Uncertain Voronoi Cell Computation based on Space Decomposition

Tobias Emrich[1], Klaus Arthur Schmid[1], Andreas Züfle[1],
Matthias Renz[1], and Reynold Cheng[2]

[1] Institute for Informatics, Ludwig-Maximilians-Universität München, Germany
{emrich,schmid,zuefle,renz}@dbs.ifi.lmu.de

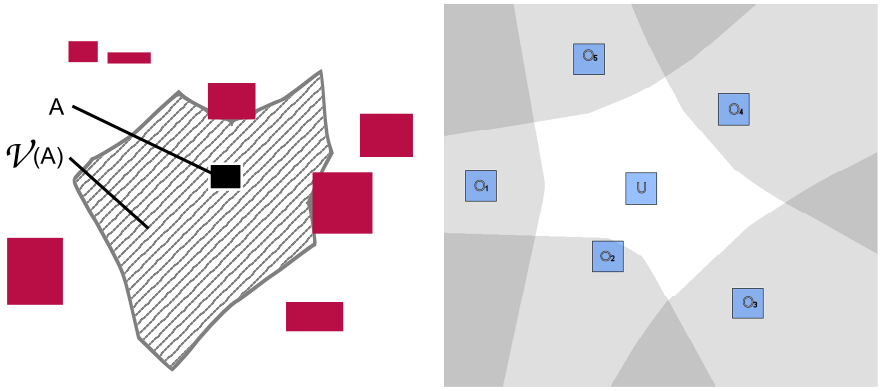
[2] Department of Computer Science, University of Hong Kong, Hong Kong
ckcheng@cs.hku.hk

Abstract. The problem of computing Voronoi cells for spatial objects whose locations are not certain has been recently studied. In this work, we propose a new approach to compute Voronoi cells for the case of objects having rectangular uncertainty regions. Since exact computation of Voronoi cells is hard, we propose an approximate solution. The main idea of this solution is to apply hierarchical access methods for both data and object space. Our space index is used to efficiently find spatial regions which must (not) be inside a Voronoi cell. Our object index is used to efficiently identify Delauny relations, i.e., data objects which affect the shape of a Voronoi cell. We develop three algorithms to explore index structures and show that the approach that descends both index structures in parallel yields fast query processing times. Our experiments show that we are able to approximate uncertain Voronoi cells much more effectively than the state-of-the-art, and at the same time, improve run-time performance.

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 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.



(a) An uncertain Voronoi cell. (b) Spatially Dominated Regions.

Fig. 1. Uncertain Voronoi cells.

– 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 taxi-customer matching, 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* $\mathcal{V}(O)$ of an object O, which is defined as the space where a query point q can possibly have O as its nearest neighbor. Applications for possible Voronoi cells include geo-location-based services, such as taxi assignments: The possible Voronoi cell of an individual taxi cab c covers the space of a city where customers may possibly have c as their nearest taxi. 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].

As an example of a possible Voronoi cell, consider Figure 1(a), where rectangles correspond to the uncertainty regions of objects. The highlighted region corresponds to the subspace $\mathcal{V}(A)$, for which it holds that any point $q \in \mathcal{V}(A)$ may possibly have object A as its nearest neighbor, i.e., the possible Voronoi cell of A. Finding this region, which is the goal of this paper, is not a trivial task: The shape of $\mathcal{V}(A)$ is a non-convex region

which is bounded by hyperbolic curves. As explained in [3, 10, 6], an exact construction of $\mathcal{V}(A)$ requires an exponential amount of time. For this reason, an approximate technique for deriving possible Voronoi cells 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 $\mathcal{V}(o)$ 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.
- 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 [13–15], 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, 3, 6]. 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 [16] . In this context, Voronoi diagrams have been used to support nearest neighbor queries in geo-spatial applications [17], location-based services [18, 19], in spatial data streams [20] and in distributed spatial environments [21] as well as in spatial network environments [22]. To support nearest neighbor queries on uncertain data, initial approaches have been presented in [13, 2]. 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 Figure 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.[3] 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.

