $\mathcal{V}_k^{\forall}(U)$ can be decided without exploring smaller subregions. The fourth algorithm uses both a kd-trie to index the space and an R-tree to index the data. For the later algorithm, the main challenge is to smartly descend both hierarchical index structures in parallel, to minimize the computational overhead.

A straightforward way of computing $\mathcal{V}_k^{\exists}(U)$ and $\mathcal{V}_k^{\forall}(U)$ is to apply an equi-distant d-dimensional grid to partition the data space. For each cell g we decide weather it belongs to $\mathcal{V}_k^{\exists}(U)$, to $\mathcal{V}_k^{\forall}(U)$, or to neither.

Algorithm The algorithm takes as input the target object U, an uncertain database $\mathcal{D}$ and a grid $\mathcal{G}$ covering the space of $\mathcal{D}$. We iterate over all grid cells $g \in \mathcal{G}$ and in order to decide whether g is part of the UV cell of U. Therefore, domination against all objects $O \in \mathcal{D} \setminus U$ has to be checked. All possible cases of domination of a grid-cell g are depicted in Fig. 4. We can decide the following cases for each grid cell g: (i) g is completely outside of $\mathcal{V}_k^{\exists}(U)$ or (ii) g is a boarder cell intersecting the edge of $\mathcal{V}_k^{\exists}(U)$, or (iii) g is completely inside $\mathcal{V}_k^{\exists}(U)$ but completely outside $\mathcal{V}_k^{\forall}(U)$, or (iv) g is a boarder cell touching the edge of $\mathcal{V}_k^{\forall}(U)$, or

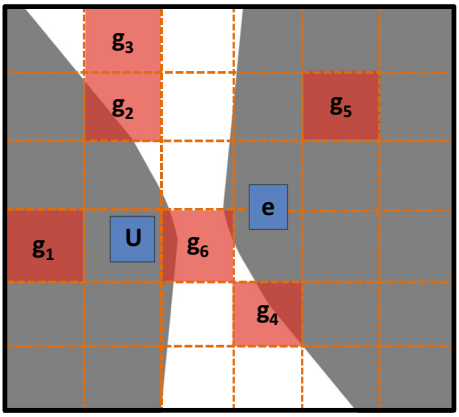


Fig. 4 Cases of domination of a grid cell

(v) g is completely inside $\mathcal{V}_k^{\forall}(U)$. To make this decision, we can exploit the six cases of Definition 6. In the following, the operator $\exists_k O \in DB : \varphi$ denotes that the predicate φ holds for at least k objects in $\mathcal{D}$. Formally, our first algorithm proceeds as follows:

i) If $\exists_k O \in \mathcal{D} \setminus U : Dom(O, U, g)$ then g is neither part of $\mathcal{V}_k^{\exists}(U)$ nor $\mathcal{V}_k^{\forall}(U)$, since at least k database object dominate g. This domination corresponds to **Case 5** of Definition 6 and cell g_5 in Fig. 4.

ii) Otherwise, if $\exists_k O \in \mathcal{D} : PDom(O, U, g)$ then at least a small part of g must be part of $\mathcal{V}_k^{\exists}(U)$. This case corresponds to the cases of cells g_4 and g_6 in Fig. 4, i.e., **Case 4** or **Case 6** of Definition 6.

iii) Otherwise we can conclude that g is completely contained in $\mathcal{V}_k^{\exists}(U)$. Since for all but at most $k-1$ database objects, it holds that g corresponds to one of the remaining cases **Case 1**, **Case 2** and **Case 3** of cells g_1, g_2 or g_3, respectively, as shown in Fig. 4. If $\exists_k O \in \mathcal{D} : \neg PDOM(U, O, g)$, i.e., if there exist k objects that are not dominated by g, then g is completely outside of $\mathcal{V}_k^{\exists}(U)$. The case $PDOM(U, O, g)$ corresponds to **Case 1**, **Case 2** and **Case 6** of Definition 6 and cell g_5 in Fig. 4.

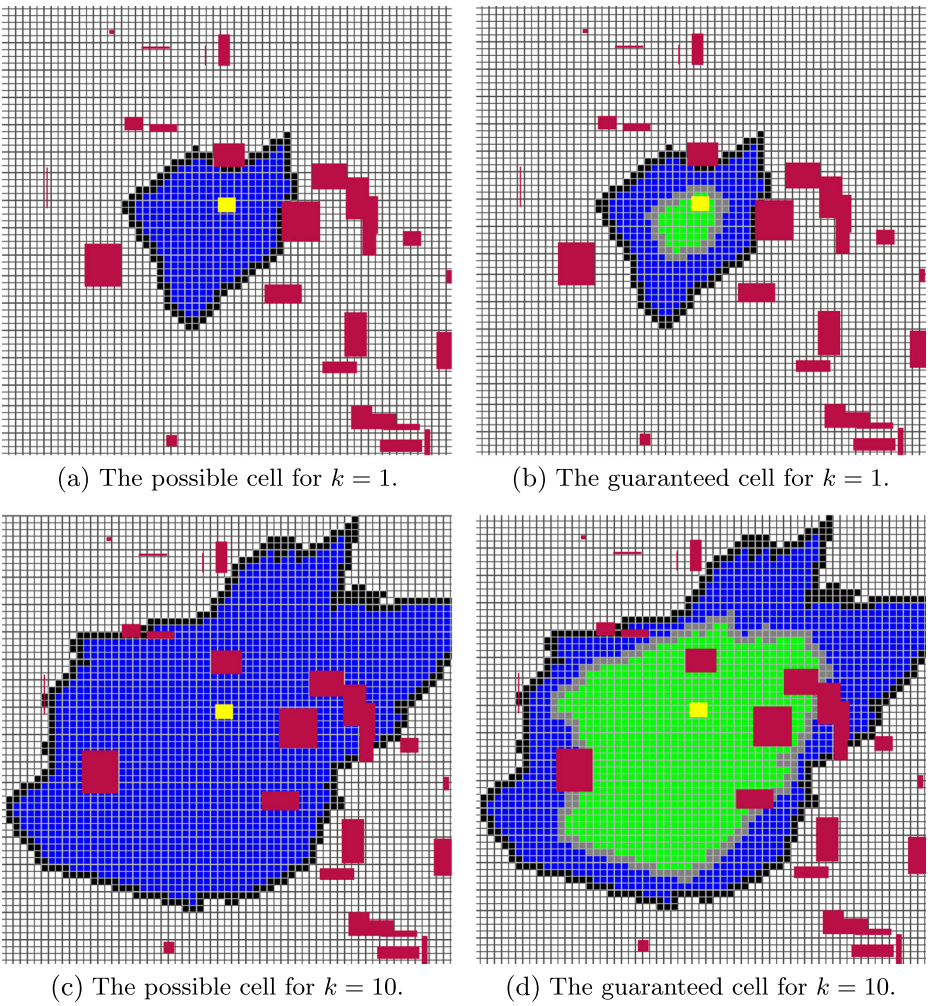
iv) Otherwise, we can guarantee that g intersects $\mathcal{V}_k^{\forall}(U)$. If $\exists_k O \in \mathcal{D} : \neg DOM(U, O, g)$, then k objects may have a chance to dominate g, such that g cannot be fully contained in $\mathcal{V}_k^{\forall}(U)$. The case $DOM(U, O, g)$ corresponds solely to **Case 1** in Definition 6.

v) Otherwise, we can conclude that $DOM(U, O, g)$ holds for all but $k-1$ database objects, such that any point in g is guaranteed to have U as its k-nearest neighbor. Thus g must be completely contained in $\mathcal{V}_k^{\forall}(U)$.

The set of all grid cells satisfying v) define a lower bound of $\mathcal{V}_k^{\forall}(U)$, and the grid cells satisfying iv) or v) define an upper bound of $\mathcal{V}_k^{\forall}(U)$. Analogously, all grids cells satisfying iii-v) define a lower bound of $\mathcal{V}_k^{\exists}(U)$ and all grid cells satisfying ii-v) define an upper bound of $\mathcal{V}_k^{\exists}(U)$.

For a small database of uncertain objects an exemplary result of this approach is depicted in Fig. 5. The object U for which the possible and guaranteed Voronoi cells are computed is highlighted in yellow. Furthermore, a space grid is shown, where (i) unfilled space cells are guaranteed to be outside of $\mathcal{V}_k^{\exists}(U)$ (and thus outside of $\mathcal{V}_k^{\forall}(U)$), (ii) black cells are guaranteed to be on the border of $\mathcal{V}_k^{\exists}(U)$, (iii) blue cells are guaranteed to be inside $\mathcal{V}_k^{\exists}(U)$, but outside of $\mathcal{V}_k^{\forall}(U)$, (iv) grey cells are on the boarder of $\mathcal{V}_k^{\forall}(U)$, and green cells are located inside $\mathcal{V}_k^{\forall}(U)$. In particular, Fig. 5a shows only the possible Voronoi cell $\mathcal{V}_1^{\exists}(U)$, for the computation of which previous solutions have been presented [6, 13] in the literature. In addition, Fig. 5 illustrates the new concepts presents in this work:

- the novel concept of a guaranteed Voronoi-cell $\mathcal{V}_1^{\forall}(U)$, which is additionally shown in Fig. 5b, denoting the space for which any point is guaranteed to have uncertain object U as its nearest neighbor,
- the novel concept of a higher order possible-Voronoi cell $\mathcal{V}_{k>1}^{\exists}(U)$, depicted in Fig. 5c. Here, the 10th order possible-Voronoi cell $\mathcal{V}_{10}^{\exists}(U)$ is shown. Thus, any space cell highlighted in blue in Fig. 5c has a non-zero probability to have the yellow object U as one of its ten nearest neighbors. Note that, for illustration purpose, only a small part of the database is shown in Fig. 5c and some of the objects responsible for the shape of $\mathcal{V}_{10}^{\exists}(U)$ are not shown,
- and the combination of both new concepts, which creates the notion of a higher-order guaranteed Voronoi cell $\mathcal{V}_{k>1}^{\forall}(U)$, which is illustrated in Fig. 5d.



