# Probabilistic Range Query over Uncertain Moving Objects in Constrained Two-Dimensional Space

Zhi-Jie Wang, Dong-Hua Wang, Bin Yao, and Minyi Guo, *Senior Member, IEEE*

**Abstract**—Probabilistic range query (PRQ) over uncertain moving objects has attracted much attentions in recent years. Most of existing works focus on the PRQ for objects moving freely in two-dimensional (2D) space. In contrast, this paper studies the PRQ over objects moving in a constrained 2D space where objects are forbidden to be located in some specific areas. We dub it the constrained space probabilistic range query (CSPRQ). We analyze its unique properties and show that to process the CSPRQ using a straightforward solution is infeasible. The key idea of our solution is to use a strategy called *pre-approximation* that can reduce the initial problem to a highly simplified version, implying that it makes the rest of steps easy to tackle. In particular, this strategy itself is pretty simple and easy to implement. Furthermore, motivated by the cost analysis, we further optimize our solution. The optimizations are mainly based on two insights: (i) the number of *effective subdivision*s is no more than 1; and (ii) an entity with the larger *span* is more likely to subdivide a single region. We demonstrate the effectiveness and efficiency of our proposed approaches through extensive experiments under various experimental settings, and highlight an extra finding—the precomputation based method suffers a non-trivial preprocessing time, which offers an important indication sign for the future research.

**Index Terms**—Constrained space, probabilistic range query, uncertain moving objects, obstacles, query processing

✦

## 1 INTRODUCTION

R ANGE query for moving objects has been the subject of much attentions [7], [10], [14], [15], [17], [24], [26], [28], as it can find applications in various domains such as the digital battlefield, mobile workforce management, and transportation industry. It is usual that for a moving object $o$, only the discrete location information is stored on the database server, due to various reasons such as the limited battery power of mobile devices and the limited network bandwidth [4]. The recorded location of $o$ can be obtained by accessing the database, the whereabouts of its current location is usually uncertain [25]. For example, a common location update policy called *dead reckoning* [4], [25] is to update the recorded location $l_r$ when the deviation between $l_r$ and the actual location of $o$ is larger than a given distance threshold $\tau$. Before the next update, the specific location of $o$ is uncertain, except knowing that it lies in a circle with the center $l_r$ and radius $\tau$. To capture the location uncertainty, the idea of incorporating *uncertainty* into moving objects data has been proposed [25]. From then on, the probabilistic range query (PRQ) as a variant of the traditional *range query* has attracted much attentions in the data management community [3], [4], [5], [16], [19], [21], [22], [27]. A well known

uncertainty model is using a closed region (in which the object can always be found) together with a probability density function (PDF). The closed region is usually called *uncertainty region*, and the PDF is used to denote object's *location distribution* [3], [4], [25]. (See Section 2 for a more formal definition.) Given a query range $R$, the main difference between the traditional range query and the PRQ is that the latter returns not only the objects being located in $R$ but also their appearance probabilities. Assume that the location of object $o$ follows uniform distribution in its uncertainty region $u$ for ease of discussion, the probability of object $o$ being located in $R$ is equal to the ratio of the two areas, i.e., the probability $p = \frac{area\ of\ u \cap R}{area\ of\ u}$. Fig. 1a illustrates an example and the PRQ returns { $(o, 39\%)$ }.

In existing works, an important branch is to address the PRQ for objects moving freely in 2D space. In this branch, many *uncertainty models* and techniques are proposed for various purposes. (Section 2.1 gives a brief survey about those models, purposes and techniques.) Surprisingly, little efforts are made for the PRQ over objects moving in a constrained 2D space where objects are forbidden to be located in some *specific areas*. For clarity, we term such specific areas as *restricted areas*, and dub the query above the Constrained Space Probabilistic Range Query (CSPRQ). The CSPRQ can also find many applications as objects moving in a constrained 2D space are common in the real world. For example, the tanks in the digital battlefield usually cannot run in lakes, forests and the like, the areas occupied by those obstacles can be naturally regarded as *restricted areas* (of tanks). With similar observations, in a zoo, tourists usually cannot roam in the dwelling spaces of dangerous animals such as tigers and lions, those dwelling spaces can be regarded as the *restricted areas* (of tourists).

- *Z.-J. Wang, B. Yao, and M. Guo are with the Shanghai Key Laboratory of Scalable Computing and Systems, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: zjwang888@sjtu.edu.cn, {yaobin, guo-my}@cs.sjtu.edu.cn.*
- *D.-H. Wang is with the Research and Development Department, NetEase, Inc., Guangzhou 510665, Guangdon, China. E-mail: alex_wang@126.com.*
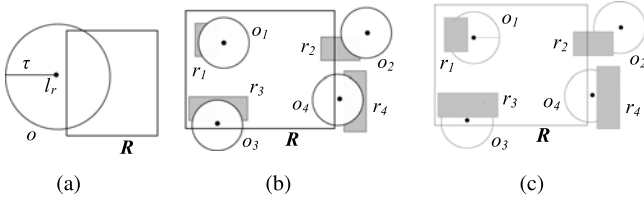
Fig. 1. Illustration of the PRQ and CSPRQ. The small black dot denotes the recorded location $l_r$, the radius of circle denotes the distance threshold $\tau$, the biggest rectangle denotes query range $R$, the small rectangle denotes the restricted area $r$.

Existing solutions cannot be directly applied to the CSPRQ as it involves a set $\mathcal{R}$ of *restricted areas*. Imagine if we directly use existing methods, implying that we ignore each restricted area $r$ ($\in \mathcal{R}$) in the computation phase. Fig. 1b depicts this case, the circle $o.\odot$ is regarded as the uncertainty region $u$, and the query answer is {($o_1$, 100%), ($o_3$, 56%), ($o_4$, 42%)}. In contrast, Fig. 1c presents the case considering $\mathcal{R}$ in the computation phase, here $o.\odot - \bigcup_{r \in R} r$ is regarded as $u$, and the query answer is { ($o_1$, 100%), ($o_3$, 22%), ($o_4$, 76%)}. The two answers above are different, and clearly the second one is correct. At first sight, to process the CSPRQ is simple as it seems to be a straightforward adaptation of existing methods. The fact however is not so, as this idea will be confronted with the overcomplicated geometrical operations, rendering its implementation infeasible. (Section 3 gives more detailed explanations.) In addition, computing the uncertainty region $u$ is also not a simple subtraction operation, as a straightforward computation incurs possible mistakes. On the other hand, the CSPRQ needs to consider a new set $\mathcal{R}$ compared to the previous works, it implies that the amount of data to be processed is larger and the computation is more complicated, which is another challenge and thus needs more considerations.

Motivated by the fact above, this paper makes the effort to the CSPRQ. The key idea of our solution is to use a strategy called *pre-approximation* that can reduce the initial problem to a highly simplified version, implying that it makes the rest of steps easy to tackle. In particular, this strategy itself is pretty simple and easy to implement. To operate different entities in a unified and efficient manner, a *label based data structure* is developed. Ascribing the pre-approximation and label based data structure, it is pretty simple to compute the appearance probability. To improve the I/O efficiency, a twin index is naturally adopted. Furthermore, motivated by the cost analysis, we further optimize our solution. The optimizations are mainly based on two insights: (i) the number of *effective subdivision*s is no more than 1, we utilize this insight to improve the power pruning restricted areas; and (ii) an entity with the larger *span* is more likely to subdivide a single region, this insight motivates us to sort the entities to be processed according to their spans. In addition to the main insights above, we also realize two other (simple but usually easy to ignore) facts and utilize them. Specifically, two mechanisms are developed: *postpone processing* and *lazy update*. After we finish the main tasks of this work, we also attempt another approach inspired by the curiosity, its basic idea is to precompute uncertainty regions and index them. Unfortunately, this approach suffers a non-trivial preprocessing time although it outperforms the aforementioned approaches in terms of

both query and I/O performance. This extra finding offers us an important indication sign for the future research. In summary, we make the following contributions.

- We formulate the CSPRQ based on an *extended* uncertainty model, and analyse its unique properties.
- We show a straightforward solution will be confronted with non-trivial troubles, rendering its implementation infeasible. We also show it is (almost) infeasible to develop an exact solution.
- We propose our solution that utilize an (extremely) important but pretty simple strategy.
- We further optimize our solution based on two insights and two (simple but usually easy to ignore) facts.
- We demonstrate the performance of our solution through extensive experiments under various experimental settings.
- We report an extra finding that offers an important indication sign for the future research.

In the next section we formulate the problem to be studied and review the related work. We analyse this problem and propose our solution in Sections 3 and 4, respectively. We further optimize our solution in Section 5. We attempt the precomputation based approach in Section 6. We evaluate the efficiency and effectiveness of our proposed algorithms through extensive experiments in Section 7. Finally, Section 8 concludes this paper with several interesting research topics.

## 2 PROBLEM DEFINITION

Given a territory $\mathbb{T}$ with a set $\mathcal{R}$ of disjoint restricted areas, we assume there exist a set $\mathcal{O}$ of moving objects that can freely move in $\mathbb{T}$ but cannot be located in any restricted area $r$ ($\in \mathcal{R}$), and assume the last sampled location of each moving object $o$ is already stored on the database server. (Note that in this paper the terms the *last sampled location* and *recorded location* are used interchangeably.) Moreover, suppose each object $o$ reports its new location to the sever when the deviation between the recorded location $l_r$ and the actual location of $o$ is larger than a given distance threshold $\tau$. We denote the location of $o$ at an arbitrary instant of time $t$ by $l_t$. Furthermore, for any two different moving objects $o$ and $o'$, we assume they cannot be located in the same location at the same instant of time $t$, i.e., $l_t \neq l'_t$. Since the realistic application environment varies from place to place, the shapes of restricted areas should be diversified, whereas our objective is to establish a general approach instead of focusing on certain specific environment. Therefore throughout this paper we use polygons to denote the restricted areas (note: this assumption is feasible, since any shaped area can be transformed into polygon shaped area beforehand). Finally, we set the following conditions are always satisfied:

$$\begin{cases} l_t \notin \bigcup_{r \in \mathcal{R}} r & \text{(1a)} \\ l_t \in \mathbb{T} - \bigcup_{r \in \mathcal{R}} r & \text{(1b)} \\ \bigcup_{r \in \mathcal{R}} r \subset \mathbb{T}. & \text{(1c)} \end{cases}$$

The specific location of $o$ at the current time is usually uncertain, a well known model [4], [25] allows us to capture the location uncertainty of $o$ through two components:

**Definition 1 (Uncertainty region).** *The uncertainty region of a moving object $o$ at a given time $t$, denoted by $u^t$, is a closed region where $o$ can always be found.*