3 Problem Definition

Figure 1(b) shows how the possible Voronoi cell $\mathcal{V}(U)$ of an uncertain object U is defined. Each shaded region in Figure 1(b) corresponds to a pruning region $S_A(U)$, i.e., the smallest region such that for any $q \in S_A(U)$, object A must be closer to q than U. 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.[4] 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:*

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

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 [23].

Figure 1(b) shows five nearest neighbor pruning regions $S_{O_1}(U), ..., S_{O_5}(U)$ as shaded regions. Using Definition 1, we can now define the possible Voronoi cell $\mathcal{V}(U)$ of an object U as the space that does not intersect any nearest neighbor pruning region associated with U, formally:

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

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

In Figure 1(b), the white (i.e., non-shaded) region corresponds to the Voronoi cell $\mathcal{V}(U)$. The problem tackled in this paper is to compute $\mathcal{V}(U)$ for a given object $U \in \mathcal{D}$ efficiently.

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

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

Notation	Meaning	Notation	Meaning
$\mathcal{D}$	The Database	$S = \mathcal{R}^d$	d-dimensional data space
$U \in \mathcal{D}$	an uncertain object	$\mathcal{V}(U)$	possible Voronoi cell of U
$\mathcal{I}_\mathcal{D}$	Hierarchical Data Index	$\mathcal{I}_\mathcal{S}$	Hierarchical Space Index
$\mathcal{G}$	d-dimensional grid	$g_i \in \mathcal{G}$	Rectangular Grid Cell
$S_A(B) \subseteq \mathcal{R}^d$	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 [24].		
$PDom(A, B, R)$	Predicate that is true iff rectangle R intersects $S_A(B)$. Can be evaluated efficiently [24].		
$h \subseteq \mathcal{R}^d$	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.		

Table 1. Table of Notations.

4 Spatial Domination revisited

The concept of spatial domination and efficient techniques to verify it were introduced in [24]. 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 3 (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 4 (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 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.*

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

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

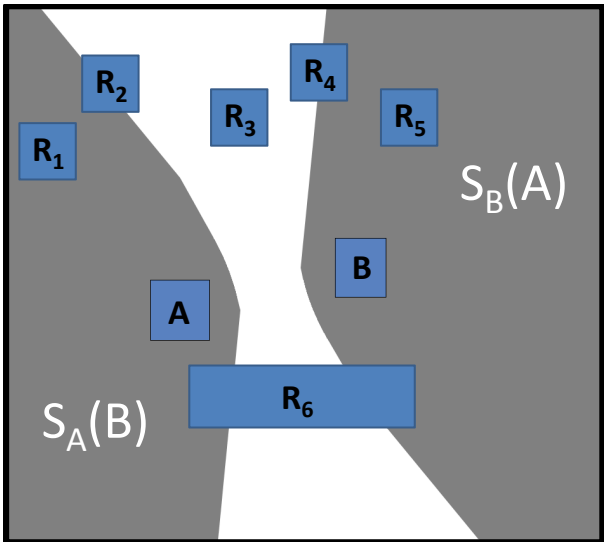


Fig. 2. Domination relation

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 Figure 2. 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 [24]. The following six cases are defined using a function DomCase(A, B, R) as follows.