(a) The possible cell for $k = 1$. (b) The guaranteed cell for $k = 1$.

(c) The possible cell for $k = 10$. (d) The guaranteed cell for $k = 10$.

Fig. 5 Uncertain Voronoi cells

The naive solution presented in this section allows to compute all of the above solutions, i.e., the possible Voronoi cell $\mathcal{V}_k^{\exists}$ and the guaranteed Voronoi cell $\mathcal{V}_k^{\forall}$. However, this solution requires to individually consider each space cell of a space grid, paired with each database object, to compute the respective domination cases. Clearly, the resulting complexity, linear in the number of space cells and linear in the number of database objects, is inapplicable to large data sets. The next sections improve this computational complexity by employing index structures for both the object space and the data objects. In Section 6, we employ an aggregate R-tree to index uncertain objects. In particular, we show how we can use a higher level R-tree entry e for the computation of $\mathcal{V}_k^{\exists}$ and $\mathcal{V}_k^{\forall}$ without having to refine e. Then, in Section 7, we proceed to employ a quad-tree to index the space grid. We show how we can decide, for non-leaf entry h of this quad-tree, how to decide if h must (not) part of $\mathcal{V}_k^{\exists}$ and $\mathcal{V}_k^{\forall}$. Finally, our main contribution is given in Section 8, where we propose an algorithm to compute Voronoi-cells $\mathcal{V}_k^{\exists}$ and $\mathcal{V}_k^{\forall}$ using both the aforementioned index structures. The

main challenge solved in the following sections is to find heuristics to descent both index structures, for space and data, in parallel, in order to minimize the number of domination checks incurred by the algorithm.

6 Indexing $\mathcal{D}$

Obviously, checking an object U against all uncertain objects $O \in \mathcal{D}$ is very expensive. Instead, we can use an MBR based index structure $\mathcal{I}_\mathcal{D}$ (such as an R*-Tree) to organize the objects hierarchically.

Algorithm The algorithm takes as input the target object U, $\mathcal{I}_\mathcal{D}$ and a grid covering the space of $\mathcal{I}_\mathcal{D}$. For each grid cell g the algorithm traverses the entries e of $\mathcal{I}_\mathcal{D}$ in a best first manner [27] according to $MinDist(e, U)$. Note that the entry e can be a single uncertain object (i.e., a leaf-entry) or an intermediate node that conservatively approximates multiple uncertain objects. Since we assume that our data index uses rectangular approximations, we can then apply Definition 6 to decide which index entries have to be accessed. For reference, the following cases are shown in Fig. 4. Keep in mind that in this case, the entries e are data index entries, which may be intermediate entries representing multiple data objects. Intuitively, by refining a data entry e into its children entries, the uncertainty of the domination regions of these children become smaller. Therefore, the dominated regions $S_U(e)$ and $S_e(U)$ (c.f. Figs. 3 and 4) grow larger, and the white region for which no decision can be made grows larger. Consequently, refining an entry e may cause the case of some other space cells to change. Recall that for each cell, we need to decide for one of the five following cases: (i) definitely outside of $\mathcal{V}_k^\exists(U)$, (ii) on the border of $\mathcal{V}_k^\exists(U)$, (iii) inside $\mathcal{V}_k^\exists(U)$ but outside of $\mathcal{V}_k^\forall(U)$, (iv) on the border of $\mathcal{V}_k^\forall(U)$ or (v) completely inside $\mathcal{V}_k^\forall(U)$. As an example, these cases are color-coded by the colors white, black, blue, grey and green, respectively in Fig. 6.

Case 1: $Dom(U, e, g_1)$: e and none of its children can exclude g_1 from either UV-cell $\mathcal{V}_k^\exists(U)$ nor $\mathcal{V}_k^\forall(U)$. Thus refinement of e cannot yield any new information.

Case 2: $PDom(U, e, g_2)$: If e is not a leaf entry, then for some (or even all) of the child nodes e' of e the predicate $Dom(U, e', g_1)$ might hold, thus possibly reducing **Case 2** to **Case 1** for some child nodes of e.

Case 3: $\neg PDom(U, e, g_3) \wedge \neg PDom(e, U, g_3)$: As long as e is not a leaf entry (an object), there might exist a child of e which excludes g_3 from $\mathcal{V}_k^\forall(U)$ (**Case 5**), or for which g_3 is guaranteed to be closer to U (**Case 2**).

Case 4: $PDom(e, U, g_4)$: In this case, if e is not a leaf entry, then for some (or even all) of the child nodes e' of e the predicate $Dom(e', U, g_1)$ might hold, thus possibly reducing **Case 4** to **Case 5** for some child nodes of e.

Case 5: $Dom(e, U, g_5)$: U is dominated by e and thus by all the children of e. Entry e will no longer be refined, and the number of **Case 5** objects is increased by the number of children of e.

Case 6: $PDom(U, e, g_6) \wedge PDom(e, U, g_6)$: In this case, parts of g are dominated by e, for some parts e dominates g, and for some parts no decision can be made. In this case, entry e does no longer need to be refined, any children of e must yield the same case.

Clearly, using the data index $\mathcal{I}_\mathcal{D}$ allows to avoid a large fraction of dominance checks: for a large page regions e that is located far distance from object U, we can already decide

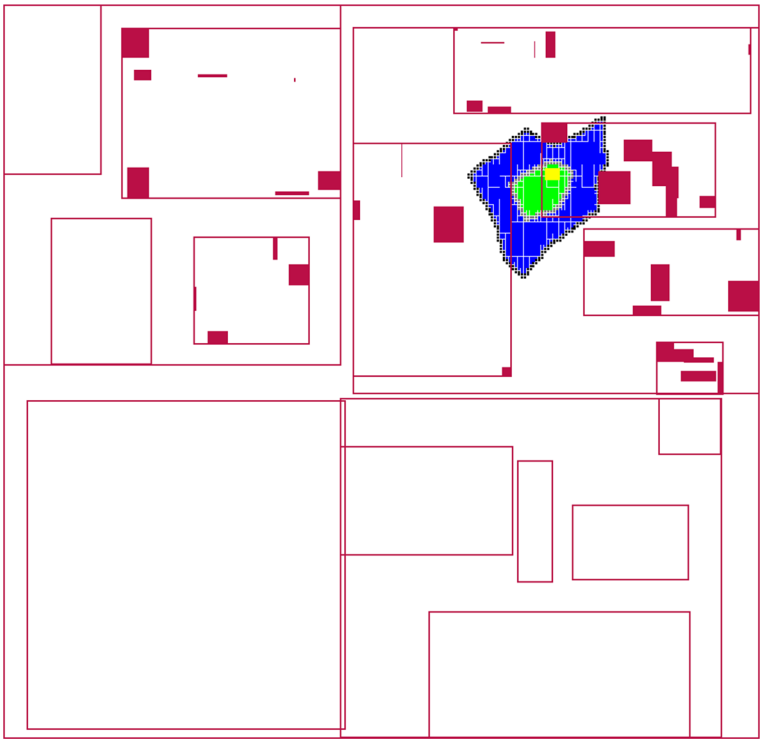


Fig. 6 Refined page regions of $\mathcal{I}_{\mathcal{D}}$ for a small example database

on a high directory level that e and all its children can not influence the shape of $\mathcal{V}_k^{\exists}(U)$ and $\mathcal{V}_k^{\forall}(U)$. To illustrate the effect of employing the data index $\mathcal{I}_{\mathcal{D}}$, a small sample database is given in Fig. 6. In addition to the Voronoi-cells $\mathcal{V}_1^{\exists}(U)$ and $\mathcal{V}_1^{\forall}(U)$, Fig. 6 also shows to what degree the employed data index $\mathcal{I}_{\mathcal{D}}$ has been refined. Solid rectangle correspond to fully refined uncertain objects. Unfilled rectangles correspond to intermediate R-tree index entries. Clearly, the number of fully refined uncertain objects decreases drastically, thus decreasing the overall number of domination checks that need to be computed, and thus decreasing the overall run-time. Our experimental evaluation in Section 9 will give a quantitative analysis of the run-time improvement gained by employing the data index $\mathcal{I}_{\mathcal{D}}$.

Still, the problem remains that each space cell has to be checked individually. Intuitively, it should be possibly to identify for large regions located far away from U, that these cannot be part of $\mathcal{V}_k^{\exists}(U)$ and $\mathcal{V}_k^{\forall}(U)$. Furthermore, it might be possible to decide for large regions close to (and possibly overlap with) U, that they must be part of $\mathcal{V}_k^{\forall}(U)$. And finally, there might be large regions for which we might be able to decide that they must be part of $\mathcal{V}_1^{\exists}(U)$ but not of $\mathcal{V}_1^{\forall}(U)$. In the next section, Section 7 we introduce a space index, which hierarchically structures space cells, allowing us to decide whether a large space region is completely outside of $\mathcal{V}_k^{\exists}(U)$, completely inside $\mathcal{V}_k^{\exists}(U)$ but completely outside $\mathcal{V}_k^{\forall}(U)$, or completely inside $\mathcal{V}_k^{\forall}(U)$. Finally, Section 8 will combine both index structures for data and for space, by showing heuristics to traverse both index structures in an efficient way.

7 Indexing $\mathcal{S}$

Instead of indexing the data objects, we show how to index the space grid cells in this section. We propose to use a tree based index structure (denoted as $\mathcal{I}_\mathcal{S}$ to organize the data space (e.g. Quadtree, kd-trie). For each entry $h \in \mathcal{I}_\mathcal{S}$ it can be checked if it is part of the UV cell of U.