**Definition 2 (Uncertainty probability density function).** *The uncertainty probability density function of a moving object $o$ at a given time $t$, denoted by $f^t(x,y)$, is the PDF of $o$'s location at the time $t$. Its value is 0 if $l_t \notin u^t$.*

Note that under the distance based update policy (a.k.a. dead-reckoning policy [4]), for any two different time $t_1$ and $t_2$, we have (i) $u^{t_1} = u^{t_2}$ and (ii) $f^{t_1}(x,y) = f^{t_2}(x,y)$, where $t_1, t_2 \in (t_r, t_n]$, $t_r$ refers to the latest reporting time, and $t_n$ refers to the current time. In view of these, in the remainder of the paper we use $u$ and $f(x,y)$ to denote the uncertainty region and PDF of $o$, respectively. Since $f(x,y)$ is a PDF, in theory, it has the property:

$$\int_u f(x,y)\,dxdy = 1. \qquad (2)$$

Under the distance based update policy, the uncertainty region $u$ can be derived based on the following formula [4], [25].

$$u = C(l_r, \tau), \qquad (3)$$

where $C(\cdot)$ denotes a circle with the centre $l_r$ and radius $\tau$. For convenience, we use $o.\odot$ to denote this region. The above representation is feasible under the case no restricted areas exist, i.e., $\mathcal{R} = \emptyset$. Whereas the real uncertainty region $u$ for our problem should be as follows:

$$u = o.\odot - \bigcup_{r \in \mathcal{R}} r. \qquad (4)$$

**Definition 3 (Constrained space probabilistic range query).** *Given a set $\mathcal{R}$ of restricted areas and a set $\mathcal{O}$ of moving objects in a territory $\mathbb{T}$, and a query range $R$, the constrained space probabilistic range query returns a set $\mathcal{O}'$ $(\subseteq \mathcal{O})$ of objects together with their appear probabilities in form of $(o, p)$ such that for any $o \in \mathcal{O}'$, $p \neq 0$, where $p$ is the probability of $o$ being located in $R$, and is computed as $p = \int_{u \cap R} f(x,y)\,dxdy$.*

Note that in this paper we assume the distance based update policy is adopted. We abuse the notation '$|\cdot|$' but its meaning should be clear from the context. In addition, a notation or symbol with the subscript 'b' usually refers to its corresponding MBR (e.g., $o.\odot_b$ refers to the MBR of $o.\odot$). For ease of reading, we summarize the frequently used symbols in Table 1.

## 2.1 Related Work

In terms of probabilistic range query over uncertain moving objects, researchers have made considerable efforts, and many outstanding techniques and models have been proposed. In this section, we review those works most related to ours.

The uncertainty model used in this paper is developed based on [4], [25]. In their papers, a moving object $o$ updates

TABLE 1
Symbols and Their Meanings

| Symbols | Description |
|---------|-------------|
| $R$ | query range |
| $o$ | moving object |
| $\mathcal{O}$ | the set of moving objects |
| $r$ | restricted area |
| $\zeta$ | the number of edges of $r$ |
| $\mathcal{R}$ | the set of restricted areas |
| $\tau$ | distance threshold |
| $l_r$ | the recorded location of $o$ |
| $f(x,y)$ | PDF of $o$'s location |
| $p$ | probability of $o$ being located in $R$ |
| $u$ | uncertainty region |
| $s$ | the intersection result between $R$ and $u$ |
| $u_o$ | the outer ring of $u$ |
| $\varphi$ | the intersection result between $R$ and $u_o$ |
| $u_h$ | the hole of $u$ |
| $\mathcal{H}$ | the set of holes in $u$ |
| $\mathcal{O}^*$ | the set of candidate moving objects |
| $\mathcal{R}^*$ | the set of candidate restricted areas |
| $e$ | the approximated equilateral polygon from $o.\odot$ |
| $\xi$ | the number of edges of $e$ |
| $d$ | a subdivision |
| $d^e$ | the effective subdivision |

its recorded location $l_r$, when the deviation (between its actual location and $l_r$) is larger than a given distance threshold $\tau$. This update policy is just the so-called *distance based update policy*.[1] In particular, they discussed two types of moving objects: (i) moving on predefined routes, and (ii) moving freely in 2D space. For the former, the route consists of a series of line segments, the uncertainty is a line segment on the route, called line segment uncertainty (LSU) model for convenience. For the latter, the route is unneeded, and the uncertainty used in their paper is a circle, called free moving uncertainty (FMU) model. Our model roughly follows the latter. The difference is that our model introduces the restricted areas, and the uncertainty region $u$ is not necessarily a circle. (Although only a slight difference viewed from the surface, the amount of data to be processed in our query however is larger, and the computation is more complicated. In particular, a straightforward adaptation of their method will incur overcomplicated geometrical operations, rendering its implementation infeasible.)

In addition, the models in [3], [20] are the same or similar as the FMU model, and also focus on the case of no restricted areas. For example, Tao et al. [20] investigated range query on multidimensional uncertain data, they proposed a classical technique PCR, and an elegant indexing mechanism U-tree. They adopted a circle to represent the uncertainty region $u$ (see Section 7 in [20]). Chen and

---

1. We also assume this update policy is adopted in our work. Another common location update policy is the time based update, i.e. updating the recorded location $l_r$ periodically (e.g., every 3 minutes). The CSPRQ is more challenging if the time based update policy is assumed to be adopted, as it needs more considerations on the time dimension and usually needs other assumptions (e.g., the velocity of object should be available). In addition, the space dimension should be more difficult to handle, as the uncertainty region $u$ is to be a continuously changing geometry over time. We leave this interesting topic as the future work, and we believe this paper will lay a foundation for the future research.

Cheng [3] addressed *location based* range query. Several clever ideas such as query expansion and query duality were proposed. They discussed two types of *target* objects: static and moving. They assume the uncertainty region $u$ is a rectangle when the *target* object is moving. (Note: our work does not belong to *location based* query. Location based CSPRQ should be more interesting, as the location of query issuer is also uncertain.)

Regarding to the case of objects moving freely in 2D space, there are many other classical uncertainty models like, the MOST model [19], the UMO model [27], the 3D cylindrical (3DC) model [16], [22], and the necklace uncertainty (NU) model [11], [21]. These models have different assumptions and purposes, but also their own advantages (note: it is a difficult task to say which one is the best). The models in [19], [27] are developed for querying the future location. For example, Sistla et al. [19] proposed the MOST model, they assume the direction and speed of each object $o$ are available, and these information should be updated if the change occurs. The future location is predicted based on three parameters: velocity, direction, and time. Later, Zhang et al. [27] proposed the UMO model, in which they use the distribution of location and the one of velocity, instead of the exact values, to characterize the location uncertainty, and assume these distributions are available at the update time. The models in [11], [16], [21], [22] are suitable for querying the trajectories of moving objects. For example, Trajcevski et al. [22] proposed to model an uncertain trajectory (UT) as a 3D cylindrical body, they assume an electrical map, all recorded locations and sampling time are available. Later, they proposed the NU model [21], which can be viewed as an enhanced version of the 3DC model. In this model, they represent the whereabouts in-between two known locations as a bead, and an uncertain trajectory as a necklace (a sequence of beads). Our work is different from aforementioned works in at least two points: (i) those works focus on the case of no restricted areas, and (ii) the underlying uncertainty model is different from theirs. (Note: it should be more interesting to extend the concept of restricted areas to those uncertainty models.)

Recently, Emrich et al. [6] proposed to model the trajectories of moving objects by stochastic processes, they assume the object is in a *discrete state space* (i.e., a finite set of possible locations in space), and assume the transition probability (from a state to another state) is available. Our work is different form theirs in two points at least: (i) the underlying models are different, and (ii) the object discussed in our paper is not in a discrete state space.

Another important branch is to focus on objects moving on predefined routes (or road networks) [5], [29]. For example, Chung et al. [5] adopted the LSU model to process range query, and proposed a clever idea—transforming the uncertain movements of objects into points in a dual space using the Hough Transform. To query the trajectories of objects moving on road networks, Zheng et al. [29] proposed the uncertain trajectory model and an elegant indexing mechanism UTH. They assume all recorded locations and sampling time are available, and objects follow the shortest paths and travel at a constant speed between two consecutive trajectory samples. Our work is different from works mentioned above, as here we focus on objects moving in a constrained 2D space where no predefined routes are given.

## 3 PROBLEM ANALYSIS

At first sight, to process the CSPRQ is simple as it seems to be a straightforward adaptation of existing methods. To process the PRQ, existing methods (see e.g., [4]) consist of three main steps.

1) For each object $o$, it computes $u \cap R$ if the uncertainty region $u$ intersects with the query range $R$.
2) It computes the probability $p$ based on a formula $p = \int_{u \cap R} f(x, y) dx dy$, and put the tuple $(o, p)$ into the result.
3) It returns the result (which usually includes a series of tuples) after all objects are processed.

By the large, we only need to add *one* step, i.e., computing the uncertainty region $u$ based on Equation (4) before checking if $u$ intersects with $R$. In other words, this straightforward method mainly consists of four steps. Now the readers should be pretty curious—why the four steps above cannot be (easily) achieved. We next look a bit deeper into those steps above, and then we can easily realize four main issues (but not limited to) arise.

First, suppose the location of an object $o$ follows uniform distribution in its uncertainty region $u$, the following equation holds [16]:

$$ p = \frac{\Lambda(u \cap R)}{\Lambda(u)}, \tag{5} $$

where $\Lambda(\cdot)$ denotes the area of the geometrical entity. Let $s$ be the *intersection result* of $u \cap R$. It is easy to know that computing the area of $u$ (or $s$) is simple for the case of no restricted areas. To the case of our concern, e.g., see Fig. 2, how to compute the area of $u$ (or $s$)? Computing the area of such complicated entity is not an easy task, as its boundary consists of both straight line segments and curves, and it includes many holes. (In fact, $s$ possibly consists of multiple subdivisions in addition to holes. Those even more complicated cases will be discussed in Section 5.) A natural method could be to divide the entity into multiple small strips shown in Fig. 3a, and then to compute the area of each strip and add them together. In practice, this solution however, is overcomplicated and difficult to implement.