Definition 5 (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 $Dom(A, B, R)$;*

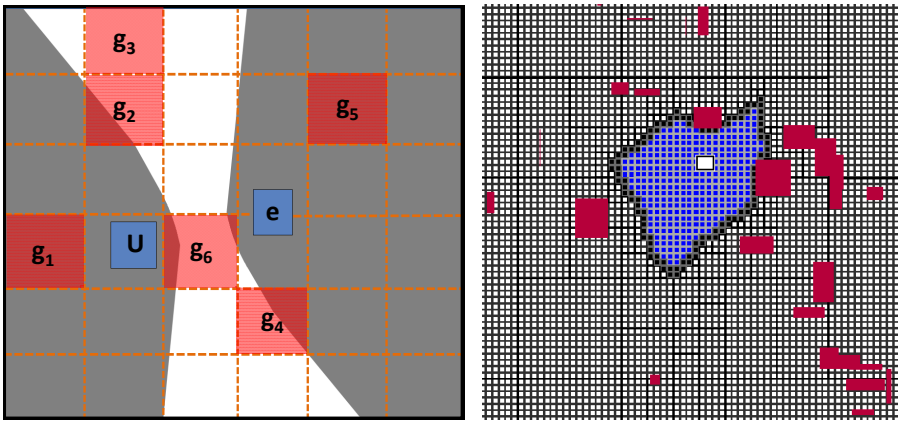
Case 2: *R intersects $S_A(B)$ but not $S_B(A)$ iff $PDom(A, B, R) \wedge \neg PDom(B, A, R)$;*

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 $PDom(A, B, R) \wedge PDom(B, A, R)$;*



(a) Cases of domination of a grid cell. (b) Example result.

Fig. 3. Illustration of the Naive approach.

Figure 2 depicts all possible cases. Here, each rectangle R_i corresponds to Case i in Definition 5. 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 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-Voronoi Cell Approximation

Computing the possible-Voronoi cell 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 $\mathcal{V}(U)$ 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 which can not be part of $\mathcal{V}(U)$ or which must be part of $\mathcal{V}(U)$. 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.

5.1 Naive solution

A straightforward way of computing $\mathcal{V}(U)$ is to apply an equi-distant d-dimensional grid to partition the data space. For each cell g_i we decide weather it belongs to $\mathcal{V}(U)$ or not.

Algorithm The algorithm takes as input the target object U, $\mathcal{D}$ 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_i is part of the UV cell of U, 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 Figure 3(a). To determine if a grid-cell is (i) completely outside of $\mathcal{V}(U)$ or (ii) completely inside $\mathcal{V}(U)$ or (iii) a boarder cell, we can apply the six cases of Definition 5 as follows:

i) If $\exists O \in \mathcal{D} \setminus U : Dom(O, U, g_i)$ then g_i is not part of $\mathcal{V}(U)$. This corresponds to **Case 5** of Definition 5 and cell g_5 in Figure 3(a).
ii) Otherwise, if $\exists O \in \mathcal{D} : PDom(O, U, g_i)$ then at least a small part of g_i can be part of $\mathcal{V}(U)$. This case corresponds to the cases of cells g_4 and g_6 in Figure 3(a), i.e., **Case 4** or **Case 6** of Definition 5.
iii) Otherwise we can conclude that g_i can be completely contained in $\mathcal{V}(U)$, since for database object, U, 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 Figure 3(a)

The set of all grid cells satisfying iii) define a lower bound of $\mathcal{V}(U)$, and all grids cells satisfying ii) or iii) define an upper bound of $\mathcal{V}(U)$. An exemplary result of this approach for a small database of uncertain objects is depicted in Figure 3(b). Here, the space grid is shown, where (i) unfilled cells are guaranteed to be outside of $\mathcal{V}(U)$, (ii) black cells are guaranteed to be on the border of $\mathcal{V}(U)$ and (iii) blue cells are guaranteed to be inside $\mathcal{V}(U)$. In the next subsection, we show how we can obtain this result in a more efficient way. Thus note that the algorithms presented in the following subsections compute the same result approximation, but in a more efficient way.

5.2 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_i the algorithm traverses the entries e of $\mathcal{I}_\mathcal{D}$ in a best first manner [25] 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 5 to decide which index entries have to be accessed. For reference, the following cases are shown in Figure 3(a). Keep in mind that in this case, the entries e are data index entries, which may be intermediate entries representing multiple data objects.