Algorithm The algorithm takes as input the target object U, $\mathcal{I}_\mathcal{S}$, $maxdepth$ and a list of all data objects $O \in \mathcal{D}$. The entries $h \in \mathcal{I}_\mathcal{S}$ are traversed in a depth-first manner. For each entry h we check all $O \in \mathcal{D}$ to decide if the traversal has to go deeper (to the children of h) or its subtree can be discarded for further processing. The parameter $maxdepth$ defines the maximum depth that $\mathcal{I}_\mathcal{S}$ is traversed. Thus the larger $maxdepth$, the finer the granularity of the UV-cell approximation.

We can again distinguish the same cases as in Section 5.1:

i) If $\exists_k O \in \mathcal{D} \setminus U : Dom(O, U, h)$ then h is neither part of $\mathcal{V}_k^\exists(U)$ nor $\mathcal{V}_k^\forall(U)$, since at least k database object dominate h. Thus h no longer needs to be refined (and can be colored in white in Fig. 7).

ii) Otherwise, if $\exists_k O \in \mathcal{D} : PDom(O, U, h)$ then at least a small part of h must be part of $\mathcal{V}_k^\exists(U)$. If h is a directory entry (i.e., an entry having a depth less than *maxdepth*), then h is refined. Otherwise, h is a leaf entry (i.e., a space entry of depth *maxdepth* and is is colored black in Fig. 7).

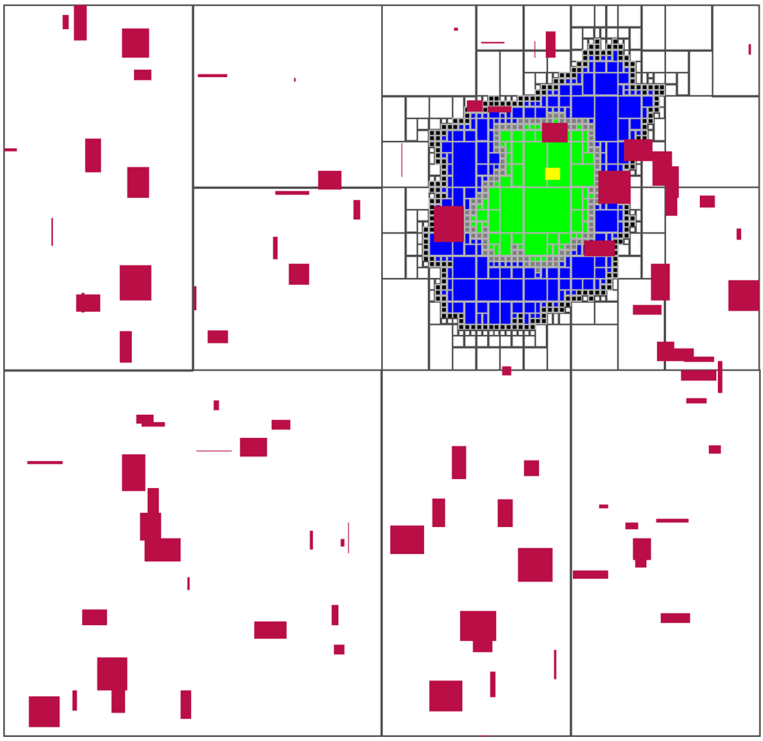


Fig. 7 Refined page regions of $\mathcal{I}_\mathcal{S}$ for $\mathcal{V}_1^\exists(U)$ and $\mathcal{V}_1^\forall(U)$

iii) Otherwise, if $\exists_k O \neg DOM(U, O, h)$, then there exists at least k objects that might dominate O with respect to h, such that h cannot be part of $\mathcal{V}_k^{\forall}(U)$. In this case, h can be colored in blue regardless of whether h is a directory of leaf node.

iv) Otherwise, if $\exists_k O \in \mathcal{D} : \neg DOM(U, O, h)$, then k objects may have a chance to dominate h, such that h cannot be fully contained in $\mathcal{V}_k^{\forall}(U)$. If h is a directory entry, then h is refined. If h is a leaf entry, then h is colored in grey in Fig. 7.

v) Otherwise, we can conclude that $DOM(U, O, h)$ holds for all but $k-1$ database objects, such that any point in h is guaranteed to have U as its k-nearest neighbor. Thus h must be completely contained in $\mathcal{V}_k^{\forall}(U)$, regardless of whether h is a directory of leaf entry, so h is colored green in Fig. 7.

Again, to show the effect of employing the space index $\mathcal{I}_{\mathcal{S}}$, a small sample database is given in Fig. 7 where $\mathcal{V}_1^{\exists}(U)$ and $\mathcal{V}_1^{\forall}(U)$ are computed. Here, no data index $\mathcal{I}_{\mathcal{D}}$ is used, thus all uncertain objects are shown as solid rectangles. It can be observed in Fig. 7 that, using the algorithm above, very large space entries of $\mathcal{I}_{\mathcal{S}}$ can be identified as being outside of $\mathcal{V}_5^{\exists}(U)$, illustrated by large white rectangles. In this example, the whole quadrant of the south-east database can be pruned. Are more detailed view on $\mathcal{V}_{10}^{\exists}(U)$ and $\mathcal{V}_{10}^{\forall}(U)$ is given in Fig. 8 for $k = 10$. Some larger entries of $\mathcal{I}_{\mathcal{S}}$ can be identified as completely contained in $\mathcal{V}_{10}^{\forall}(U)$, represented by the green rectangles of varying size. Finally, some space entries of $\mathcal{I}_{\mathcal{S}}$ are guaranteed to be outside of $\mathcal{V}_{10}^{\forall}(U)$ but inside of $\mathcal{V}_5^{\exists}(U)$, which correspond to the blue rectangles.

Yet, by employing exclusively the space index $\mathcal{I}_{\mathcal{S}}$ without using the data index $\mathcal{I}_{\mathcal{D}}$, dominance checks are required for every single uncertain database object. In the next section, we present our approach towards employing both index structures $\mathcal{I}_{\mathcal{D}}$ (indexing the uncertain objects) and $\mathcal{I}_{\mathcal{S}}$ (indexing space) at the same time. For this purpose, we need to find heuristics for a good trade-off between refining data entries of $\mathcal{I}_{\mathcal{D}}$ and refining space entries of $\mathcal{I}_{\mathcal{S}}$. Especially, we have to expect such hybrid algorithm to refine more data entries than the algorithm shown in Section 6, and to refine more space entries than the algorithm presented in this section. The reason is that the previous algorithms, which use only one index, assume that all the respective other data type is completely refined, thus all information is available. When both index structures are traversed however, then, at any time, only partial knowledge is known about both the data index and space index. Thus, it is possible that a space entry is refined which, while at a later stage of the algorithm, where more data entries are refined, it turns out that this refinement was unnecessary. Yet, we will show that our hybrid algorithm,

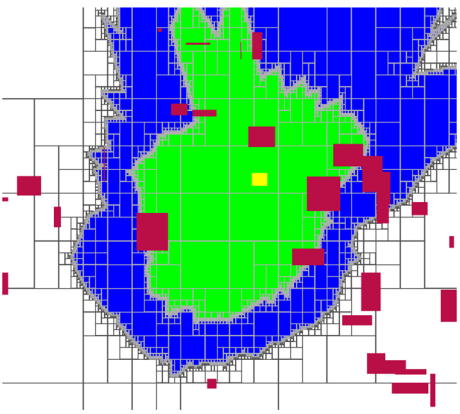


Fig. 8 Close view on the refined page regions of $\mathcal{I}_{\mathcal{S}}$ for $\mathcal{V}_{10}^{\exists}(U)$ and $\mathcal{V}_{10}^{\forall}(U)$

presented in the following Section 8 is capable of reducing both the number of refined data entries, and the number of refined space entries significantly.

8 Indexing $\mathcal{D}$ and $\mathcal{S}$

It seems apparent to combine the ideas of Section 6 and Section 7 and utilize both index structures ($\mathcal{I}_\mathcal{D}$ and $\mathcal{I}_\mathcal{S}$) to boost the performance. The non trivial task is the definition of a traversal order to minimize necessary operations.

Prelude Our approach is basically a depth-first traversal of $\mathcal{I}_\mathcal{S}$. Additionally we define $AS_\mathcal{D}$ to be the active set of entries of the index $\mathcal{D}$. Each entry $h \in \mathcal{I}_\mathcal{S}$ has its own active set and passes it on to its children (always removing irrelevant entries $e \in AS_\mathcal{D}$). $AS_\mathcal{D}$ contains all entries of $\mathcal{D}$ which have already been seen and not yet resolved during the traversal of the algorithm. For each entry $h \in \mathcal{I}_\mathcal{S}$ we first try to identify one of the two following properties (cf Fig. 9):

$\sum_{e \in AS_\mathcal{D}: Dom(e,U,h)} weight(e) \geq k$. In the case $Dom(e, U, h)$, all objects in data entry e are guaranteed to dominate U. Thus, we can increase a counter $count_{black} := \sum_{e \in AS_\mathcal{D}: Dom(e,U,h)} weight(e)$, which corresponds to the number of objects dominating U, by the weight of e. If $count_{black}$ reaches k, then h must not part of the possible UV cell $\mathcal{V}_k^\exists$ of U, and thus is colored unfilled in our visualization such as shown in Fig. 7. If $count_{black} < k$, then h is colored black, but might be assigned a different color later in the algorithm.

$\sum_{e \in AS_\mathcal{D}: Dom(U,e,h)} weight(e) > |\mathcal{D}| - k$. In the case $Dom(e, U, h)$, no object in data entry e can possibly dominate U. If this is guaranteed to hold for at least $|\mathcal{D}| - k + 1$ objects, then at most $k - 1$ database objects can possibly dominate h, and thus h must be part of the guaranteed Voronoi cell $\mathcal{V}_k^\forall$ and is colored green in our visualization. Our algorithm uses a counter $count_{green}$ to count the weight of entries e for which $Dom(U, e, h)$ does *not* hold. As long as $count_{green} < k$, the space cell h must be colored in green in Fig. 7.