Second, suppose the location of $o$ does not follow uniform distribution in $u$, a usually used method is the Monte Carlo method. Its basic idea is to randomly generate $N_1$ points in $u$. For each generated point $p'$, it computes $f(x_i, y_i)$, where $(x_i, y_i)$ is the coordinates of the point $p'$, and then checks whether or not $p' \in s$. Without loss of generality, suppose $N_2$ points (among $N_1$ points) are to be located in $s$. Finally, it gets the probability $p$ as follows:

$$ p = \frac{\sum_{i=1}^{N_2} f(x_i, y_i)}{\sum_{i=1}^{N_1} f(x_i, y_i)}. \tag{6} $$

Given a randomly generated point $p'$, to check whether or not $p' \in u$ (or $p' \in s$) is simple if no restricted areas exist. However, it is not an easy task for the case of our concern. Note that the solutions to the *point in polygon problem* [1], [9]
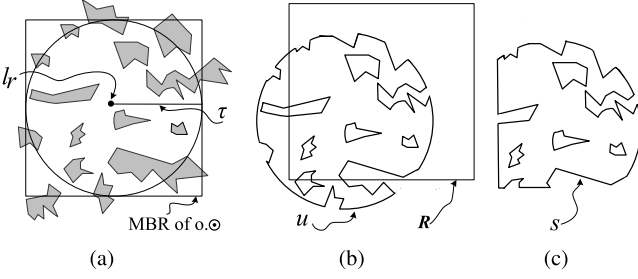
Fig. 2. Illustration of a straightforward solution. (a) The grey polygon illustrates the restricted area $r$, the circle illustrates $C(l_r, \tau)$, i.e., $o.\odot$. (b) The biggest rectangle illustrates the query range $R$, the pseudo circle illustrates the uncertainty region $u$. (c) It is the intersection result of $u \cap R$.



Fig. 3. Illustration of computing the area and uncertainty region $u$. The grey polygon illustrates the restricted area $r$, the circle illustrates $C(l_r, \tau)$, i.e., $o.\odot$.

cannot be applied to our context as geometrical entities considered here are more complicated.

Third, it is easy to know that both $u$ and $s$ are pretty simple if no restricted areas exist. Geometrical entities in our context however are more complicated. Then, how to represent and operate them in a concise and efficient way?

Fourth, computing $u$ is not a straightforward subtraction operation. Fig. 3b illustrates the case before executing the subtraction operation, and the subtraction result is shown in Fig. 3c, which has four subdivisions. Note that, only $d_2$ is the real uncertainty region, other subdivisions are invalid, the reason will be explained in the next section.

Besides the issues mentioned above, we should note that the amount of data to be processed is larger compared to the case of no restricted areas. It is easy to know that the PRQ only needs to check $O(|\mathcal{O}|)$ objects. In contrast, the CSPRQ needs to check $O(|\mathcal{R}|)$ restricted areas for each object $o$, the (worst case) complexity is $O(|\mathcal{R}||\mathcal{O}|)$.

*Discussion.* The above analysis offers insights into our problem, it reveals to us that to process the CSPRQ using a straightforward solution is infeasible. Furthermore, even if the location of $o$ follows uniform distribution in $u$, it is still non-trivial to develop an exact solution (let alone the non-uniform distribution case), implying that to develop an exact solution is also (almost) infeasible. After we realize the facts above, we also note that another easily brought to mind method that is to approximate the curves on the boundary of $u$ (or $s$) into line segments. In this way, the troubles shown before seemingly can be tackled easily. In fact, existing *curve interpolation techniques* can indeed transform the boundary of $u$ (or $s$) into line segments. It however is still inconvenient and inefficient, since there are too many such entities in the query processing. In addition, it is also difficult and troublesome to approximate curves into line segments in such a manner, as the shapes of different entities vary from one to another.

## 4   OUR SOLUTION

The key idea of our solution is to use a strategy called *pre-approximation* that can lead to a highly simplified version of the initial problem, implying that it can make the rest of steps easy to tackle. In particular, this strategy itself is pretty simple and easy to implement.

### 4.1   Pre-Approximation
The essence of the pre-approximation is that, it first transforms (or approximates) $o.\odot$ into an equilateral polygon

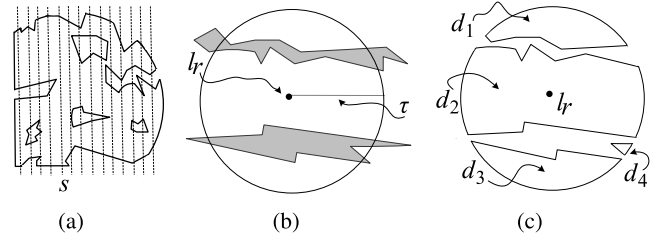denoted by $e$, and then uses $e$ to subtract the restricted areas. Thus, according to Equation (4), we have

$$u \doteq e - \bigcup_{r \in \mathcal{R}} r. \qquad (7)$$

To get $e$ is pretty simple, without loss of generality, assume that we need to approximate $o.\odot$ into an equilateral polygon $e$ with a number $\xi$ of edges. We actually only need to obtain each vertex of $e$, which can be computed based on the following equations:

$$
\begin{cases}
x_i = l_r.x + \tau \cdot \cos \dfrac{(i-1) \cdot 2\pi}{\xi} & (8a) \\[2mm]
y_i = l_r.y + \tau \cdot \sin \dfrac{(i-1) \cdot 2\pi}{\xi}, & (8b)
\end{cases}
$$

where $i \in [1, 2, \ldots, \xi]$, $(l_r.x, l_r.y)$ denote the coordinates of the recorded location $l_r$, and $(x_i, y_i)$ denote the coordinates of the $i$th vertex of $e$.

Clearly, the larger (the) $\xi$ is, the more accurate results we can get. (In fact, our experimental results show the accuracy is pretty good even if we only set $\xi = 32$.) Note that, here $o.\odot$ is the circumscribed circle of $e$, which can assure that the distance from any point in $e$ to the center is always less than the distance threshold $\tau$. The main reasons we do this transformation are as follows: (i) it is convenient for the follow-up calculations since operating on line segments, in most cases, is more simple and efficient than on curves; (ii) it is easy to represent the calculated result; and (iii) all the troubles discussed in Section 3 can be significantly simplified. In the next section, we show how to represent different entities in a unified and efficient manner.

### 4.2   Label Based Data Structure (LBDS)
Once the pre-approximation idea is adopted, the boundaries of all the geometrical entities will be no curves. Moreover, we observe that $u$ may be a closed region with hole(s) or just be a simple closed region, and $s$ possibly consists of multiple subdivisions with hole(s). For ease of operating these entities in a unified and efficient manner, we need a targeted data structure to represent them. (We remark that the doubly connected edge list (DCEL) [1] consists of three collections of records: one for the vertices, one for the faces, and one for the half-edges; to our problem, it is a little clunky and not intuitive enough.) We next introduce some basic definitions in order to easily describe the details of our proposal.

**Definition 4 (Outer ring and inner ring).** *Given a closed region $c$ with a hole $h$, the boundary of $c$ and the one of $h$ are termed as the outer ring and inner ring of $c$, respectively.*

Note that, if a closed region contains $n$ holes, it clearly has $n$ inner rings, see e.g., Fig. 2c. Specifically, in order to easily handle those entities, we present a label based data structure, which consists of three domains—one *label* domain and two *pointer* domains.

- **Flag**: This domain is the boolean type. Specifically, 0 indicates the entity has no hole, and 1 indicates it has no less than one hole.
- **OPointer**: This domain points to a simple polygon that denotes the outer ring of the entity. A simple polygon consists of two domains.
  - **VPointer**: This domain points to a linked list that stores a series of vertexes.
  - **B**: This domain stores the MBR of the polygon.
- **IPointer**: This domain points to a linked list that stores the simple polygons, which denote the inner rings of this entity.

Hence $u$ can be represented by the LBDS directly, and $s$ can be represented by a linked list in which a series of 'LBDSs' are stored. This structure is intuitive, concise, and convenient for the follow-up computation, its benefits will be demonstrated gradually in the remainder of this paper.

## 4.3 Picking Out the Real Uncertainty Region

To compute the uncertainty region $u$ (by Formula 7) is straightforward, we can use the equilateral polygon $e$ to subtract each restricted area $r$ one by one. In Section 3, we show that computing $u$ is not a simple subtraction operation. In other words, Formulas 4 and 7 actually imply some possible mistakes, we slightly abuse them for presentation simplicity. In Fig. 3c, we say $d_2$ (rather than other three subdivisions) is the real uncertainty region, which is based on the lemma below.

**Lemma 1 (Choose real uncertainty region).** *Given $o.\odot$, $l_r$, $\tau$ and $\mathcal{R}$, we let $d$ be one of subdivisions after we execute the subtraction operation based on Equation (6). If $l_r \in d$, then $\sqrt{}(d)$, where $\sqrt{}(\cdot)$ denotes it is the real uncertainty region. Otherwise, $\neg(\sqrt{}(d))$.*

**Proof.** We first prove $l_r \notin d \Rightarrow \neg(\sqrt{}(d))$. According to Definition 1, we only need to show $o$ cannot be found in $d$. Clearly, $o$ must be located in $o.\odot$, since $l_r$ is the latest recorded location, and $\tau$ is the distance threshold. Furthermore, based on *analytic geometry*, it is easy to know that $o$ cannot reach $d$ if it does not walk out of $o.\odot$ (e.g., see Fig. 3b, $o$ cannot reach the topmost (or bottommost) region of $o.\odot$). Hence we cannot find $o$ in $d$.

The proof for the argument "$l_r \in d \Rightarrow \sqrt{}(d)$" can be obtained using the similar method above; omitted due to space limit. □

We remark that once the pre-approximation idea is used, to check whether or not $l_r \in d$ is simple, as it is just the *point in polygon problem* [1], [9]. After we obtain the real uncertainty region $u$, we can get $s$ by executing an *intersection operation* on $u$ and $R$. There are many algorithms (e.g., see [8], [12], [13], [18], [23]) that can perform intersection operation on polygons with holes. They however do not well consider the case of many holes. Even so, there is a simple method that is adapted from the algorithms mentioned above. Its general idea is to compute the intersection result between $R$ and the outer ring of $u$ at first, and then to use this intersection result to subtract each inner ring of $u$ one by one, finally it gets $s$.

## 4.4 The Appearance Probability

For uniform distribution PDF, the crucial task is to compute the areas of $u$ and $s$ (cf. Equation (5)). We show in Section 3 that computing these areas using a straightforward solution is overcomplicated. Now, we can easily compute them using the following method, which ascribes the pre-approximation and LBDS. Let $u_o$ be the outer ring of $u$, and $u_h^i$ be the $i$th hole in $u$. Let $s[i]$ be the $i$th subdivision of $s$, $s[i]_o$ be the outer ring of $s[i]$, and $s[i]_h^k$ be the $k$th hole of $s[i]$. First, given a polygon denoted by $P$, its area can be easily obtained based on the following equation [2].

$$\Lambda(P) = \frac{1}{2} \cdot \left( \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & x_3 \\ y_2 & y_3 \end{vmatrix} + \cdots + \begin{vmatrix} x_n & x_1 \\ y_n & y_1 \end{vmatrix} \right), \quad (9)$$

where $\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = (x_1 \cdot y_2 - x_2 \cdot y_1)$, and $(x_1, y_1)$ denote the coordinates of a vertex, other symbols have similar meanings. Furthermore, since we use the LBDS to represent $u$, and polygons are the basic elements of the LBDS, the area of $u$ can be obtained as follows:

$$\Lambda(u) = \Lambda(u_o) - \sum_{i=0}^{|\mathcal{H}|} \Lambda(u_h^i), \quad (10)$$

where $|\mathcal{H}| (\geq 0)$ is the number of holes in $u$. Similarly, since $s$ consists of a series of LBDSs, we have

$$\Lambda(s) = \sum_{i=1}^{|s|} \Lambda(s[i]) = \sum_{i=1}^{|s|} \left( \Lambda(s[i]_o) - \sum_{k=0}^{|s[i]_h|} \Lambda(s[i]_h^k) \right), \quad (11)$$

where $|s| (\geq 1)$ is the number of subdivisions of $s$, $|s[i]_h| (\geq 0)$ is the number of holes in $s[i]$. For arbitrary distribution PDF, we also use the Monte Carlo method to compute the probability $p$. We should note that the trouble shown in Section 3 does not exist now, as no curve is on the boundary of $u$ (or $s$), ascribing the pre-approximation idea.