Case 1: $Dom(U, e, g_1)$: e and none of its children can exclude g_1 from the UV-cell $\mathcal{V}(U)$. Thus, e don't has to be resolved and g_1 can be part of $\mathcal{V}(U)$.

Case 2: $PDom(U, e, g_2)$: same as case 1.

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 the UV-cell, thus e has to be resolved. If e is a leaf entry g_3 is labeled as candidate for being part of $\mathcal{V}(U)$

Case 4: $PDom(e, U, g_4)$: same as case 3.

Case 5: $Dom(e, U, g_5)$: g_5 (and all child nodes of g_5) cannot be part of $\mathcal{V}(U)$.

Case 6: $PDom(U, e, g_6) \wedge PDom(e, U, g_6)$: same as case 1.

5.3 Indexing $\mathcal{S}$

Instead of indexing the data objects one could also think of indexing the space containing the grid cells. 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:

1. If $\exists O \in \mathcal{D} : Dom(O, U, h)$ (**Case 5**) then h is not part of the UV cell of U and it does not have to be resolved further.
2. Otherwise if $\exists O \in \mathcal{D} : PDom(O, U, h)$ (**Case 4** or **Case 6**) then at least a small part of h can be part of the UV cell of U. Thus we have to resolve h further. If h is on the $maxdepth$-level we label it as candidate to be part of $\mathcal{V}(U)$.
3. Otherwise (**Cases 1-3**) we can conclude that h can be completely contained in the UV cell of U. In this case we label h as candidate to be part of $\mathcal{V}(U)$ and don't have to resolve it, even if h is not on the $maxdepth$-level.

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

It seems apparent to combine the ideas of Section 5.2 and Section 5.3 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 Figure 4):

Case 5: $\exists e \in AS_\mathcal{D} : Dom(e, U, h) \Rightarrow h$ is not part of the UV cell of U.

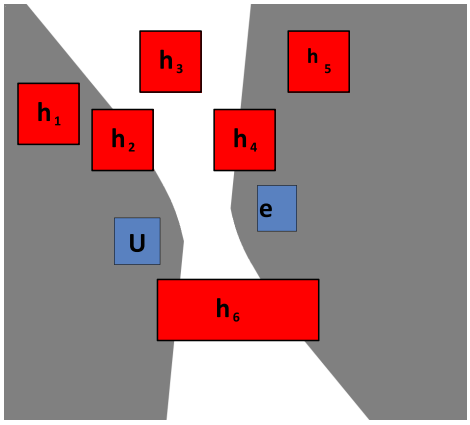


Fig. 4. Cases of domination for a data index entry e.

Case 1: $\forall e \in AS_{\mathcal{D}} : Dom(U, e, h) \Rightarrow h$ can be part of $\mathcal{V}(U)$.

If neither of the two conditions hold, 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 Figure 4, 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 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 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 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), and the active set $AS_\mathcal{D}$ containing a set of $\mathcal{I}_\mathcal{D}$-entries. The algorithm works as follows:

- In a loop (lines 2 - 11)the algorithm first searches for the entry e defining the most prioritized case (8 - 10). Of course we can stop further consideration of h if we find an entry e which defines case 5 (lines 3 - 5). On the other hand side if an entry e defines case 1, it can never disqualify the current h thus can be excluded from $AS_\mathcal{D}$ (lines 6 - 7)
- In lines 12-14 we check if all entries in the active set $AS_\mathcal{D}$ have been pruned. If that is the case, no object may possible prune h and thus h must be a true hit, i.e. fully contained in the Voronoi cell.
- Now we decide whether we want to refine e_{max} or h, depending on the case (c.f. Figure 4 and Definition 5).

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. [5] 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.