If neither of the above cases hold, then we check if $\sum_{e \in AS_\mathcal{D}: PDom(e,U,h)} weight(e) < k$ and $\sum_{e \in AS_\mathcal{D}: PDom(U,e,h)} weight(e) \leq |\mathcal{D}| - k$. If both conditions hold, then space region U must be inside $\mathcal{V}_k^\exists$ and must not be inside $\mathcal{V}_k^\forall$. This corresponds to the case

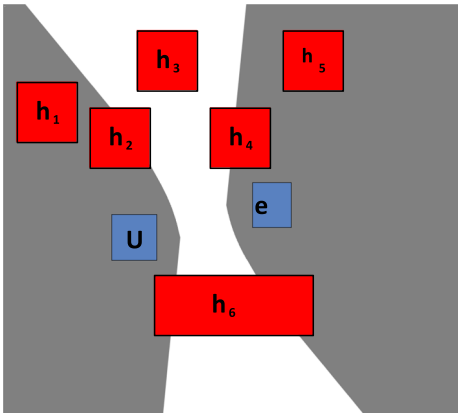


Fig. 9 Cases of domination for a data index entry e

where h is colored in blue in Fig. 7. Our algorithm uses the counters $count_{green}$ and $count_{grayorblue}$ to check this case.

If neither of the above two cases hold, then no final decision can be made for space region U, and either the current entry h or an entry $e \in AS_{\mathcal{D}}$ has to be resolved. Here we propose the following heuristics:

Case 2: $PDom(U, e, h) \Rightarrow$ resolve e or h depending on which one covers more space.
Intuition: uncertain area becomes small if both constructing objects are small

Case 3 $\neg PDom(U, e, h) \wedge \neg PDom(e, U, h) \Rightarrow$ resolve e.
Intuition: Resolving h can not yield any new information, since any child of h must also yield Case 3.

Case 4 $PDom(e, U, h) \Rightarrow$ resolve h if we find another data entry for which Case 4 holds (for this space entry h). Otherwise resolve e or h depending on which one covers more space. If e is a leaf entry only resolve h.
Intuition: If more than one data entry constructs Case 4, chances are good that large portions of h can be decided.

Case 6 $PDom(U, e, h) \wedge PDom(e, U, h) \Rightarrow$ resolve h. (cf Fig. 9, case 6)
Intuition: Resolving e can not yield any new information

Clearly, at one point there may be multiple data entries in the activate set of a space node h, which may yield different cases. It may be smart to prioritize the refinement of some data entries. In a nutshell, a data entry should be chosen which maximizes the chance that we can guarantee that h is not part of $\mathcal{V}(U)$. We propose to choose an entry e according to the following priority schema:

1. directory entries are prioritized over leaf entries.
2. prioritize cases in order 5, 4, 6, 3, 2, 1.
3. prioritize entries according to mindist to query

For ease of presentation of our algorithm, we define the function $maxprio(U \in \mathcal{D}, h \in \mathcal{I}_{\mathcal{S}}, E \subseteq \mathcal{I}_{\mathcal{D}})$ which maps an uncertain object U, a space region h and a set of data index entries E to the object which has the highest priority corresponding to the heuristics above.

Algorithm 1 Takes as parameters the object U for which the UV-cell is to be computed; the database $\mathcal{D}$ indexed by an R^*-tree $\mathcal{I}_{\mathcal{D}}$; and the Quadtree/KD-trie $\mathcal{I}_{\mathcal{S}}$ indexing the space. The idea of Algorithm 1 is to build an initial active set $AS_{\mathcal{D}}$ that is reasonable for all space partitions $h_i \in \mathcal{I}_{\mathcal{S}}$ to come during query processing. For this we perform a window-query-like operation. windowQuery*$(U, \mathcal{I}_{\mathcal{D}})$ basically performs a window query on $\mathcal{I}_{\mathcal{D}}$, but discards entries $e \in \mathcal{D}$ that fall in the window (since these entries cannot help to decide the borders of $\mathcal{V}(U)$). The result are now all entries $e \in \mathcal{I}_{\mathcal{D}}$ that have been seen during the window-query but have not been resolved. This set is then used as an initial *active set*(denoted as $AS_{\mathcal{D}}$) in the recursive Algorithm 2 which is initiated by Algorithm 1.

Algorithm 1 UV-Cell computation

Require: U,$\mathcal{I}_{\mathcal{D}}$,$\mathcal{I}_{\mathcal{S}}$
1: $AS_{\mathcal{D}}$ = windowQuery*(U,$\mathcal{I}_{\mathcal{D}}$)
2: UVCellCheck(U,$\mathcal{I}_{\mathcal{S}}.root$,$AS_{\mathcal{D}}$)

Algorithm 2 UVCellCheck

Require: U,h,$AS_{\mathcal{D}}$,k
1: e_{max} *//entry with maximum priority*
2: $count_{green}, count_{gray}, count_{grayorblue}, count_{blue}, count_{black} = 0$
3: **while** $AS_{\mathcal{D}} \neq \emptyset$ **do**
4: **for all** $e \in AS_{\mathcal{D}}$ **do**
5: **if** case$(e, U, h) > 1$ **then**
6: $count_{green}$ += weight
7: $e_{max} = maxprio(e_{max}, e)$
8: **if** case$(e, U, h) \in 3, 4, 5, 6$ **then**
9: $count_{gray}$ += weight
10: **if** case$(e, U, h) \in 3, 4, 5$ **then**
11: $count_{grayorblue}$ += weight
12: **if** case$(e, U, h) \in 4, 5, 6$ **then**
13: $count_{blue}$ += weight
14: **if** case$(e, U, h) = 5$ **then**
15: $count_{black}$ += weight
16: **if** $count_{black} \geq k$ **then**
17: h is not part of UVCell
18: return
19: **if** $count_{green} < k$ **then**
20: h is part of inner UVCell $\mathcal{V}_k^{\forall}$
21: return
22: **if** $count_{gray} < k$ **and** isLeaf(h) **then**
23: h is border of inner UVCell $\mathcal{V}_k^{\forall}$
24: return
25: **if** $count_{blue} < k$ **and** $count_{grayorblue} \geq k$ **then**
26: h is part of outer UVCell $\mathcal{V}_k^{\exists}$
27: return
28: **if** $count_{black} < k$ **and** isLeaf(h) **then**
29: h is border of outer UVCell $\mathcal{V}_k^{\exists}$
30: return
31: $splitData = false$, $splitSpace = false$
32: **if** isLeaf(h) **or** case$(e_{max}, U, h) = 3$ **or** p(e_{max}) > p(h) **then**
33: **if** ¬isLeaf(e_{max}) **and** case$(e_{max}, U, h) \neq 6$ **then**
34: $splitData = true$
35: **if** ¬isLeaf(h) **and** case$(e_{max}, U, h) \neq 3$ **then**
36: **if** isLeaf(e_{max}) **or** case$(e_{max}, U, h) = 6$ **or** p$(e_{max}) \leq$ p(h) **or** $\neg splitData$ **then**
37: $splitSpace = true$
38: **if** $\neg splitSpace$ **and** ¬isLeaf(h) **then**
39: $splitSpace = true$
40: *//Execute splits*
41: **if** $splitData$ **then**
42: $AS_{\mathcal{D}} = AS_{\mathcal{D}} \setminus e_{max} \cup e_{max}.children$
43: **if** $splitSpace$ **then**
44: **for all** $h_c \in h.children$ **do**
45: UVCellCheck(U,h_c,$AS_{\mathcal{D}}$.clone())

Algorithm 2 This algorithm requires the uncertain object U for which the UV-cell is being computed, *one* region of the result space h(initially the root of the kd-tree), the active set $AS_{\mathcal{D}}$ containing a set of $\mathcal{I}_{\mathcal{D}}$-entries, and the parameter k. The algorithm works as follows:

- The main loop (line 3) iterates over the active set until either the space region has been flagged or there are no more $\mathcal{I}_{\mathcal{D}}$-entries to consider.
- In another loop (lines 4 – 18), the algorithm tests whether the current active set suffices for flagging the space region. To achieve this, a set of counter variables is defined:
  - $count_{green}$ checks for entries that include parts of the result space in their outer cell ($domCase > 1$). The space can only be part of $\mathcal{V}_k^{\forall}(U)$ if this counter is below k.
  - $count_{gray}$ tracks border regions around the inner cell that cannot be further expanded because the spatial index is at its maximum depth (isLeaf(h) is true). All domCases greater than 2 qualify to increase this counter.
  - $count_{grayorblue}$ keeps track of undecided cases ($domCase = 3$) that need to be further explored before flagging.

- $count_{blue}$ counts entries that include parts of the result space in their inner cell ($domCase > 3$). The space can only be part of $\mathcal{V}_k^{\exists}(U)$ if this counter is below k.
- $count_{black}$ tracks border regions around the outer cell that cannot be further expanded because the spatial index is at its maximum depth (isLeaf(h) is true). Only domCase 5 increments this counter.

If an entry qualifies to increment a counter, the entry's weight is added. If entry is a leaf, i.e., is not an index page, the weight is one. Otherwise, weight is the number of all leaf entries included in this entry. Consideration of $\mathcal{I}_{\mathcal{D}}$-entries is interrupted if more than k dominating cases ($domCase = 5$) have been found (lines 16 – 18). For further exploration, the entry with highest priority is determined in line 7.

– Once every entry has been checked, the algorithm tests if the space region can be assigned a flag based on the counters in lines 19 – 30. Once a flag has been assigned, the recursion branch ends.
– If no flag could be assigned, the algorithm decides which index will be refined next – e_{max} (data split) or h (space split). The dicision is made based on whether once of the indices is already fully refined (isLeaf() is true), which of the two pages has the larger perimeter (p(e_{max}) < / > p(h)), and the domCase of e_{max}, namely:

  Case 4: there is a chance that refining h may allow child entries of h to be pruned, and refining e_{max} may allow child entries of e_{max} to prune all of h. Therefore, we refine both entries in this case.
  Case 6: refining e cannot possibly allow us to prune h. However, refining h may allow us to either prune children of h or to return children of h as true hits. Thus we refine h.
  Case 3: no children of h can possibly be pruned.[3] Thus we split e_{max}, which may allow h to be pruned.
  Case 2: we refine h.