## 4.5 Query Processing

A naive method is to do a linear scan—for each object $o$, it scans each restricted area $r$, and compute $u$ based on Formula 9, and then compute the probability $p$ if $u$ intersects with $R$. Clearly, it is inefficient to process the CSPRQ in such a way. We now present another natural but more efficient method as follows. Let $R_b$, $r_b$ and $o.\odot_b$ be the MBRs of $R$, $r$ and $o.\odot$, respectively.

**Definition 5 (Candidate moving object).** *Given the query range $R$ and a moving object $o$, $o$ is a candidate moving object such that $R_b \cap o.\odot_b \neq \emptyset$.*

**Definition 6 (Candidate restricted area).** *Given a restricted area $r$ and a moving object $o$, $r$ is a candidate restricted area such that $r_b \cap o.\odot_b \neq \emptyset$.*

Let $\mathcal{O}^*$ be the set of candidate moving objects, and $\mathcal{R}^*$ be the set of candidate restricted areas of the object $o$. We can rewrite Formula 9 as follows:

$$u \doteq e - \bigcup_{r \in \mathcal{R}^*} r. \tag{12}$$

The MBRs of the set $\mathcal{R}$ of restricted areas can be obtained easily, since each restricted area $r$ is static. Furthermore, since the recorded location $l_r$ and the distance threshold $\tau$ are already stored on the database server, the MBR of each moving object can be computed easily, it is a square centering at $l_r$ and with $2\tau \times 2\tau$ size (in fact it is just the MBR of $o.\odot$). Clearly, for all restricted areas and moving objects, we can use a twin-index to manage their MBRs. For instance, we can build two R-trees (or a variant such as the R*-tree) to manage the MBRs of moving objects and the ones of restricted areas, respectively. Let $\mathcal{I}_o$ and $\mathcal{I}_r$ be the index of moving objects and the one of restricted areas, respectively. Our query processing algorithm is illustrated below.

---

**Algorithm 1.** Constrained space probabilistic range query

---

(1)  Let $\Re = \emptyset$
(2)  Search $\mathcal{O}^*$ on $\mathcal{I}_o$ using $R_b$ as the input
(3)  **for** each $o \in \mathcal{O}^*$ **do**
(4)      Search $\mathcal{R}^*$ on $\mathcal{I}_r$ using $o.\odot_b$ as the input
(5)      Obtain $e$ based on Equation (8a) and (8b)
(6)      Compute $u$ based on Formula 12
(7)      **if** $u$ consists of multiple subdivisions **then**
(8)          Choose the real $u$ based on Lemma 1
(9)      Let $s = u \cap R$, and $p = 0$
(10)     **if** $s \neq \emptyset$ **then**
(11)         **if** uniform distribution PDF **then**
(12)             Compute $p$ based on Equation (5), (9), (10) and (11)
(13)         **else** // non-uniform distribution PDF
(14)             Compute $p$ based on Equation (6)
(15)         **if** $p \neq 0$ **then**
(16)             Let $\Re = \Re \bigcup (o, p)$
(17)  **return** $\Re$

---

*Cost analysis.* Let $C_o$ be the cost to search the set $\mathcal{O}^*$ of candidate moving objects. Clearly, we have

$$C_o \propto (|R_b|, |\mathcal{O}|), \tag{13}$$

where $\propto$ means "is proportional to", $|R_b|$ is the size of MBR of $R$, $|\mathcal{O}|$ is the cardinality of $\mathcal{O}$. Let $C_r$ be the cost to search the set $\mathcal{R}^*$ of candidate restricted areas. Similarly, we have

$$\begin{cases} C_r \propto (|o.\odot_b|, |\mathcal{R}|) & \text{(14a)} \\ |o.\odot_b| \propto \tau, & \text{(14b)} \end{cases}$$

where $|o.\odot_b|$ is the size of MBR of $o.\odot$, $|\mathcal{R}|$ is the cardinality of $\mathcal{R}$, $\tau$ is the distance threshold of $o$. Let $C_e$ be the cost to obtain the equilateral polygon $e$. We have

$$C_e \propto \xi, \tag{15}$$

where $\xi$ is the number of edges of $e$. Let $C_u$ be the cost to compute $u$ (lines 6-8), and $\zeta$ be the average number of edges of restricted areas. We have

$$\begin{cases} C_u \propto (\xi, \zeta, |\mathcal{R}^*|) & \text{(16a)} \\ |\mathcal{R}^*| \propto (\tau, |\mathcal{R}|), & \text{(16b)} \end{cases}$$

where $|\mathcal{R}^*|$ is the cardinality of $\mathcal{R}^*$. Let $C_s$ be the cost to compute $s$, and $\gamma$ be the number of edges of $u_o$ (the outer ring of $u$). Since the set $\mathcal{R}^*$ of candidate restricted areas form the holes of $u$, the number of edges of hole is also $\zeta$. We have

$$\begin{cases} C_s \propto (\gamma, |u_h|, \zeta) & \text{(17a)} \\ |u_h| \propto (\tau, |\mathcal{R}^*|). & \text{(17b)} \end{cases}$$

Let $C_p^u$ be the cost to compute $p$ in the case of uniform distribution, and $C_p^n$ be the cost to compute $p$ in the case of non-uniform distribution. Note that each cost (in the for loop) refers to the average cost, and we overlook the cost of adding a tuple $(o, p)$ into $\Re$ as it is trivial. We also note that $C_o$ is related to $\mathcal{I}_o$ (e.g., the fan-out of $\mathcal{I}_o$), and $C_r$ is related to $\mathcal{I}_r$ (e.g., the fan-out of $\mathcal{I}_r$). We assume existing indexing technique is to be adopted, we hence omit this discussion in our analysis for simplicity. Let $C_t$ denote the total cost, we have

$$C_t = \begin{cases} C_o + |\mathcal{O}^*|(C_r + C_e + C_u + C_s + C_p^u) & \text{(18a)} \\ C_o + |\mathcal{O}^*|(C_r + C_e + C_u + C_s + C_p^n). & \text{(18b)} \end{cases}$$

Clearly, to reduce the total cost $C_t$, we should reduce at least one sub-cost. Since $|\mathcal{O}|$ and $|\mathcal{R}|$ are depended on the application scenario, and $|R_b|$ is depended on the input of the user, we can easily know by Formulas 13, 14a and 14b that there is little space to reduce $C_o$ and $C_r$. Furthermore, there is also (almost) no space to reduce $C_e$, as $\xi$ is used to assure the accuracy of our algorithm, and the natural solution to execute Equations (8a) and (8b) is already pretty efficient. We also note that our solution to compute the area of $u$ (or $s$) is already pretty simple and efficient, implying that there is also (almost) no space to reduce $C_p^u$. Regarding to $C_p^n$, it is mainly depended on the number $N_1$ of random generated points (cf. Equation (6)), and $N_1$ is used to assure the accuracy of our algorithm. Naturally, we need to set $N_1$ to an acceptable value at least which can assure an allowable workload error. This implies that there is also no much space to reduce $C_p^n$. Recall Section 4.3, we compute $u$ and $s$ using the simple methods, which are somewhat inefficient, and for each single query, the cost to compute $u$ and $s$ is $|\mathcal{O}^*|(C_u + C_s)$, which is non-trivial compared to $C_t$. (Note that in the previous works, $C_u = 0$, $C_s$ is pretty small and almost can be overlooked, as $u$ is a circle in the case of no restricted areas.) These facts motivate us to further optimize our solution by reducing $C_u$ and $C_s$. In the next section, we show how to reduce $C_u$ and $C_s$ based on two insights and two simple but usually easy to ignore facts.

## 5 FURTHER OPTIMIZE OUR SOLUTION

The optimizations are mainly based on two insights: (i) the number of *effective subdivision*s is no more than 1; and (ii) an entity with the larger *span* is more likely to subdivide a single region. In addition to the main insights above, we also realize two other (simple but usually easy to ignore) facts and utilize them; specifically, two mechanisms are developed: *postpone processing* and *lazy update*.
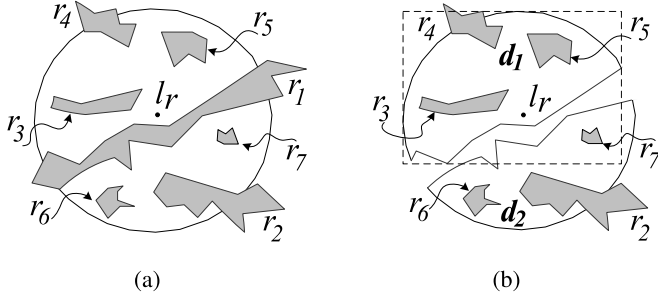
Fig. 4. Illustration of subdivisions pruning and span. The grey polygons illustrate the set $\mathcal{R}^*$ of candidate restricted areas, and the big equilateral polygon illustrates $e$ with 32 edges.