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

Algorithm 2 UVCellCheck

Require: U,h,$AS_{\mathcal{D}}$
1: e_{max} *//entry with maximum priority*
2: **for all** $e \in AS_{\mathcal{D}}$ **do**
3: **if** $Dom(e, U, h)$ **then**
4: h is not part of UVCell
5: return
6: **else if** $Dom(U, e, h)$ **then**
7: $AS_{\mathcal{D}} = AS_{\mathcal{D}} \setminus e$
8: **else**
9: $e_{max} = maxprio(e_{max}, e)$
10: **end if**
11: **end for**
12: **if** $AS_{\mathcal{D}}$ is empty **then**
13: h is part of UVCell
14: return
15: **end if**
16: **if** case(e_{max}, U, h) != 6 **then**
17: $AS_{\mathcal{D}} = AS_{\mathcal{D}} \setminus e_{max} \cup e_{max}.children$
18: **end if**
19: *//redundant calculations can be reduced in the following*
20: **if** case(e_{max}, U, h) = 4 or 6 && $\neg$ maxdepth **then**
21: **for all** $h_c \in h.children$ **do**
22: UVCellCheck(U,h_c,$AS_{\mathcal{D}}$.clone())
23: **end for**
24: **else**
25: UVCellCheck(U,h,$AS_{\mathcal{D}}$)
26: **end if**

Figure 5 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 5) holds.

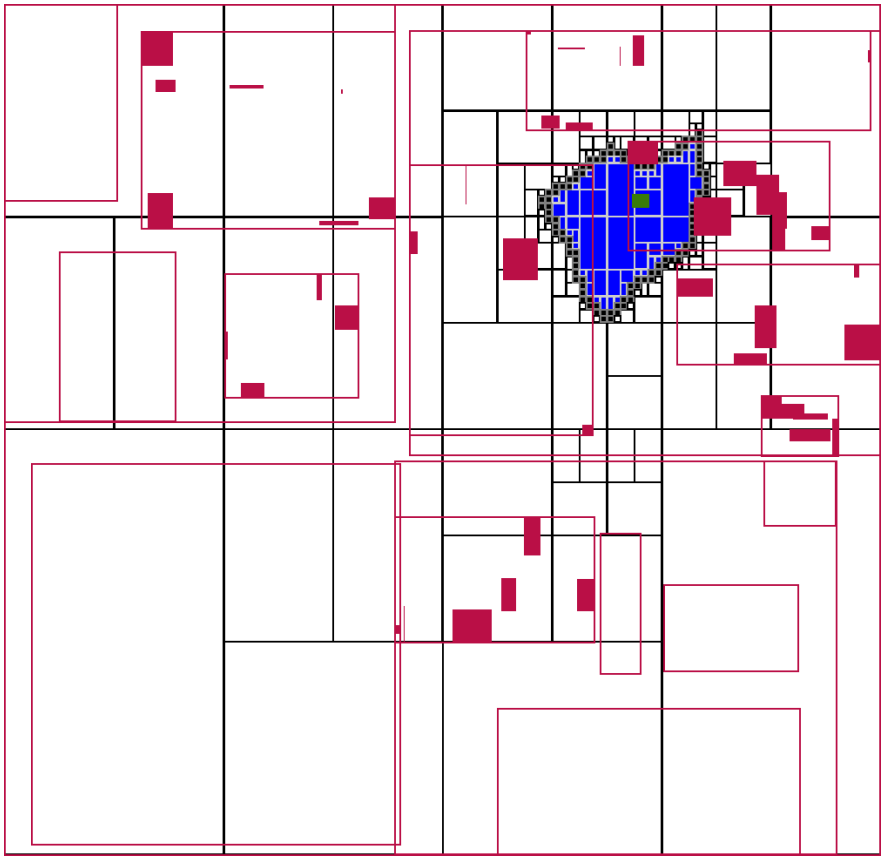


Fig. 5. Example of refinement

6 Experiments

Our experimental evaluation investigates algorithm behaviour w.r.t. maximum kd-trie depth, database size, object extent and dimension. Extent is a parameter to control the size of the uncertain objects (object 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 trajectory dataset[7, 8] which we modified to fit the scope. We implemented all approaches in the ELKI framework[26], which also provided an R-tree implementation.