– Finally, space index entries h which must be completely contained in $\mathcal{V}(U)$ are identified as entries having only **Cases 1-3** in their active set. Computation breaks if this is the case. After splitting the objects according to the rules above. We recursively restart the algorithm with the new objects.

Figure 10 illustrates in which manner the algorithm resolves entries of $\mathcal{I}_{\mathcal{D}}$ and $\mathcal{I}_{\mathcal{S}}$. The figures shows all pages and objects of $\mathcal{I}_{\mathcal{D}}$ which have been seen during the computation of the possible Voronoi-cell $\mathcal{V}(U)$ of the green objects U. Refined data objects are represented by filled red rectangles and refined directory nodes are represented by unfilled red rectangles. Furthermore, refined entries of $\mathcal{I}_{\mathcal{S}}$ are shown as (i) unfilled black rectangles if they are guaranteed to be fully outside of $\mathcal{V}(U)$, (ii) as black rectangles if on the border of $\mathcal{V}(U)$, and (iii) as blue rectangles if completely inside $\mathcal{V}(U)$. We can observe that in areas far away from the UV cell, $\mathcal{I}_{\mathcal{S}}$ is resolved coarse whereas at the border of the cell it is resolved at very fine granularity. The entries of $\mathcal{I}_{\mathcal{D}}$ are also only resolved around the UV cell. Note that although the number of resolved objects seems large, most of the objects are only needed for a small fraction of the computations, especially on coarser levels of $\mathcal{I}_{\mathcal{S}}$. Finally, note that a nice side effect of this computation is that we obtain a tight superset of the (uncertain-) Delaunay neighbors of U. This can be achieved by memorizing the objects O for which Case 4 or Cast 6 (see Definition 6) holds.

[3] recall that if $e_{max}^{\mathcal{D}}$ corresponds to case 3, then there exists no R^*-entry such that case 4 holds.

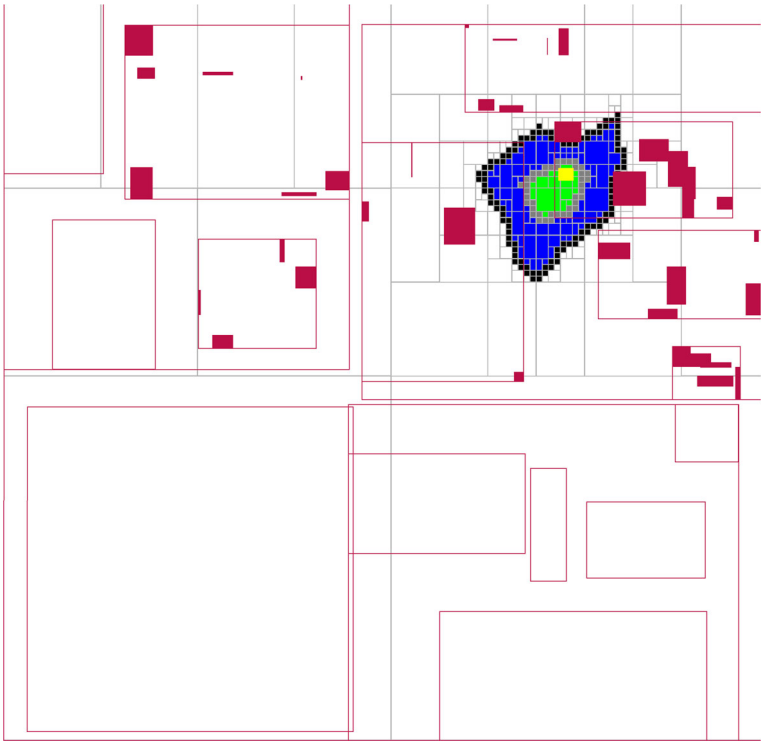


Fig. 10 Example of refinement

8.1 Discussion: dynamic data

In many applications for uncertain Voronoi regions, such as the ride-sharing application presented in Section 1, data changes continuously. In such an application drivers update their location frequently, thus changing their uncertainty region. And even if a driver does not update its location, its uncertainty region grows. Clearly, maintaining and updating the data index $\mathcal{I}_\mathcal{D}$ using a traditional R^*-tree of drivers uncertainty regions in not viable in such a case. However, we do not aim at updating Voronoi-cells each time the database changes. Rather, we aim to compute the Voronoi-cells at a single snapshot point-in-time. For this purpose, we propose to simply build a new data index $\mathcal{I}_\mathcal{D}$ (using a bulk-load strategy) for each query. In our experiments, we use an R-Tree and show that the time required for this bulk load is neglectable. But it should be noted, that we use the data index $\mathcal{I}_\mathcal{D}$ solely for the purpose of grouping data objects for pruning. Thus, we keep the type of data structure abstract, only assuming that $\mathcal{I}_\mathcal{D}$ uses rectangular approximation of groups of uncertainty regions, in order for the techniques of Section 4 to be applicable. Yet, we want to note that solutions to index continuously moving objects have been proposed, with and without uncertainty [28–30]. If available, such index structures, if maintained over a changing moving object database, can be used as directly as data index $\mathcal{I}_\mathcal{D}$ without having to bulk-load a new index. We also note that the space index $\mathcal{I}_\mathcal{S}$ is never materialized, and only use to prune search space. There is no need to update this tree.

8.2 Discussion: spherical regions

In many applications, such as applications using GPS uncertainty, a more realistic modelling of the uncertain location of an object is to employ spherical uncertainty regions, commonly used to bound (at some given level of significance) a gaussian distribution. In the case of a spherical approximation of the uncertainty region of an object, the spatial domination scheme revisited in Section 4 can no be directly applied. Yet, an equivalent spatial domination technique has been presented concurrently and independently in [31] and [32]. For three points a, b, and c bound by spherical regions A, B, and C, respectively, this techniques allow to decide if a is guaranteed to be closer (farther) from b than c. Applying this technique for our Voronoi-cell computation, we can simply approximate each spherical region by a minimum bounding square. Then, whenever our algorithm applies the decision criterion of Section 4 to three leaf rectangles of the R-tree, we apply the techniques of [31, 32] to check for spherical domination.

9 Experiments

Our experimental evaluation investigates algorithm behaviour w.r.t. maximum kd-trie depth, database size, uncertainty, and data dimension. To assess the uncertainty in our synthetic data, we define a parameter *extent* to control the size of the uncertain objects (i.e., the object's MBR) and corresponds to the maximum extent of an object in one dimension. Experiments use synthetically generated datasets as well as an excerpt from the T-Drive taxicab trajectory dataset [7, 8] where we added synthetic uncertainty to each GPS signal of a taxi cab. We implemented all approaches in the ELKI framework [33], which also provided an R-tree implementation.

The data space is always normalized to [0,1] in each dimension. For our synthetic data, objects are uniformly distributed over space with a randomly assigned side length between 0 and maximum extent. Most examples previously used in this work use synthetic data generated this way, including Figs. 5a–d, 6, 7, 8 and 10.

Data points from the real world dataset were sampled as a single snapshot of the world, on the afternoon of February 2nd, 2008. Therefore, one data point corresponds to the position of one taxicab within the city of Beijing, China. After removing some outliers, this dataset contains 890 separate entities. To suit our application of location obfuscation, sample locations were randomized using a Gaussian distribution based on this object's location. A single sample from this distribution is then set as center of the object's new MBR, with its extent set to 6σ of this object's Gaussian (3σ to each direction). On said city scale, an extent of 0.01 would equal an area of 100m side length.

Table 2 Default settings

Parameter	default value	Notation	Algorithm
Dimension	2	DI	Data Index traversal (Section 6)
db size	1000	SI	Space Index traversal (Section 7)
Extent	0.01	DSI	Data & Space Index traversal (Section 8)
Tree depth	14	SR	Single Rectangle (Implementation of [6])

Table 2 denotes input parameters and their default settings, as well as an explanation of our algorithm notation. If not otherwise specified, the following experiments use the specified default values. Our evaluation focusing on approximation quality use DSI exemplarily for all algorithms from Sections 6–8, since these algorithms do not differ in the resulting approximation quality, but in efficiency only. Naturally, our real world dataset T-Drive has inherent values that override parameters, namely dimension and size of database. The standard depth of 14 refers to a maximum of 14 splits in our index structure, corresponding to $16384 (= 2^{14})$ individual grid cells. Applied to a city scale of 10 by 10 kilometers, each grid cell side would measure some 78 meters. As the proposed approach is later scaled up to a depth of 22, grid cells correspond to an area of only 4.8 by 4.8 meters, which on a city scale is extremely precise.

9.1 State of the art competitor

As a competitor solution to validate our computation of uncertain first-order Voronoi-cells, in terms of run-time and effectiveness, is the approach of [6]. This approach approximates the possible Voronoi-cell of an uncertain object by a single rectangular region. However, this approach is only applicable to approximate the possible first-order Voronoi cells $\mathcal{V}_1^\exists(U)$ of an uncertain object U. To the best of our knowledge, our solution is the first to compute higher-order Voronoi cells for uncertain data, and the first to compute a guaranteed Voronoi cell. In order to allow a fair experimental evaluation, we extend the solution of [6] to approximate higher-order possible Voronoi cells as described in the following.