## 5.1 Effective Subdivision

**Definition 7 (Effective subdivision).** *Given $o.\odot$ and a set $\mathcal{R}^*$ of candidate restricted areas, without loss of generality, assume that $|o.\odot - r| > 1$ when the $i$th "subtraction operation" is executed, where $|\cdot|$ denotes the number of subdivisions of the subtraction result, and $1 \le i \le |\mathcal{R}^*|$. A subdivision $d$ is an effective subdivision such that the recorded location $l_r \in d$.*

**Lemma 2 (Number of effective subdivisions).** *Assume that $|o.\odot - r| > 1$, the number of effect subdivisions is no more than 1.*

**Proof.** It follows from Definition 7 and *analyst geometric*, the details are omitted due to space limit. □

Let $d^e$ be the effective subdivision when the $i$th "subtraction operation" is executed, where $1 \le i \le |\mathcal{R}^*|$. We have

**Lemma 3 (Subdivisions pruning).** *Assume that $|o.\odot - r| > 1$, all subdivisions except $d^e$ can be pruned safely.*

**Proof.** By Lemma 2, we only need to show the uncertainty region $u \subseteq d^e$. This can be proved based on Lemma 1 and *analyst geometric*; omitted due to space limit. □

Let $d_b^e$ be the MBR of the effective subdivision $d^e$. Lemmas 2 and 3 indicate that we can immediately discard unrelated subdivisions once multiple subdivisions appear. In particular, we can use $d_b^e$ to prune the rest of candidate restricted areas, as it has a stronger pruning power compared to $o.\odot_b$. We remark that the entity $o.\odot$ is continuous evolving when it subtracts candidate restricted areas one by one, and in fact we use $e$ (rather than $o.\odot$) to do "subtraction operation", as we adopt the *pre-approximation* strategy. We abuse the notation $o.\odot$ in Definition 7 and Lemmas 2 and 3.

*Comparison.* The approach above is superior to the approach in Section 4.3 (called the *prior approach*) in the following points (note that the prior approach chooses the real uncertainty region at the last step):

1. The prior approach needs to use *each* subdivision to subtract the rest of candidate restricted areas. In contrast, the approach above only needs to use $d^e$ to subtract the rest of candidate restricted areas. For instance, in Fig. 4b, the prior approach uses not only $d_1$ but also $d_2$ to subtract the rest of candidate restricted areas $(r_2, \ldots, r_7)$, whereas the approach above only needs to use $d_1$ to subtract the rest of candidate restricted areas.

2. The prior approach cannot prune the rest of candidate restricted areas. In contrast, the approach above can prune the unrelated candidate restricted areas. For example, in Fig. 4b, the prior approach cannot prune candidate restricted areas as they are related to either $d_1$ or $d_2$, whereas the approach above can use the MBR of $d_1$ to prune $r_2$ and $r_6$, and use $d_1$ to prune $r_7$.

We show the superiority of the approach above. The natural method to compute $u$ however, is using $e$ to *randomly* subtract each $r$ $(\in \mathcal{R}^*)$ one by one (cf. Section 4.3), implying that $r_1$ in Fig. 4 may be handled at last. In this case, the superiority of the approach above disappears. In the next section, we show how to maximize its superiority by utilizing the span.

## 5.2 Span

Let $g$ be a 2D entity, and $g_b$ be the MBR of $g$. Let $(g_b.x^-, g_b.y^-)$ and $(g_b.x^+, g_b.y^+)$ be the left-bottom point and right-top point of $g_b$, respectively. We denote by $g_s$ the span of $g$, which is defined as follows.

**Definition 8 (Span).** *Given a 2D entity $g$, its span $g_s$ is computed as*

$$g_s = \begin{cases} g_b.x^+ - g_b.x^-, & \text{if } g_b.x^+ - g_b.x^- \ge g_b.y^+ - g_b.y^- \\ g_b.y^+ - g_b.y^-, & \text{otherwise.} \end{cases}$$

**Heuristic 1.** *A 2D entity with the larger span usually is more likely to subdivide a single (closed) region.*

See Fig. 4a for example. Clearly, here $e$ can be regarded as a single (closed) region, and each $r \in \mathcal{R}^*$ can be regarded as a 2D entity. Compared to other candidate restricted areas, here $r_1$ has the largest span and it is more likely to subdivide $e$ into multiple subdivisions. Heuristic 1 motivates us to handle $r$ that has the larger span as early as possible. This can be achieved by sorting their spans according to the *descending* order. We remark that the span is a real number, hence the overhead to sort $|\mathcal{R}^*|$ candidate restricted areas is pretty small, and (almost) can be overlooked compared to the overhead to execute $O(|\mathcal{R}^*|)$ times geometrical subtraction operations.

*Another application.* To compute $s$, the method in Section 4.3 (called the *prior method*) uses the intersection result, denoted by $\varphi$, between $u_o$ and $R$ to subtract each hole $u_h$. (Here $u_o$ refers to the outer ring of $u$.) We now show how to use the span of hole to improve the prior method. Let $\mathcal{H}$ be the set of holes of $u$.

**Lemma 4 (Subdivisions retaining).** *Given $\varphi$ and $u_h$, if $|\varphi - u_h| > 1$, any subdivision of this subtraction result cannot be discarded, where $|\cdot|$ denotes the number of subdivisions.*

**Proof.** By contradiction, assume the subdivision $d$ $(\in \varphi - u_h)$ can be discarded. This implies any point $p' \in d$ can be discarded, i.e., $o$ cannot reach $p'$ (from $l_r$) if it does not walk out of $o.\odot$. (Note that $d$ possibly contains or intersects with other holes, but the case $d$ itself being a hole is impossible. Otherwise, $d$ and $u_h$ form a larger hole as they are connected.) However, by the definition of $u$, $o$ can reach any point $p' \in u_o - \bigcup_{u_h \in \mathcal{H}} u_h$ without the need of walking out of $o.\odot$, it is contrary to the conclusion above. This completes the proof. □
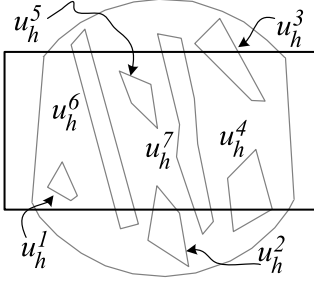
Fig. 5. Illustration of another application. The biggest rectangle denotes the query range $R$. The uncertainty region $u$ is a closed region with seven holes.

The lemma above indicates that once $|\varphi - u_h| > 1$, we need to use each subdivision to subtract the rest of holes. Without loss of generality, assume that it produces $k$ subdivisions after we handle $|\mathcal{H}| - i$ holes, where $i \leq |\mathcal{H}|$. For ease of discussion, we assume each hole $u_h$ at most can subdivide $\varphi$ into two subdivisions, and all the previous $|\mathcal{H}| - i$ holes can subdivide $\varphi$, implying that $k = |\mathcal{H}| - i + 1$. We can easily know that handling the previous $|\mathcal{H}| - i$ holes needs $1 + 2 + \cdots + (k-1) = \frac{(k-1)(k-2)}{2}$ times subtraction operations, since a new subdivision is to be produced when a hole is handled. For the rest of holes, assume that each of them cannot subdivide $\varphi$, handling them needs $k \times i$ times subtraction operations. Let $x_1$ be the total number of the subtraction operations when handling all the $|\mathcal{H}|$ holes, we have $x_1 = \frac{(k-1)(k-2)}{2} + ki$. In contrast, if we swap the order to process the $|\mathcal{H}|$ holes. That is, we first handle $i$ holes that cannot subdivide $\varphi$ and then handle those $|\mathcal{H}| - i$ holes that can subdivide $\varphi$. Similarly, let $x_2$ be the total subtraction operation times. We have $x_2 = i + \frac{(k-1)(k-2)}{2}$. Since $k = |\mathcal{H}| - i + 1 - x_1 - x_2 = |\mathcal{H}|i - i^2$. Hence, we have

$$\operatorname*{argmax}_{i}(x_1 - x_2) = \frac{|\mathcal{H}|^2}{4}. \tag{20}$$

The formula above and Lemma 4 motivate us to handle holes that cannot subdivide $\varphi$ as early as possible. This can be achieved by sorting their spans according to the *ascending* order. For example, in Fig. 5 we handle $u_h^6$ and $u_h^7$ at last. We remark that although some subtraction operations may be *empty* operations when two entities are disjoint, it still incurs extra comparison overhead. In the sequel, we show two additional observations, yielding two (small) mechanisms.

*Additional observations.* To compute $u$, the method in Section 4.3 is using $e$ to subtract each candidate restricted area $r$ on by one. Consider the case $r \subset e$. Clearly, $e - r$ forms a polygon with hole. For ease of discussion, let $e'$ be the subtraction result between $e$ and $r$, and assume that the next candidate restricted area to be processed is $r'$. The natural approach is using $e'$ to subtract $r'$. This approach however, complicates the follow-up computation, and thus incurs extra overhead. This is mainly because geometrical operation on polygons with holes is generally more complicated and time consuming than on polygons without holes. To overcome this drawback, we employ a *postpone processing* mechanism. Specifically, if

$r \subset e$, we postpone the subtraction operation by caching $r$ in a temporary place; after all other candidate restricted areas are handled, we finally fetch $r$ from the temporary place and then handle it. For instance, in Fig. 4b we handle $r_3$ and $r_7$ at last.

Another common case is that $r$ intersects with $e$ but $|e - r| = 1$, where $|\cdot|$ denotes the number of subdivisions. To this case, the natural method is using $e$ to subtract $r$, and then update the MBR of this subtraction result. This approach is inefficient, due to two main reasons: (i) such a new MBR usually does not make enough contribution to the rest of computation, i.e., its pruning power is weak in most cases; (ii) to obtain such a new MBR also needs to traverse the vertexes of this subtraction result, which incurs the extra overhead. To overcome this drawback, we employ a *lazy update* mechanism. Specifically, if $|e - r| = 1$ (i.e., no multiple subdivisions appear), we only execute the subtraction operation but do not update the MBR of the subtraction result. $r_4$ in Fig. 4b illustrates this case, for example. We remark that the two mechanisms above can be directly applied to the case of computing $s$. For instance, see Fig. 5, the *lazy update* can be applied to $u_h^3$, and the *postpone processing* can be applied to $u_h^1$.

## 6 PRECOMPUTATION BASED METHOD

In the previous discussion, we assume a twin-index is adopted: $\mathcal{I}_r$ is used to manage restricted areas, and $\mathcal{I}_o$ is used to manage moving objects. Once a moving object $o$ reports its new location to the server, we update its recorded location $l_r$, and also update $\mathcal{I}_o$ (see Section 4.5). An obvious characteristic of this method is to compute uncertainty regions on the fly, and an easily brought to mind method is to incorporate the precomputation strategy.

Simply speaking, we index restricted areas at first, and then precompute uncertainty regions and index them. Here we also adopt a twin-index. One is used to manage restricted areas, which is the same as the previous. Another, called $\mathcal{I}_u$, is used to manage the uncertainty regions. Specifically, for each moving object $o$, we search its candidate restricted areas on $\mathcal{I}_r$, and then compute its uncertainty region $u$ and index it using $\mathcal{I}_u$.

Note that, it is possible that an object $o$ reports its new location to the server in the process of constructing $\mathcal{I}_u$. To this issue, we differentiate two cases: (i) the uncertainty region of this object $o$ has ever been precomputed and indexed; and (ii) the uncertainty region of this object $o$ has not been precomputed. Both of the cases can be tackled easily. For instance, regarding to the first case, we can update its recorded location $l_r$ in the database, and then recompute its uncertainty region and update the current $\mathcal{I}_u$ right now. For the second case, we only need to update its recorded location $l_r$ in the database for the present. Once the precomputation is accomplished, the query can be executed, which is the similar as the previous. Henceforth, if an object $o$ reports its new location to the server, we also compute its uncertainty region (offline) and update $\mathcal{I}_u$. Note that although the precomputation based solution seems to be more efficient, it however has a (non-trivial) drawback, i.e., its preprocessing time is rather large, which will be demonstrated in the next section.