Dataspace is always normalized to [0,1] per dimension. In synthetic data, objects are uniformly distributed over space with a randomly assigned side length between 0 and maximum extent. 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 (3 to each direction). On said city scale, an extent of 0.01 would equal an area of 100m side length.

Parameter	default value	Notation	Algorithm
Dimension	2	DI	Data Index traversal (Section 5.2)
db size	1000	SI	Space Index traversal (Section 5.3)
extent	0.01	DSI	Data & Space Index traversal (Section 5.4)
tree depth	14	SR	Single Rectangle (Implementation of [6])

Table 2. Default settings.

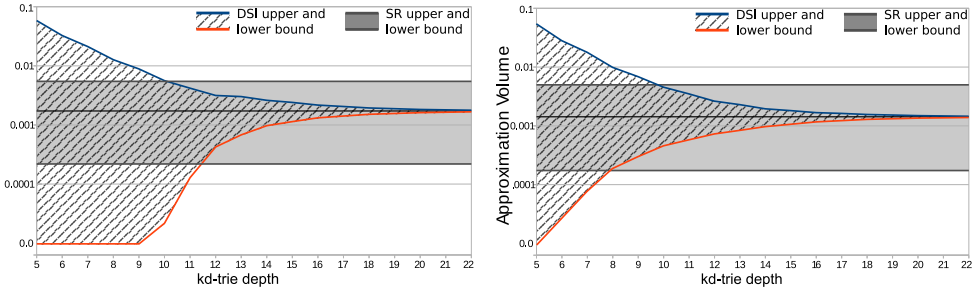
Table 2 denotes input parameters and their default settings, as well as an explanation of our algorithm notation. If not otherwise specified, the following experiments use these input values. Those setups focusing on approximation quality use DSI exemplarily for all algorithms from Sections 5.2–5.4, since result quality is the same. 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.

6.1 Approximation Quality

Our first evaluation explores how well the generated bounds approximate a cell. For this, 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 6 visualizes how upper and lower bounds converge with higher tree granularities. 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 tree granularities.

Performance was tested on different datasets. Figure 6(a) represents average results for runs on synthetic data, while Figure 6(b) 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. SR itself shows fairly similar behaviour on both datasets, with results looking even more similar than they are due to logarithmic scale.

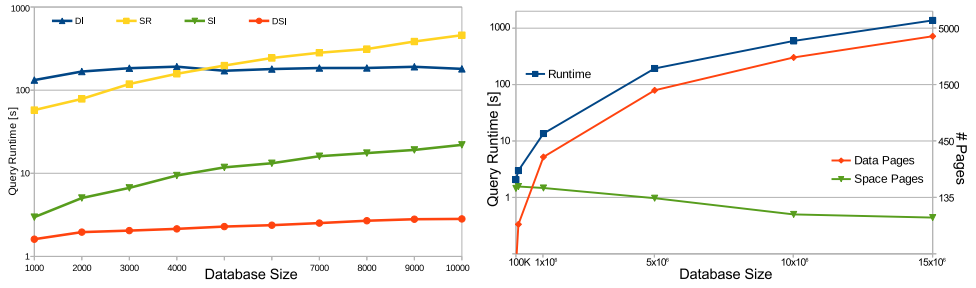


(a) Synthetic Dataset (b) Real-world Dataset

Fig. 6. Approximation Quality for DSI and SR

6.2 Algorithmic Runtime

Runtime experiments were conducted while modifying database population and dimensionality, between our three different traversal approaches compared to SR as well as for DSI alone to cover larger ranges of database size (others have been excluded due to their worse performance). Although the taxi dataset is not applicable here since we modify parameters that are inherent to specific datasets, the semantics still stand: inserting more objects into a database of the same geometric expansion could represent offering more taxis for hire in a city, hence changing the nearest neighbor situation in most of the places. Therefore, the maximum object extent remained unchanged for all database sizes, since obfuscation of one's location is independent of the world's object density.