In a nutshell, the approach of [6] computes a progressive and a conservative rectangular approximation of $\mathcal{V}_1^\exists(U)$. An initial conservative approximation is obtained by bounding the whole data space by a single rectangle. This approximation trivially covers $\mathcal{V}_1^\exists(U)$. Then, this rectangle is split recursively, where split dimensions and locations are chosen heuristically. Whenever a split is performed, the concept of spatial domination (c.f. Section 3) is applied to see if any of the new residual regions obtained from the split completely contains $\mathcal{V}_1^\exists(U)$. If not, the split if rejected and a new split dimension and location are chosen heuristically. If one residual split region does contain $\mathcal{V}_1^\exists(U)$, then the algorithm recursively continues with that region. After each split, only at most one region can completely contain $\mathcal{V}_1^\exists(U)$. The algorithm terminates when the gain by further splitting the current approximation of $\mathcal{V}_1^\exists(U)$ drops below a specified threshold. A progressive approximation of $\mathcal{V}_1^\exists(U)$ is obtained by choosing the boundaries of U itself as an initial approximation. This initial approximation is progressive, as any point inside U is guaranteed to have a chance U as a nearest neighbor, as U might be located at the exactly same location. Then, the progressive approximation is again expanded iteratively, using the concept of spatial domination to check whether a new approximation is completely contained within $\mathcal{V}_1^\exists(U)$. Note that for each of the spatial domination checks, each database object has to be tested for spatial domination, since no index structures are employed.

To apply this solution for higher-order Voronoi cells, we adapt the spatial domination check of [6] as follows: given the current conservative approximation C, rather than finding a single database object O such that $Dom(O, U, C)$, i.e., such that O is guaranteed to dominate U with respect to C, we now continue the domination checks until k such objects are found. Analogously, for the current progressive approximation P, we now find k objects O such that $PDom(O, U, P)$ holds, that is, we find k objects which are might possibly dominate U, rather than just attempting to find one such object, as done by the algorithm proposed in [6]. In the following experimental evaluation, the resulting adapted algorithm of [6], which computes a single rectangular progressive approximation and a single rectangular

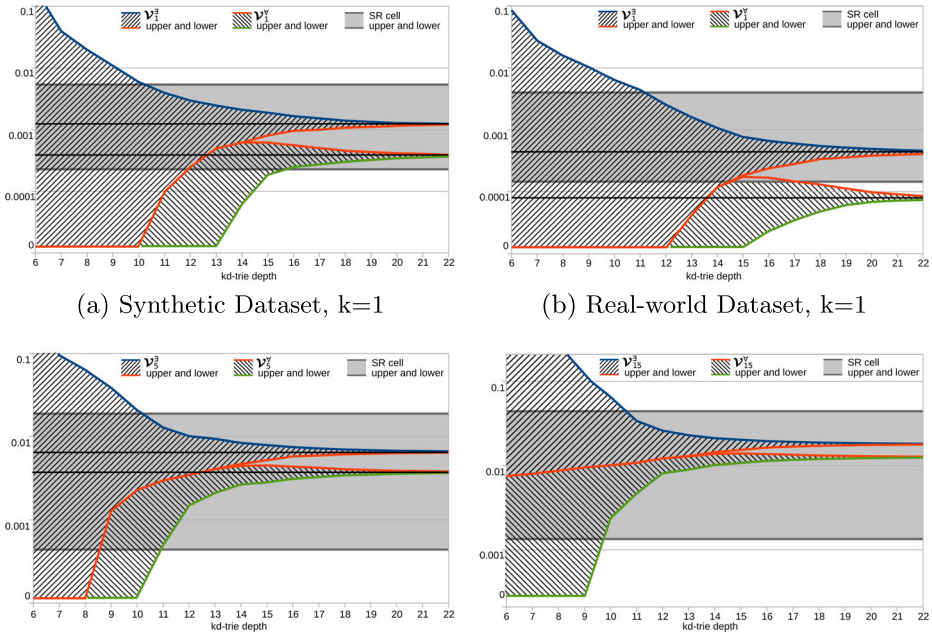
conservative approximation, will be used as our competitor approach. This approach will be denoted as *Single Rectangle (SR)*.

9.2 Approximation quality

Our first evaluation explores how well the generated bounds approximate a cell. Therefore, we set the tree depth for our implementation to various levels between 5 and 22, corresponding to the number of splits. Evidently, smaller grid cells can more closely follow the outline of a UV-cell.

Figure 11 visualizes how upper and lower bounds converge with larger tree depth. The dark blue line refers to the upper bound of DSI, the orange line to its lower bound, each represented by the total volume of their cells. The hatched space in between the two lines refers to the range in which the true cell volume must be located. As a point of reference, upper and lower bounds from the *Single Rectangle (SR)* approach have also been denoted in the same graphic, with the area shaded in grey corresponding to the approximation error. Since SR does not use an index, its results remain unchanged for all settings of the maximum tree depth.

Performance was tested on different datasets. Figure 11a represents average results for runs on synthetic data, while Fig. 11b contains the results for our real world dataset. While overall performance is fairly comparable, DSI provides a usable lower bound remarkably early, with as little as 8 tree splits necessary to outperform SR. We can see that for a sufficiently tree depth between 18 and 22, the lower- and upper-bound approximation of the inner-Voronoi cell $\mathcal{V}_k^{\forall}(U)$ and the outer-Voronoi cell $\mathcal{V}_k^{\exists}(U)$ approach each other. Consequently, the approximation error converges to zero, thus indicating that our approach is able



(a) Synthetic Dataset, k=1 (b) Real-world Dataset, k=1

(c) Synthetic Dataset, k=5 (d) Synthetic Dataset, k=15

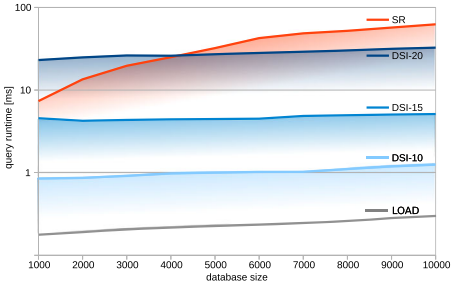
Fig. 11 Approximation quality for DSI and SR

to near-optimally approximate $\mathcal{V}_k^{\forall}(U)$ and $\mathcal{V}_k^{\exists}(U)$. We further observe that for larger values of k, the space covered by the possible Voronoi-cell $\mathcal{V}_k^{\exists}(U)$ and the guaranteed Voronoi-cell $\mathcal{V}_k^{\forall}(U)$ seem to approach each other. The reason is that all cells grow radially in k. Thus, the space covered by both cells $\mathcal{V}_k^{\forall}(U)$ and $\mathcal{V}_k^{\exists}(U)$ cover space quadratic in the diameter of each cell. Yet, the space covered by the set-difference $\mathcal{V}_k^{\exists}(U) \setminus \mathcal{V}_k^{\forall}(U)$ forms the shape of a ring, the surface of which grows only linear in it's diameter. Thus, we observe that indeed both cells $\mathcal{V}_k^{\forall}(U)$ and $\mathcal{V}_k^{\exists}(U)$ increase for larger k, but they relative size approaches each other for large values of k. Finally, our competitor approach SR shows fairly similar behaviour on both datasets, with results looking even more similar than they are due to logarithmic scale. We argue that the large approximation error of SR makes it a bad choice for reliable decision making. The following subsection will answer the question if the SR approach can redeem itself in terms of run-time.

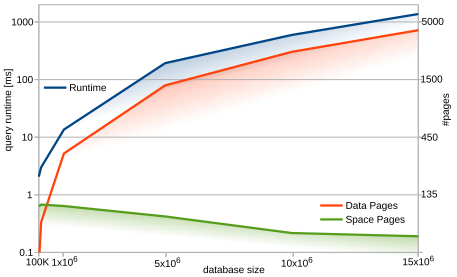
9.3 Algorithmic runtime

Runtime experiments were conducted by varying database size and dimensionality, between our three different traversal approaches compared to SR as well as for DSI alone to cover larger ranges of database size (other approaches have been excluded for large database sizes due to their excessive run-time performance).

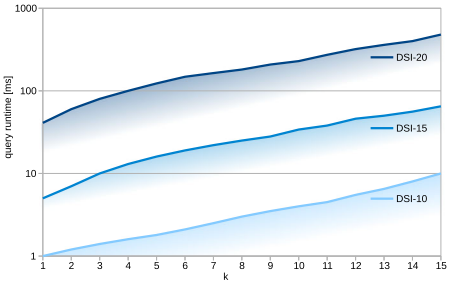
In Fig. 12, run times to calculate one UV-cell are denoted over different database sizes. Figure 12a contains results for the approach *Data and Space Index Traversal (DSI)* in three different configurations of kdtrie-depth, and SR. As is to be expected, DSI has higher run-times for higher depth of its spatial index, since border regions will be explored further and require more domination checks. However, runtime increases only very lightly for greater



(a) All Approaches Compared



(b) DSI (Data & Space)



(c) DSI (Data & Space)

Fig. 12 A runtime comparison for our DSI and the SR-approach over different sizes of DB

database sizes. This is because the combined approach of data and space index allows for early pruning of large portions of the database. SR starts off at a considerable speed, but since it features pairwise comparisons without the use of an index, it does not scale well for higher numbers ob database objects. For comparison, $LOAD$ depicts the necessary runtime to bulk-load the R-tree of DSI. Bulk-load time takes roughly one order of magnitude less than the actual query, which we consider negligible.

As query performance generally deteriorates for larger datasets, further scaling experiments were conducted using DSI only. Figure 12b shows the results of database populations from 10K to 15 Million objects. To avoid gross overlapping of objects, object extent has been lowered to 0.001 for these runs. The left axis again refers to the average time to perform one UV-cell calculation, which corresponds to the blue data line. We observe a slightly superlinear scaling, confirming our theoretical observations that (i) adding more objects leads to linearly more intersections with Voronoi cells, which are at least as big as U, and (ii) a linear increase in object count causes logarithmic tree index growth. This results in a combined log-linear growth in runtime.

The right scale denotes average page views during cell calculation, with the orange line referring to pages of the data index, and the green line for pages of the space index. Note that data index exploration roughly follows runtime development, while the space index is used less for larger databases. This is easily explained by a constant tree depth, resulting in a constant resolution of space. With a higher database population, the likelihood of all relevant objects being enclosed in a small space increases.