# 7 PERFORMANCE STUDY

## 7.1 Experiment Settings

*Datasets.* In our experiments, both real and synthetic datasets are used. Two real datasets are named as CA and LB,[2] respectively. The CA contains 104,770 2D points, and the LB contains 53,145 MBRs. We use the CA to denote recorded locations of moving objects, and the LB to denote restricted areas. In order to simulate moving objects with different characteristics, we randomly generate different distance thresholds (from 20 to 50) for them. The size of 2D space is fixed at 10,000 × 10,000, all datasets are normalized in order to fit this size of 2D space. Synthetic datasets also include two types of information. We generate a number of polygons to denote restricted areas, and let them uniformly distributed in 2D space. We generate a number of points to denote recorded locations of moving objects, and let them randomly distributed in 2D space. Note that, there is an extra constraint — these points cannot be located in any restricted area.[3] We use the RE and SY to denote the real and synthetic datasets, respectively.

*Methods.* Existing methods are invalid to our problem, we thus do not compare with them[4]. The straightforward method is infeasible and difficult to implement, as analysed in Section 3, we thus do not discuss this method in our experiments (we believe the readers can understand this situation)[5]. Specifically, we implemented our solution (Section 4), our solution together with the optimization (Section 5), and the precomputation based solution together with the optimization (Section 6). We use the same indexing structure, R-tree, for the three algorithms above. For brevity, we use S, SO, and PSO to denote them, respectively. Furthermore, by the convention, we implemented a baseline method that is to do a linear scan when searching candidate moving objects and candidate restricted areas (note: other strategies are the same as the ones of the S). We use B to denote it for short.

*Distributions.* In our experiments, two types of PDFs are used: uniform distribution and distorted Gaussian (note: our solutions can also work for other distribution PDFs, since we adopt the Monte Carlo method that can work for arbitrary distribution PDF). The definition of distorted Gaussian is based on the general Gaussian.[6] Let $pdf_G(x, y)$

and $pdf_{DG}(x, y)$ be the PDFs of general Gaussian and distorted Gaussian, respectively, and let $\lambda$ be a coefficient, where $\lambda = \int_{\forall (x,y) \in u} pdf_G(x, y) dx dy$, we have

$$pdf_{DG}(x, y) = \begin{cases} \frac{pdf_G(x,y)}{\lambda}, & \text{if } (x, y) \in u \quad (21a) \\ 0, & \text{otherwise.} \quad (21b) \end{cases}$$

In theory, we should have calculated $\lambda$ and converted $pdf_G(x, y)$ into $pdf_{DG}(x, y)$ for each object $o$. Fortunately, we need to neither calculate $\lambda$, nor do any conversion. This is because $\lambda$ will be eliminated when we substitute $pdf_{DG}(x, y)$ with $\frac{pdf_G(x,y)}{\lambda}$ in the following formula:

$$p = \frac{\sum_{i=1}^{N_2} pdf_{DG}(x_i, y_i)}{\sum_{i=1}^{N_1} pdf_{DG}(x_i, y_i)}, \quad (22)$$

where $N_1$, $N_2$ are the number of random points being located in $u$ and $s$, respectively. For brevity, we use UD and DG to denote uniform distribution and distorted Gaussian, respectively.

*Metrics.* The performance metrics in our experiments include: the I/O time, query time (the sum of I/O and CPU time), preprocessing time and accuracy. We use the workload error to measure the accuracy. Two types of common workload errors are the relative workload error (RWE) and absolute workload error (AWE).[7] In order to investigate I/O and query time, we randomly generate 50 query ranges, and run 10 times for each test, and finally compute the average I/O and query time for estimating a single query. We run 10 times and compute the average value for estimating the preprocessing time. In order to get the workload error, we generate an object $o$ at the centre of the 2D space, and assign a value to the distance threshold $\tau$, and then compute its uncertainty region $u$. Next, we generate 100 query ranges that have the same size, but have different intersections with $u$. At first run, we get the real answer of each query by setting $N' = 1e + 7$. (We remark that an *absolute* real answer is unavailable, since the Monte Carlo method itself is an approximation algorithm. Even though, this obtained answer can be almost regarded as the real value, as we assign a very large number to $N'$. In the remainder of this paper, we slightly abuse the term real value.) Next, we vary the size of $N'$ to get several groups of workload errors. We note that another parameter $\xi$ also results in workload errors. When we study the impact of $\xi$ on the accuracy, we also use the workload error to estimate the returning results. The test method is similar to the one used to test $N'$. Specifically, we get the real answer of each query by setting $\xi = 32^2$, then vary $\xi$ to get several groups of workload errors.

*Parameters.* All codes used in our experiments are written in C++ language; all experiments are conducted on a computer with 2.16 GHz dual core CPU and 1.86 GB of memory, running Windows XP. The page size is fixed to 4,096 bytes; the maximum number of children nodes in the R-tree is fixed to 50. The standard deviation of $pdf_G(x, y)$ (used for defining $pdf_{DG}(x, y)$) is set to $\frac{\tau}{5}$, the mean $u_x$ and $u_y$ are set to $l_r.x$ and $l_r.y$, respectively, where $(l_r.x, l_r.y)$ denote the coordinates of $l_r$.

---

2. The CA dataset is available in site: http://www.cs.utah.edu/ life-ifei/SpatialDataset.htm, and the LB dataset is available in site: http://www.rtreeportal.org/.

3. In fact, once this constraint is employed, the number of effective 2D points in the CA is 101,871. Furthermore, since some MBRs in the LB are line segments, or they are not disjoint, the number of effective rectangles in the LB is 12,765 after we remove those unqualified MBRs.

4. Imagine if we directly use existing methods (e.g., [4]), which renders the following unfair comparison: (i) the query answer is clearly incorrect, as analysed in Section 1; and (ii) the query time is clearly less than our algorithm's, as existing methods do not need to handle restricted areas.

5. From another perspective, the problem studied is different from most problems for which a straightforward, easy to implement and exact method can always be found.

6. The general Gaussian has an infinite input space that is symmetric, its PDF is $\frac{1}{2\pi\delta^2} e^{\frac{(x-u_x)+(y-u_y)}{-2\delta^2}}$. The input space of distorted Gaussian, however, is limited to the uncertainty region $u$ and it may be not symmetric.
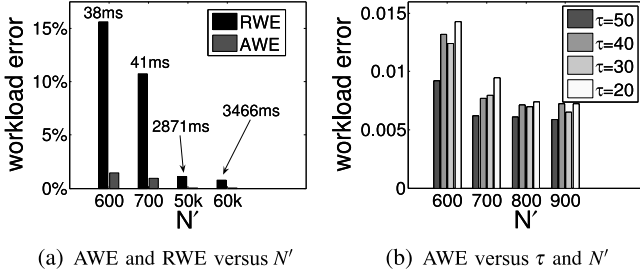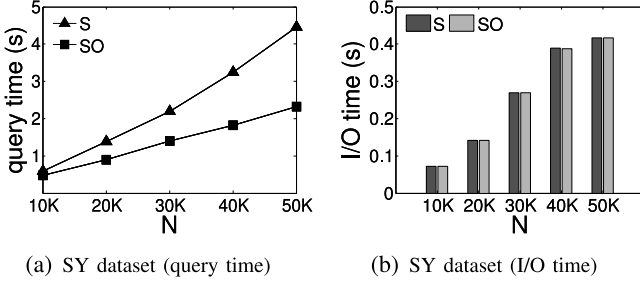
7. $RWE = \frac{|estimated\ value - real\ value|}{real\ value}$, $AWE = |estimated\ value - real\ value|$.

(a) AWE and RWE versus $N'$

(b) AWE versus $\tau$ and $N'$

Fig. 6. Workload Error Comparison.



(a) SY dataset (query time)

(b) SY dataset (I/O time)

Fig. 7. Query and I/O performance versus $N$.

The settings of other parameters are illustrated in Table 2, in which the numbers in **bold** represent the default settings. $N$, $M$ and $\zeta$ refer to the settings of synthetic datasets, the default setting of each restricted area $r$ is a rectangle with $40 \times 10$ size, and the one of the query range $R$ is a rectangle with $500 \times 500$ size.

## 7.2 Results

### 7.2.1 Choose the Error Metric and the Size of $N'$

Recall Section 7.1 that there are two error metrics, both of them can be used to measure the accuracy. Note that, to ensure a small RWE takes more time (a non-trivial number) than to ensure a same value of AWE. The results shown in Fig. 6a confirm this fact. These results are derived by setting $\tau = 20$. In this figure, the AWE is 0.95% (i.e., 0.0095) and the RWE is 10.75%, when $N' = 700$. It is unreasonable if we choose 10.75% as the RWE. Otherwise, it implies that returning a value of 89.25% will be tolerated even if the real value is 100%. Therefore, we need to increase $N'$ in order to get a smaller RWE. By doing so, we get RWE= 1.12% and AWE= 0.05% at $N' = 50,000$. Therefore, if we want to assure a value 1% of RWE, we have to set $N' > 50,000$ at least. However, even if we let $N' = 50,000$, and only compute a single object's probability, it takes about 2,871 milliseconds, implying that to assure a small RWE (e.g., 1%) takes much time. To further verify