(a) All Approaches Compared (b) DSI (Data & Space)

Fig. 7. A runtime comparison for all algorithms over different sizes of DB

In Figure 7, run times to calculate one UV-cell are denoted over different database sizes. Figure 7(a) contains results for the approaches *Dataindex Traversal (DI)*, *Spaceindex traversal (SI)*, *Data and Space Index Traversal (DSI)* and SR. Note how DI shows a relatively constant, high runtime since for each query, every grid cell g_i is explored, independently of database population. SR starts off better, but since it features pairwise

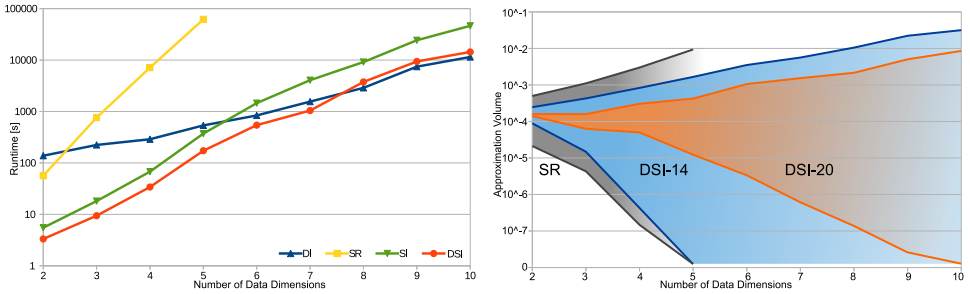
comparisons without the use of an index, it does not scale well for higher numbers ob database objects. SI clearly shows how such an index improves performance drastically, but also scales up rather fast. DSI also increases in runtime for higher dimensional datasets, but at generally much lower absolute values than the other approaches. Also, DSI increases at a lower rate. This is because the combined approach of data and space index allows for early pruning of large portions of the database.

As query performance generally deteriorates for larger datasets (or remains at high values in the case of DI), further scaling experiments were conducted using DSI only. Figure 7(b) 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 likelyhood of all relevant objects being enclosed in a small space increases.

6.3 Effect of data dimensions

Although the trivial 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 computational complexity.



(a) Runtime (b) Approximation Quality

Fig. 8. A comparison for increasing data dimensions.

Figure 8 displays performance of all approaches 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 Figure 8(a) 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.

Approximation quality for higer dimensions is shown in Figure 8(b). 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 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.

6.4 Conclusions

In this work, we proposed an index-supported approach to approximate the shape of a possible Voronoi-cell to support nearest neighbor queries on uncertain data. Our approache 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. As future work, we want to extend our ideas to find *certain Voronoi-cells*, that is regions, where a query object has a probability of one of having some object U as its nearest neighbor. Furthermore, we want to extend our solution to the case of k'th-order Voronoi-cells to support k-nearest neighbor queries. Even in the case of certain data, k'th-order Voronoi-cells become complexly shaped, having a representation complexity exponential in k. However, since we are using space approximation techniques, rather than computing exact bounds, we can avoid this computational drawback.

Acknowledgements

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)).

References

1. Chow, C.Y., Mokbel, M.F., Aref, W.G.: Casper*: Query processing for location services without compromising privacy. ACM TODS **34**(4) (2009) 24
2. Beskales, G., Soliman, M.A., Ilyas, I.F.: Efficient search for the top-k probable nearest neighbors in uncertain databases. VLDB Endowment **1**(1) (2008) 326–339