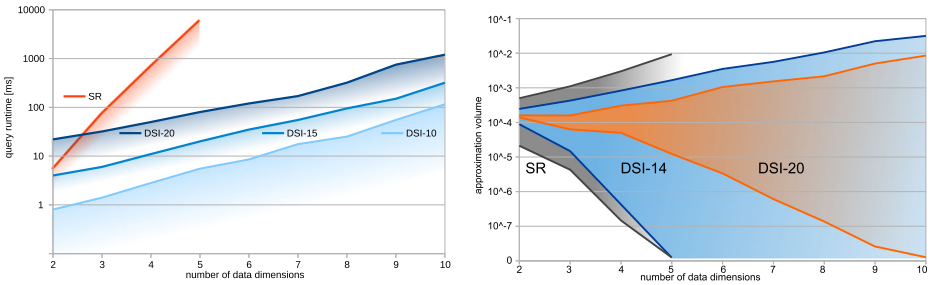
In addition, we also scale the parameter k in Fig. 12c. Since the competitor approach SR is applicable not applicable to the case of having $k > 1$, this competitor is omitted for this experiment. We observe that the run-time increases sub-linear (note the logarithmic runtime scale) in k. This is explained by the fact that the main bottleneck of our algorithm is the traversal of the space-index $\mathcal{I}_S$ to leaf-level, which is performed around the border of the Voronoi-cells. Thus, we expect our algorithm to scale linear in the extend of the Voronoi cells. As we increase k, we expect the *area* of the Voronoi cell to increase linear in k, thus yielding a sub-linear increase in extent.

9.4 Effect of data dimensions

Although the simple case of a two-dimensional world is most intuitive for most applications mentioned before, all approaches can manage high-dimensional datasets as well. The main limitation here is keeping the approximation error low in all dimensions at once, as well as balancing computational complexity.

Figure 13 displays performance for different kdtrie-depths of our DSI approach (depths 10, 15 and 20) as well as SR for multi-dimensional datasets. As runtime and memory usage of SR do not scale well for more than five data dimensions, experiments excluded this approach for higher dimensionalities than 5. An evaluation of runtime as shown in Fig. 13a shows constant increase for all approaches. The relative steepness of increase is due to the growing inefficiency of pruning methods in high dimensions, which deteriorates searches toward a linear scan, which itself has quadratic complexity. As expected, runtime of DSI increases with a higher kdtrie-depth, since border regions in the spatial index will cause repeated domination checks.

Approximation quality for higher dimensions is shown in Fig. 13b. As mentioned before, fitting a bound to a more and more complex object leaves much room for approximation error. Therefore, volumes of upper and lower bounds diverge more for higher dimensions. Displayed here are bounds for SR up to dimension 5 (grey) and two different settings of



(a) Runtime (b) Approximation Quality

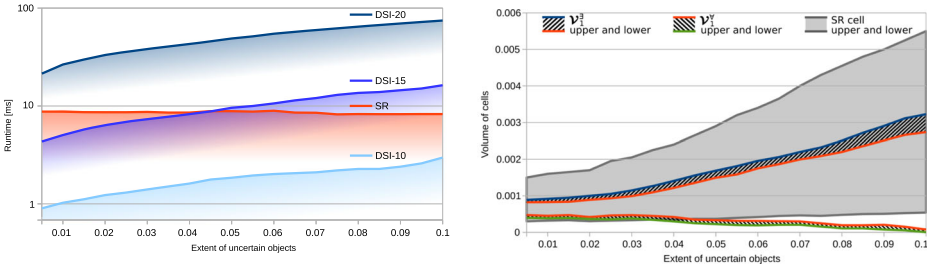
Fig. 13 A comparison for increasing data dimensions

our DSI approach, once with a depth of 14 (blue) and a depth of 20 (orange). As expected, a higher depth allows for more tree splits per dimension and thus a better approximation.

9.5 Effect of uncertainty extent

Finally, we evaluate the effect of the uncertainty, on both run-time and approximation quality. Clearly, the extent of uncertainty is highly application specific. In an application such as ride-sharing, this uncertainty depends on the GPS-quality as well as on the frequency of GPS updates of drivers. To evaluate this parameter, we increase the bounding boxing *extent* of uncertain objects in Fig. 14 for our DSI approach (depths 10, 15 and 20) as well as SR. Before we take a closer look at the results, recall what a higher uncertainty regions means: The data set becomes more fuzzy and uncertainty regions of objects overlap more. As a consequence, the guaranteed Voronoi cell $\mathcal{V}_k^{\forall}(U)$ of an object U shrinks. The reason is that due to uncertainty, less regions in space can be uniquely identified as having U as their k-nearest neighbor. At the same time, the possible Voronoi cell $\mathcal{V}_k^{\exists}(U)$ of an object U grows, as more regions in space might possibly have U as their nearest neighbor.

First, Fig. 14a, shows that the single rectangle approach SR exhibits are nearly constant run-time. The reason is that the main challenge of this approach is to fit a rectangle on the unknown Voronoi-cell. Initially, this approach starts by using the full space as a first approximation. This rectangle is iteratively halved to fit the unknown Voronoi cell. Thus, by having a larger Voronoi cell, less halving-steps are required, thus even yielding a slight performance gain for larger Voronoi cells. In contrast, the run-time of our DSI approach increases. The reason is that the extent of the possible Voronoi cell increases, and thus, more



(a) Runtime (b) Approximation Quality

Fig. 14 Effect of maximum uncertainty extent

cells of the space index $\mathcal{I}_S$ become "borderline" and need to refined to the maximum depth of the kd-trie $\mathcal{I}_S$. But Fig. 14a also shows that, in a case where the uncertainty regions are large, a smaller value of *tree depth* can be used to offset the additional run-time. In addition, Fig. 14b evaluates the approximation quality for larger uncertainty regions. We see that the constant run-time of the SR approach also incurs an extremely low approximation quality for larger uncertainty regions. We see that the lower- and upper-bound of the $\mathcal{V}_1^{\exists}$ Voronoi cell using the SR approach becomes extremely loose. In this case, the corresponding upper-bound doubles the space of the actual cell, whereas the lower-bound cell is extremely small. Note that the SR approach is not able to compute the $\mathcal{V}_1^{\forall}$ cell. In contrast, we see that using our DSI approach, the uncertain Voronoi cell of an object can still be approximated very accurately, even for large uncertainty regions. For the possible Voronoi cell $\mathcal{V}_1^{\exists}$, the margin of error remains low, for our default setting of a *tree depth* of 14. We also see that the average size of guaranteed Voronoi cells $\mathcal{V}_1^{\forall}$ approaches zero here, as in most cases, there is not space that is guaranteed to have the query object as its nearest neighbor.

10 Conclusions

In this work, we proposed an index-supported approach to approximate the shape of Voronoi-cells to support nearest neighbor queries on uncertain data. In an uncertain database the attribute values of objects are random variables. Consequently, the location, size and shape of the Voronoi-cell of an object U is a random variable, too. Therefore, we proposed to approximate, progressively and conservatively, the space which is guaranteed to be part of the Voronoi-cell of U, denoted as the guaranteed Voronoi-cell $\mathcal{V}_k^{\forall}$. In addition, we also approximate the space which has a non-zero probability to be part of the Voronoi-cell of U, which we call the possible-Voronoi cell $\mathcal{V}_k^{\exists}$. We show how our solutions can be extended to k'th-order Voronoi cells, i.e., the space which is guaranteed to (may possibly) have U as one of their k-nearest neighbors. Our approach uses an R^*-tree as a hierarchical access method to efficiently find the set of uncertain objects that influence the possible Voronoi-cell of an uncertain object U, i.e., the set of Delauny-neighbors of U. In addition, we propose to use a kd-trie as a hierarchical access method to identify regions of space which must (not) be part of a Voronoi-cell. Compared to the state-of-the-art of computing uncertain Voronoi-cells, our approach allows for much higher approximation quality, since our result approximation consists of a set of rectangular kd-trie nodes, rather than a single bounding rectangle.

Our solution focuses on computing the Voronoi cell of a single query object only, as desirable in a ride-sharing application where a driver tries to locally optimize his own influence on customers. Yet, other applications may have a broader view, such as a classic taxi-application, where a central server organizes drivers and matches customers to them. In such an application, it may be useful to compute the Voronoi-cells of all taxi's, and direct drivers in a way such that Voronoi cells becomes balanced, thus minimizing worst-case wait-times of customers, and creating a fair environment for drivers. While our solution can be simply iterated over each object, an interesting future direction is to perform such computation with less redundancy, thus reusing domination checks for one query object that have been made for another.

Another future direction, is to compute, for an object U, the region having U as a k-nearest neighbor with a probability of at least $\tau \in [0, 1]$. For this problem, the $\mathcal{V}_k^{\forall}$ Voronoi cell can be seen as the special case where $\tau = 1$, and the $\mathcal{V}_k^{\exists}$ Voronoi cell can be seen as the special case where τ approaches zero. Without making any assumption on the distribution of the uncertain objects, this is a $\#P$-hard problem, as arbitrary probabilistic distributions

and stochastic dependencies need to be considered [34]. Yet, if independence is assumed between objects, and objects are assumed to follow a discrete uncertainty model, then a computation of contour lines is a possible direction for future work.

Acknowledgments Part of the research leading to these results has received funding from the Deutsche Forschungsgemeinschaft (DFG) under grant number RE 266/5-1 and from the DAAD supported by BMBF under grant number 57055388. Reynold Cheng was supported by the Research Grants Council of Hong Kong (RGC Project (HKU 711110)).

Andreas Zufle has been supported by National Science Foundation AitF grant CCF-1637541.

Reynold Cheng was supported by the Research Grants Council of HK (Project HKU 17205115) and HKU (Projects 102009508 and 104004129). We would like to thank the reviewers for their insightful comments.