TABLE 2
Parameters Used in Our Experiments

| Para. | Description | Value |
|---|---|---|
| $\xi$ | number of edges of $e$ | $[16, 24, \mathbf{32}, 48, 64, 32^2]$ |
| $N$ | cardinality of $\mathcal{O}$ | $[10k, 20k, 30k, 40k, \mathbf{50k}]$ |
| $M$ | cardinality of $\mathcal{R}$ | $[10k, 20k, 30k, 40k, \mathbf{50k}]$ |
| $\theta$ | size of $R$ | $[100, 200, 300, 400, \mathbf{500}]$ |
| $\zeta$ | number of edges of $r$ | $[\mathbf{4}, 8, 16, 32, 64]$ |
| $N'$ | number of random points | $[600, \mathbf{700}, 800, 900, 5k, 6k, 10^7]$ |
| PDF | distribution in $u$ | $[\mathbf{UD}, DG]$ |
| $\tau$ | distance threshold of $o$ | $[20, 21, \ldots, 49, 50]$ |

this fact, we conduct another set of experiments using both real and synthetic datasets where we set $N' = 50,000$ and others are the default settings. The results are listed in the table below.
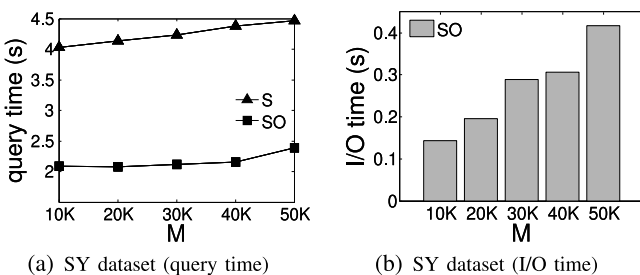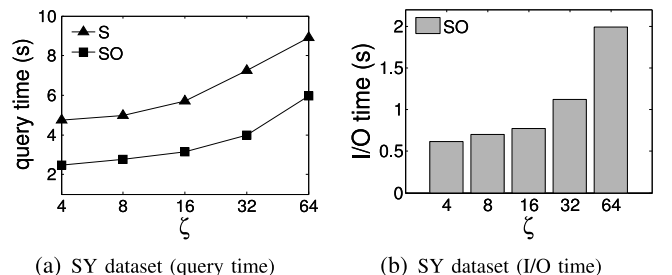
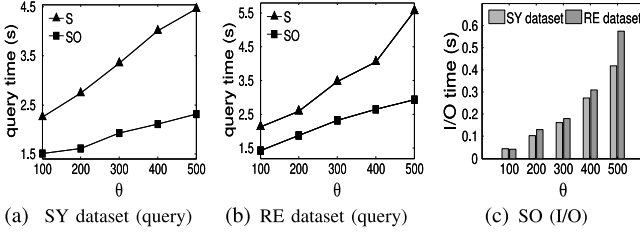| Dataset | Total test time (sec.) | Query time (sec.) |
|---|---|---|
| RE | 21399.25 | 427.985 |
| SY | 166022.6 | 332.004 |

The total test time takes about 59 and 46 *hours*, respectively; and each single query takes several *minutes*. In view of these, and in most cases a small AWE is enough to satisfy our demand, in the rest of experiments we choose to assure a small AWE by setting $N'$ to a smaller value. Fig. 6b depicts the results by setting $\tau$ to 20, 30, 40, and 50, respectively. We can see that on the whole, an object with a smaller $\tau$ usually needs a larger $N'$, if we want to assure the same value of AWE. Based on the facts above, unless stated otherwise, we choose $N' = 700$ in the rest of experiments, which can ensure a value 0.01 of AWE.

### 7.2.2 S versus SO

We first study the impact of $N$, $M$ and $\zeta$ based on synthetic datasets, and then study the impact of $\theta$ and $\xi$ based on both real and synthetic datasets.

*Impact of $N$.* Fig. 7 illustrates the results by varying $N$ from 1e+4 to 5e+4. We can see that with the increase of $N$, both the query and I/O time increase for the two methods. In terms of query performance, we find that the SO outperforms the S, which demonstrates the efficiency of our optimization. In particular, their performance differences are more obvious especially when $N$ is large, which demonstrates the scalability of SO is better than the one of S. We remark that the I/O performance of the two methods is identical, this is because same location and restricted area records are fetched from the database for the two methods.



(a) SY dataset (query time)

(b) SY dataset (I/O time)

Fig. 8. Query and I/O performancee versus $M$.



(a) SY dataset (query time)

(b) SY dataset (I/O time)

Fig. 9 Query and I/O performance versus $\zeta$.

Fig. 10. Query and I/O performance versus $\theta$.



Fig. 11. Query and I/O performance versus $\xi$.

In the rest of experiments, we only report the IO performance of SO.

*Impact of M.* Fig. 8 depicts the results by varying $M$ from 1e+4 to 5e+4. We can see that with the increase of $M$, the I/O time increases. This is because with the same size of $R$, we have to fetch more restricted area records from the database. In terms of query performance, the SO always outperforms the S, which further demonstrates the efficiency of our optimization.

*Impact of $\zeta$.* Fig. 9 illustrates the results by varying $\zeta$ from 4 to 64. Specifically, we let equilateral polygons denote restricted areas. For each restricted area $r$, it has the property that the distance from its center to its vertice is 20. For different test groups, we only vary $\zeta$ of each restricted area $r$. As we expected, with the increase of $\zeta$, both the query and I/O time increase for the two methods. The reasons are as follows. The distance from the center to the vertice is a fixed value, an equilateral restricted area $r$ with more edges will occupy more areas. With the increase of $\zeta$, some restricted areas that do not intersect with $e$ (when $\zeta$ is a small value) will possibly intersect with $e$, where $e$ is the equilateral polygon approximated from $o.\odot$. We thus have to compute the subtraction between $e$ and these restricted areas, which incurs more CPU time. Furthermore, with the increase of $\zeta$, the amount of information in each $r$ also increases. Thus, it takes more time to fetch restricted area records from the database, which incurs more I/O cost. In terms of query performance, we can also see that the SO always
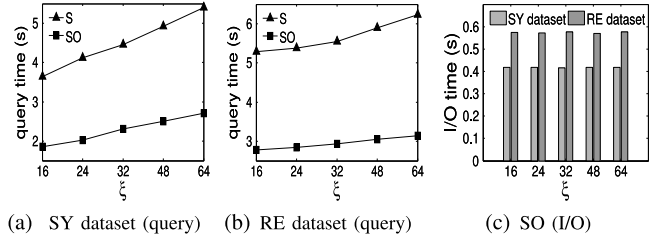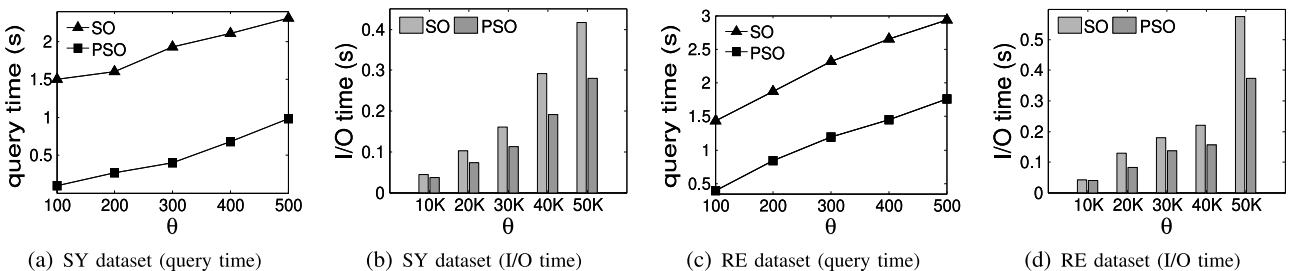
outperforms the S, which once again illustrates the efficiency of our optimization.

*Impact of $\theta$.* Fig. 10 depicts the performance results by varying $\theta$ from $100 \times 100$ to $500 \times 500$. We can see that with the increase of $\theta$, both the query and I/O time increase for the two methods. This is because more objects are to be located in $R$, more records thus should be fetched from the database, which incurs more I/O time. On the other hand, for those increased objects, we also have to compute their probabilities, which incurs more CPU time. In particular, we can see that the SO always outperforms the S in terms of the query performance. This further demonstrates the efficiency of our optimization.

*Impact of $\xi$.* Fig. 11 illustrates the results by varying $\xi$ from 16 to 64. We can see that with the increase of $\xi$, the I/O time is almost constant. This is mainly because the number of records fetched form the database are almost same for two queries with different $\xi$. On the other hand, we can see that in terms of query time, the growth rate of the S is greater than the one of the SO, which demonstrates that the SO is more stable than the S. Furthermore, the SO always outperforms the S, which further demonstrates the efficiency of our optimization.

### 7.2.3 B versus S

We can see from Fig. 12 that both the query and I/O performance of the S significantly outperform the ones of B. Note that when we vary other parameters in addition to $\theta$, we also get similar results, i.e., the S significantly outperforms
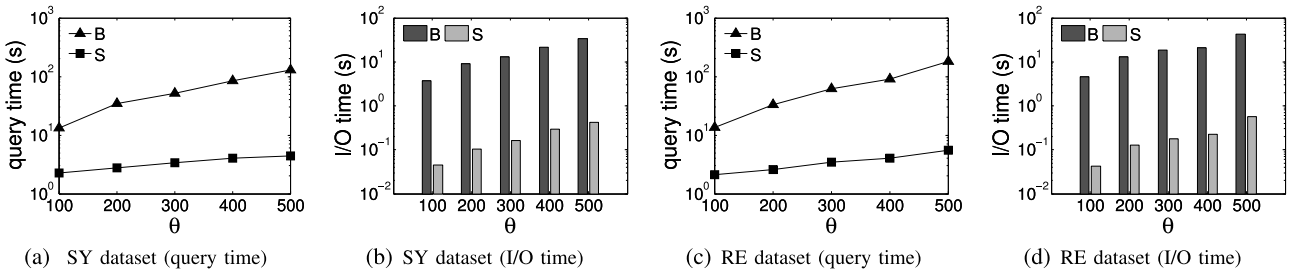


Fig. 12. Query and I/O performance versus $\theta$.



Fig. 13. Query and I/O performance versus $\theta$.

(a) SY dataset     (b) SY dataset     (c) SY dataset
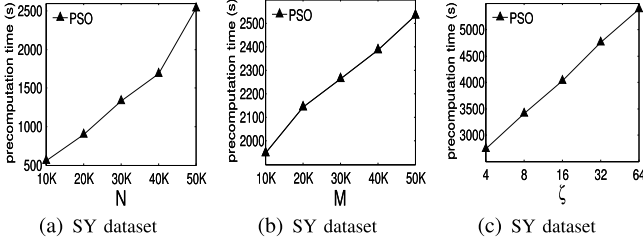
Fig. 14. Precomputation time versus $N$, $M$ and $\zeta$.



(a) SY dataset (query time)     (b) RE dataset (query time)

Fig. 15. Query efficiency versus PDF.

the B. Clearly, the SO also significantly outperforms the B as it is superior than the S.

### 7.2.4  SO versus PSO

From Fig. 13, we can see that the PSO outperforms the SO regardless of query or I/O performance, which demonstrates the benefits of precomputing uncertainty regions. Note that we also vary other parameters and find that the PSO also outperforms the SO in terms of query and I/O performance.

However, we find that the time for precomputing uncertainty regions is rather long, the results are plotted in Fig. 14. The PSO takes 2532.828 seconds (about 42 minutes) when the default settings of the synthetic datasets are used (note: the SO do not need to precompute uncertainty regions, and only need to index restricted areas and moving objects, which can be finished in several seconds). In addition, when we set $\zeta$ to 64, the PSO takes 5386.812 seconds (about an hour and a half). The long preprocessing time can be regarded as a (non-trivial) drawback of this approach.