3. Cheng, R., Xie, X., Yiu, M.L., Chen, J., Sun, L.: Uv-diagram: A voronoi diagram for uncertain data. In: ICDE, IEEE (2010) 796–807
4. Ali, M.E., Tanin, E., Zhang, R., Kotagiri, R.: Probabilistic voronoi diagrams for probabilistic moving nearest neighbor queries. DKE **75** (2012) 1–33
5. Bernecker, T., Emrich, T., Kriegel, H.P., Mamoulis, N., Renz, M., Züfle, A.: A novel probabilistic pruning approach to speed up similarity queries in uncertain databases. In: Proc. ICDE. (2011) 339–350
6. Zhang, P., Cheng, R., Mamoulis, N., Renz, M., Zufle, A., Tang, Y., Emrich, T.: Voronoi-based nearest neighbor search for multi-dimensional uncertain databases. In: ICDE, IEEE (2013) 158–169
7. Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G., Huang, Y.: T-drive: Driving directions based on taxi trajectories. In: SIGSPATIAL. (2010) 99–108
8. Yuan, J., Zheng, Y., Xie, X., Sun, G.: Driving with knowledge from the physical world. In: SIGKDD. (2011) 316–324
9. Emrich, T., Kriegel, H.P., Mamoulis, N., Renz, M., Züfle, A.: Querying uncertain spatio-temporal data. In: ICDE, IEEE (2012) 354–365
10. Emrich, T., Kriegel, H.P., Kröger, P., Renz, M., Züfle, A.: Incremental reverse nearest neighbor ranking in vector spaces. In: SSTD. Springer (2009) 265–282
11. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. Volume 19. ACM (1990)
12. Orenstein, J.A., Merrett, T.H.: A class of data structures for associative searching. In: ACM SIGACT-SIGMOD, ACM (1984) 181–190
13. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Querying imprecise data in moving object environments. In: IEEE TKDE. (2004)
14. Li, J., Saha, B., Deshpande, A.: A unified approach to ranking in probabilistic databases. VLDB Endowment **2**(1) (2009) 502–513
15. Bernecker, T., Kriegel, H.P., Mamoulis, N., Renz, M., Zuefle, A.: Scalable probabilistic similarity ranking in uncertain databases. TKDE **22**(9) (2010) 1234–1246
16. Aurenhammer, F.: Voronoi diagrams-a survey of a fundamental geometric data structure. ACM CSUR **23**(3) (1991) 345–405
17. Sharifzadeh, M., Shahabi, C.: Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries. VLDB Endowment **3**(1-2) (2010) 1231–1242
18. Zheng, B., Xu, J., Lee, W.C., Lee, L.: Grid-partition index: a hybrid method for nearest-neighbor queries in wireless location-based services. VLDB Journal **15**(1) (2006) 21–39
19. Nutanong, S., Zhang, R., Tanin, E., Kulik, L.: The v*-diagram: a query-dependent approach to moving knn queries. VLDB Endowment **1**(1) (2008) 1095–1106
20. Sharifzadeh, M., Shahabi, C.: Approximate voronoi cell computation on spatial data streams. VLDB Journal **18**(1) (2009) 57–75
21. Akdogan, A., Demiryurek, U., Banaei-Kashani, F., Shahabi, C.: Voronoi-based geospatial query processing with mapreduce. In: IEEE CloudCom, IEEE (2010) 9–16
22. Kolahdouzan, M., Shahabi, C.: Voronoi-based k nearest neighbor search for spatial network databases. In: VLDB Endowment, VLDB Endowment (2004) 840–851
23. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: ACM SIGMOD. Volume 24., ACM (1995) 71–79
24. Emrich, T., Kriegel, H.P., Kröger, P., Renz, M., Züfle, A.: Boosting spatial pruning: On optimal pruning of MBRs. In: Proc. SIGMOD. (2010) 39–50
25. Hjaltason, G.R., Samet, H.: Ranking in spatial databases. In: Proc. SSD. (1995) 83–95
26. Achtert, E., Kriegel, H.P., Schubert, E., Zimek, A.: Interactive data mining with 3D-Parallel-Coordinate-Trees. In: Proc. SIGMOD. (2013) 1009–1012