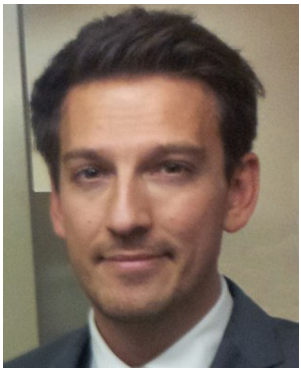
References

1. Chow CY, Mokbel MF, Aref WG (2009) Casper*: query processing for location services without compromising privacy. ACM TODS 34(4):24
2. Beskales G, Soliman MA, IIyas IF (2008) Efficient search for the top-k probable nearest neighbors in uncertain databases. Proc VLDB Endowment 1(1):326–339
3. Cheng R, Xie X, Yiu ML, Chen J, Sun L (2010) Uv-diagram: a voronoi diagram for uncertain data. In: ICDE, IEEE, pp 796–807
4. Ali ME, Tanin E, Zhang R, Kotagiri R (2012) Probabilistic voronoi diagrams for probabilistic moving nearest neighbor queries. DKE 75:1–33
5. Bernecker T, Emrich T, Kriegel HP, Mamoulis N, Renz M, Zufle A (2011) A novel probabilistic pruning approach to speed up similarity queries in uncertain databases. In: Proceedings of the 27th international conference on data engineering (ICDE). Hannover, pp 339–350
6. Zhang P, Cheng R, Mamoulis N, Renz M, Züfle A, Tang Y, Emrich T (2013) Voronoi-based nearest neighbor search for multi-dimensional uncertain databases. In: Proceedings of the 29th International conference on data engineering (ICDE). Brisbane, pp 158–169
7. Yuan J, Zheng Y, Zhang C, Xie W, Xie X, Sun G, Huang Y (2010) T-drive: Driving directions based on taxi trajectories. In: SIGSPATIAL, pp 99–108
8. Yuan J, Zheng Y, Xie X, Sun G (2011) Driving with knowledge from the physical world. In: SIGKDD, pp 316–324
9. Niedermayer J, Züfle A, Emrich T, Renz M, Mamoulis N, Chen L, Kriegel HP (2013) Probabilistic nearest neighbor queries on uncertain moving object trajectories. Proc VLDB Endowment 7(3):205–216
10. Emrich T, Kriegel HP, Kröger P, Renz M, Züfle A (2009) Incremental reverse nearest neighbor ranking in vector spaces. In: Proceedings of the 11th International symposium on spatial and temporal databases (SSTD). Aalborg, pp 265–282
11. Beckmann N, Kriegel HP, Schneider R, Seeger B (1990) The R*-Tree: An efficient and robust access method for points and rectangles. In: Proceedings of the ACM International conference on management of data (SIGMOD). Atlantic City, pp 322–331
12. Orenstein JA, Merrett TH (1984) A class of data structures for associative searching. In: ACM SIGACT-SIGMOD. ACM, pp 181–190
13. Schmid KA, Emrich T, Zufle A, Renz M, Cheng R (2015) Approximate uv computation based on space decomposition. In: Proceedings of the 14th International symposium on spatial and temporal databases (SSTD). Hong Kong
14. Cheng R, Kalashnikov DV, Prabhakar S (2004) Querying imprecise data in moving object environments. In: IEEE Transactions on knowledge and data engineering
15. Li J, Saha B, Deshpande A (2009) A unified approach to ranking in probabilistic databases. Proc VLDB Endowment 2(1):502–513
16. Bernecker T, Kriegel HP, Mamoulis N, Renz M, Zufle A (2010) Scalable probabilistic similarity ranking in uncertain databases. IEEE Trans Knowl Data Eng 22(9):1234–1246
17. Cheema MA, Lin X, Wang W, Zhang W, Pei J (2010) Probabilistic reverse nearest neighbor queries on uncertain data. IEEE Trans Knowl Data Eng 22(4):550–564

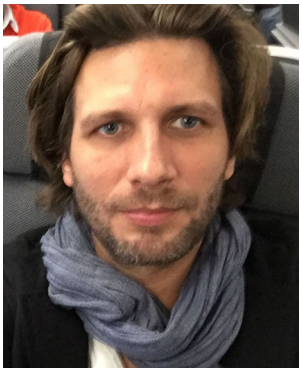
18. Aurenhammer F (1991) Voronoi diagrams-a survey of a fundamental geometric data structure. ACM CSUR 23(3):345–405
19. Sharifzadeh M, Shahabi C (2010) Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries. VLDB Endowment 3(1–2):1231–1242
20. Zheng B, Xu J, Lee WC, Lee L (2006) Grid-partition index: a hybrid method for nearest-neighbor queries in wireless location-based services. VLDB J 15(1):21–39
21. Nutanong S, Zhang R, Tanin E, Kulik L (2008) The v*-diagram: a query-dependent approach to moving knn queries. VLDB Endowment 1(1):1095–1106
22. Sharifzadeh M, Shahabi C (2009) Approximate voronoi cell computation on spatial data streams. VLDB J 18(1):57–75
23. Akdogan A, Demiryurek U, Banaei-Kashani F, Shahabi C (2010) Voronoi-based geospatial query processing with mapreduce. In: IEEE CloudCom. IEEE, pp 9–16
24. Kolahdouzan M, Shahabi C (2004) Voronoi-based k nearest neighbor search for spatial network databases. In: VLDB Endowment, VLDB Endowment, pp 840–851
25. Roussopoulos N, Kelley S, Vincent F (1995) Nearest neighbor queries. In: ACM SIGMOD, vol 24. ACM, pp 71–79
26. Emrich T, Kriegel HP, Kröger P, Renz M, Zufle A (2010) Boosting spatial pruning: on optimal pruning of MBRs. In: Proceedings of the ACM international conference on management of data (SIGMOD). Indianapolis, pp 39–50
27. Hjaltason GR, Samet H (1995) Ranking in spatial databases. In: Proceedings of the 4th international symposium on large spatial databases (SSD). Portland, pp 83–95
28. Šaltenis S, Jensen CS, Leutenegger ST, Lopez MA (2000) Indexing the positions of continuously moving objects, vol 29. ACM
29. Zhang M, Chen S, Jensen CS, Ooi BC, Zhang Z (2009) Effectively indexing uncertain moving objects for predictive queries. Proc VLDB Endowment 2(1):1198–1209
30. Emrich T, Kriegel HP, Mamoulis N, Renz M, Züfle A (2012) Indexing uncertain spatio-temporal data. In: Proceedings of the 21st ACM conference on information and knowledge management (CIKM). Maui, pp 395–404
31. Lian X, Chen L (2009) Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data. VLDB J Int J Very Large Data Bases 18(3):787–808
32. Emrich T, Graf F, Kriegel HP, Schubert M, Thoma M (2010) Optimizing all-nearest-neighbor queries with trigonometric pruning. In: Proceedings of the 22nd international conference on scientific and statistical database management (SSDBM). Heidelberg, pp 501–518
33. Achtert E, Kriegel HP, Schubert E, Zimek A (2013) Interactive data mining with 3D-Parallel-Coordinate-Trees. In: Proceedings of the ACM international conference on management of data (SIGMOD). New York City, pp 1009–1012
34. Dalvi N, Suciu D (2007) Efficient query evaluation on probabilistic databases. VLDB J 16(4):523–544

Klaus Arthur Schmid received his degree from the LMU Munich, Germany in 2012. His thesis addressed the problem of "Efficient Processing of Probabilistic Histograms over Wireless Sensor Networks". Klaus Arthur Schmid is now a PhD student at the Database Systems Group at the Department of Computer Science of the LMU Munich and has since been a coauthor in nine peer-reviewed publications.

Andreas Zufle is an assistant professor at the department of Geography and Geoinformation Science at George Mason University. He received his PhD in Computer Science and his Diploma in Statistics and Computer Science from LMU Munich, Germany. Dr. Zufle's research quest is to bridge the gap between data-science and geo-science, two fields working independently on often identical research problems. To bring these communities together, Dr. Zufle has been co-organizing and program-cochairing the International ACM SIGMOD Workshop on Managing and Mining Enriched Geo-Spatial Data (GeoRich) for the last three years. His research is focused on spatio-temporal query processing, spatio-temporal data-mining, uncertain data management. Since 2011, he published more than 50 papers in refereed conferences and journals and has an h-index of 14.



Tobias Emrich received a diploma in Computer Science in 2008 and a PhD in Computer Science from LMU Munich in 2013. He was an Academic Assistant at the Computer Science Department at the University of Southern California in 2014. Currently, he is Executive Director of the Data Science Lab at LMU Munich His research interests include querying and mining on spatial, spatio-temporal, uncertain and graph data. To date, he has more than 40 publications in refereed conferences. He has been a member of the program committees in numerous conferences and workshops, and is currently in the steering committee of the "GeoRich" SIGMOD workshop.



Matthias Renz is Associate Professor at the Computational and Data Science Department at the George Mason University (GMU), Fairfax, VA, USA. His main research topics are data science, scientific and spatial databases, data mining and uncertain databases. To date, he has more than 100 peer-reviewed publications that in total received over 1800 citations with an h-index of 19. He served as general chair and program chair for several international conferences including SSTD, ACM SIGSPATIAL, and DASFAA, founded and organized workshops at ACM SIGMOD and ACM SIGPATIAL, gave several invited tutorials, seminars, and keynotes.

Reynold Cheng is an Associate Professor of Computer Science in HKU. He was an Assistant Professor in HKU in 2008-12. He obtained MSc(CS) and PhD from Purdue in 2003 and 2005. In 2005-08, he was an Assistant Professor in HK PolyU. He was granted an Outstanding Young Researcher Award 2011-12 by HKU. He received the 2010 Research Output Prize of HKU CS, and the U21 Fellowship in 2011. He got the Performance Reward in 2006 and 2007 from HK PolyU. He is the MPhil/PhD Programme Director of HKU CS. He is an editorial member of TKDE, IS, and DAPD. He co-chairs SSTD 2013, and has reviewed papers from SIGMOD, VLDB, ICDE, TODS, VLDBJ, TKDE and IS.