### 7.2.5  Compare Different PDFs

We next test the impact of PDFs. Specifically, we let all parameters be totally same except the PDF (note: here we use our preferred method, i.e., the SO). On one hand, we compare their query time by varying $\theta$ (here $N'$ is the default setting). On the other hand, we compare their accuracies by varying $\xi$ (here we set $N'$ to $10^7$).

*Query time.* Figs. 15a and 15b depict the results when we vary $\theta$. We can see that the query time when the PDF is DG is more than the one when the PDF is UD. This is mainly because the time computing a single object's probability is relatively long when the PDF is DG.

*Accuracy.* In addition, by varying $\xi$ from 16 to 64, their accuracies are plotted in Fig. 16a. As we expected, the larger (the) $\xi$ is, the more accurate answer we can get. In particular, we can see that, compared to the case of uniform distribution, $\xi$ makes less impact on the accuracy when the PDF is
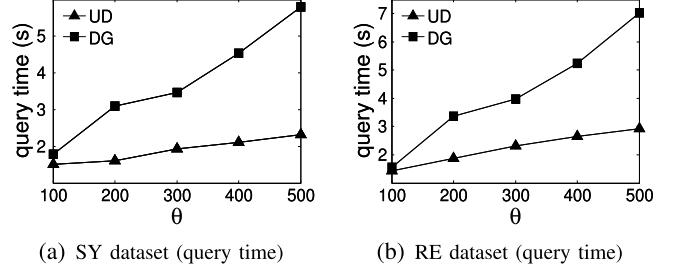
DG. Hearteningly, even if the PDF is UD, the accuracy of the proposed method is still high since the AWE is about $634 \times 10^{-6}$ when $\xi = 32$. Moreover, we can see from Fig. 16b that, with the same $\xi$, the smaller the distance threshold $\tau$ is, the more accurate answer we can get when the PDF is UD. Interestingly, the case of DG is exactly the opposite, which confirms (in a different way) the previous conclusion derived from Fig. 6b. Furthermore, we also report the RWEs of the set of experiments, which are shown in Fig. 16c and 16d, respectively. By comparing Figs. 16a/16b and Figs. 16c/16d, we can easily see that in the same settings, the AWE is significant smaller than the RWE, which is in line with the conclusion derived from Fig. 6a.

## 8  CONCLUSION

This paper studies the CSPRQ for uncertain moving objects. The deliberate analyses offer insights into the problem considered, and show that to process the CSPRQ using a straightforward method is infeasible. We propose the targeted solution and demonstrate its efficiency and effectiveness through extensive experiments. An additional finding is the precomputation based method has a non-trivial preprocessing time (although it outperforms our preferred solution in other aspects), which offers an important indication sign for the future research. We conclude this paper with several interesting research topics: (i) how to process the CSPRQ in 3D space? (ii) if the location update policy is the time based update, rendering that the uncertainty region $u$ is to be a continuously changing geometry over time, how to process the CSPRQ in such a scenario? (iii) if the query issuer is also moving, the location of query issuer is also uncertain, how to process the location based CSPRQ?

(a) AWE versus $\xi$ ($\tau = 50$)     (b) AWE versus $\tau$ and $\xi$     (c) RWE versus $\zeta$ ($\tau = 50$)     (d) RWE versus $\tau$ and $\xi$
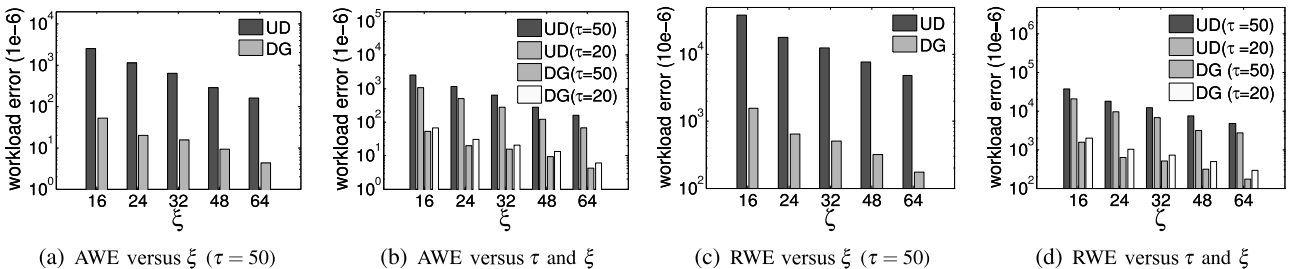
Fig. 16. Accuracy versus PDF.

# REFERENCES

[1] M. D. Berg, O. Cheong, M. V. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd Ed. Berlin, Germany: Springer, 2008.

[2] B. Braden, "The surveyor's area formula," *College Math. J.*, vol. 17, no. 4, pp. 326–337, 1986.

[3] J. Chen and R. Cheng, "Efficient evaluation of imprecise location dependent queries," in *Proc. IEEE Int. Conf. Data Eng.*, 2007, pp. 586–595.

[4] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, "Querying imprecise data in moving object environments," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1112–1127, 2004.

[5] B.S. E. Chung, W.-C. Lee, and A. L. P. Chen, "Processing probabilistic spatio-temporal range queries over moving objects with uncertainty," in *Proc. Int. Conf. Extending Database Technol.*, 2009, pp. 60–71.

[6] T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle, "Querying uncertain spatio-temporal data," in *Proc. IEEE Int. Conf. Data Eng.*, 2012, pp. 354–365.

[7] B. Gedik, K.-L. Wu, P. S. Yu, and L. Liu, "Processing moving queries over moving objects using motion-adaptive indexes," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 5, pp. 651–668, 2006.

[8] G. Greiner and K. Hormann, "Efficient clipping of arbitrary polygons," *ACM Trans. Graph.*, vol. 17, no. 2, pp. 71–83, 1998.

[9] K. Hormann and A. Agathos, "The point in polygon problem for arbitrary polygons," *Comput. Geometry*, vol. 20, no. 3, pp. 131–144, 2001.

[10] H. Hu, J. Xu, and D. L. Lee, "A generic framework for monitoring continuous spatial queries over moving objects," *ACM Int. Conf. Manag. Data*, 2005, pp. 479–490.

[11] B. Kuijpers and W. Othman, "Trajectory databases: Data models, uncertainty and complete query languages," *J. Comput. Syst. Sci.*, vol. 76, no. 7, pp. 538–560, 2010.

[12] Y. K. Liu, X. Q. Wang, S. Z. Bao, M. Gombosi, and B. Zalik, "An algorithm for polygon clipping, and for determining polygon intersections and unions," *Comput. Geosci.*, vol. 33, no. 5, pp. 589–598, 2007.

[13] A. Margalit and G. D. Knott, "An algorithm for computing the union, intersection or difference of two polygons," *Comput. Graph.*, vol. 13, no. 2, pp. 167–183, 1989.

[14] M. F. Mokbel and W. G. Aref, "SOLE: Scalable on-line execution of continuous queries on spatio-temporal data streams," *VLDB Journal*, vol. 17, no. 5, pp. 971–995, 2008.

[15] H. Mokhtar, J. Su, and O. H. Ibarra, "On moving object queries," *Int. Symp. Principles Database Syst.*, 2002, pp. 188–198.

[16] D. Pfoser and C. S. Jensen, "Capturing the uncertainty of moving-object representations," in *Proc. Int. Symp. Large Spatial Databases*, 1999, pp. 111–132.

[17] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch, "Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects," *IEEE Trans. Comput.*, vol. 51, no. 10, pp. 1124–1140, 2002.

[18] A. Rappoport, "An efficient algorithm for line and polygon clipping," *Vis. Comput.*, vol. 7, no. 1, pp. 19–28, 1991.

[19] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao, "Modeling and querying moving objects," in *Proc. IEEE Int. Conf. Data Eng.*, 1997, pp. 422–432.

[20] Y. Tao, X. Xiao, and R. Cheng, "Range search on multidimensional uncertain data," *ACM Trans. Database Syst.*, vol. 32, no. 3, article 15, 2007.

[21] G. Trajcevski, A. N. Choudhary, O. Wolfson, L. Ye, and G. Li, "Uncertain range queries for necklaces," in *Proc. Int. Conf. Mobile Data Manage.*, 2010, pp. 199–208.

[22] G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain, "Managing uncertainty in moving objects databases," *ACM Trans. Database Syst.*, vol. 29, no. 3, pp. 463–507, 2004.

[23] B. R. Vatti, "A generic solution to polygon clipping," *Commun. ACM*, vol. 35, no. 7, pp. 56–63, 1992.

[24] H. Wang and R. Zimmermann, "Processing of continuous location-based range queries on moving objects in road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 7, pp. 1065–1078, 2011.

[25] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha, "Updating and querying databases that track mobile units," *Distrib. Parallel Databases*, vol. 7, no. 3, pp. 257–387, 1999.

[26] K.-L. Wu, S.-K. Chen, and P. S. Yu, "Incremental processing of continual range queries over moving objects," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 11, pp. 1560–1575, 2006.

[27] M. Zhang, S. Chen, C. S. Jensen, B. C. Ooi, and Z. Zhang, "Effectively indexing uncertain moving objects for predictive queries," *Proc. Very Large Database Endowment*, vol. 2, no. 1, pp. 1198–1209, 2009.

[28] R. Zhang, H. V. Jagadish, B. T. Dai, and K. Ramamohanarao, "Optimized algorithms for predictive range and KNN queries on moving objects," *Info. Syst.*, vol. 35, no. 8, pp. 911–932, 2010.

[29] K. Zheng, G. Trajcevski, X. Zhou, and P. Scheuermann, "Probabilistic range queries for uncertain trajectories on road networks," in *Proc. Int. Conf. Extending Database Technol.*, 2011, pp. 283–294.

**Zhi-Jie Wang** is currently the fourth-year PhD candidate in the department of computer science and engineering, Shanghai Jiao Tong University, China. His research interests include query processing on uncertain data, management of moving object data, scalable data analytics, algorithm design and analysis in computational geometry, and spatial databases.
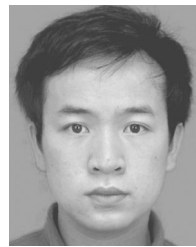
**Dong-Hua Wang** received the MS degree in computer science from Shanghai Jiao Tong University, China, 2013. He is currently working at NetEasy company as a software engineer. His research interests include programming language, distributed computing, and algorithm design and analysis for scalable data.

**Bin Yao** received the PhD degree in computer science from the Florida State University. He is currently an associate professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His research interests include management and indexing of large databases, query processing in spatial and multimedia databases, string and keyword search, and scalable data analytics.

**Minyi Guo** received the PhD degree in computer science from the University of Tsukuba, Japan. He is currently a chair professor and the head of the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His research interests include, but not limited to, parallel and distributed computing, compiler optimizations. He is on the editorial board of the journals *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He is a senior member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.