Probabilistic Convex Hull Queries over Uncertain Data

Da Yan[1], Zhou Zhao[2], Wilfred Ng[3], and Steven Liu[4]

Abstract—The convex hull of a set of two-dimensional points, P, is the minimal convex polygon that contains all the points in P. Convex hull is important in many applications such as GIS, statistical analysis and data mining. Due to the ubiquity of data uncertainty such as location uncertainty in real-world applications, we study the concept of convex hull over uncertain data in 2D space. We propose the *Probabilistic Convex Hull* (PCH) query and demonstrate its applications, such as Flickr landscape photo extraction and activity region visualization, where location uncertainty is incurred by GPS devices or sensors. To tackle the problem of possible world explosion, we develop an $O(N^3)$ algorithm based on geometric properties, where N is the data size. We further improve this algorithm with spatial indices and effective pruning techniques, which prune the majority of data instances. To achieve better time complexity, we propose another $O(N^2 \log N)$ algorithm, by maintaining a probability oracle in the form of a circular array with nice properties. Finally, to support applications that require fast response, we develop a Gibbs-sampling-based approximation algorithm which efficiently finds the PCH with high accuracy. Extensive experiments are conducted to verify the efficiency of our algorithms for answering PCH queries.

Index Terms—Convex hull, uncertain data, Gibbs sampling.

1 Introduction

Convex hull is a geometric concept that is fundamental to a wide spectrum of applications, including pattern recognition [2], cluster analysis [3] and linear optimization [4]. In 2D plane, the convex hull of a set of points, P, is the minimal convex polygon that contains all the points in P. Figure 1(a) illustrates the concept of convex hull, where solid points are those on the convex hull and hollow ones are not. In this work, we focus on objects with 2D location uncertainty, which are common in geo-spatial applications [8].

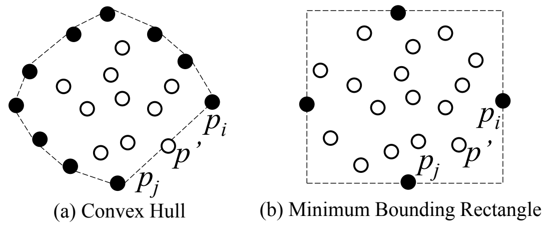


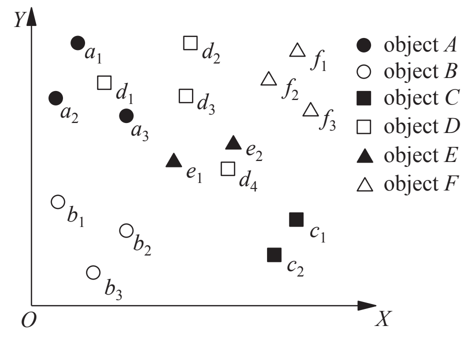
Fig. 1. Convex hull and MBR in 2D plane

Convex Hull in 2D Space. Let n be the number of points in set P. We consider the non-degenerated case with $n > 1$. Referring to Figure 1(a) again, if we order the points on the convex hull clockwise, then each point p_i has a successor $succ(p_i) = p_j$, or equivalently, each point p_j has a predecessor $pred(p_j) = p_i$. Notably, even though p' is on segment p_ip_j, it is not a point on the convex hull as it does not decide the polygon shape. ■

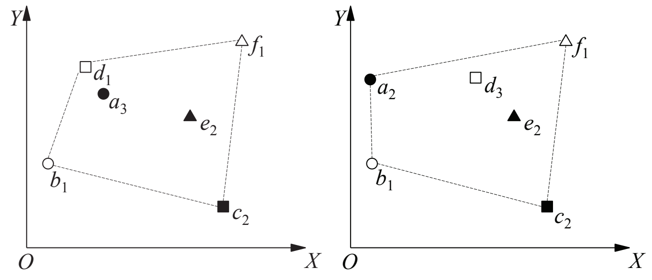
• *Da Yan, Zhou Zhao and Wilfred Ng are with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. Steven Liu is with the Department of Computer Science, Stony Brook University.*
E-mails: {[1]yanda, [2]zhaozhou, [3]wilfred}@cse.ust.hk, [4]kiliu@cs.stonybrook.edu

A lot of algorithms have been proposed for convex hull computation, such as Andrew's Monotone Chain algorithm [6], which finds the convex hull of a set of 2D points in $O(n \log n)$ time. Existing algorithms assume that data points are certain. However, data collected in real applications may be imprecise due to environment factors, device limitations and privacy issues. This is especially true for those real-life applications involving 2D location uncertainty. For example, in an RFID indoor positioning and tracking system, RFID readers are deployed at fixed indoor positions [29]. If a person walks close to a reader, he/she will be detected by the reader and the system decides that the person is at the location near the reader. However, since the detection range of RFID readers changes continuously with environment factors, the person may also be detected by another reader a little bit farther away, causing location uncertainty. Another example is animal tracking [15], where sensors are deployed in the wild, and animals are implanted with microchips indicating their identities. An animal may be detected by several nearby sensors at different locations within some time period, causing location uncertainty.

Probabilistic Convex Hull. In this paper, we propose the concept of *probabilistic convex hull* (or PCH) over a set of objects that are specified by uncertain 2D coordinates, and design efficient algorithms for PCH evaluation. We now illustrate the concept of PCH using a set of six uncertain objects A–F shown in Figure 2(a), where each object has several location instances that represent its possible position. For example, object A may occur at location a_1, a_2 or a_3. Each uncertain object has a certain probability to be on the convex hull. For example, Figures 2(b) and (c) illustrate two possible worlds of the uncertain object dataset presented in Figure 2(a). In the first possible world, object A is not on the convex hull and object D is on the convex hull; while in



(a) A Set of Uncertain Objects



(b) Possible World 1 (c) Possible World 2

Fig. 2. Uncertain Object DB & Two Possible Worlds

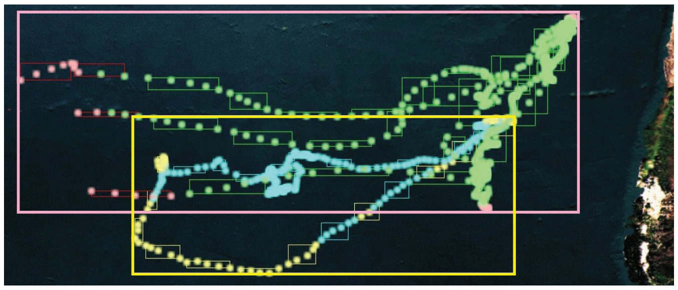
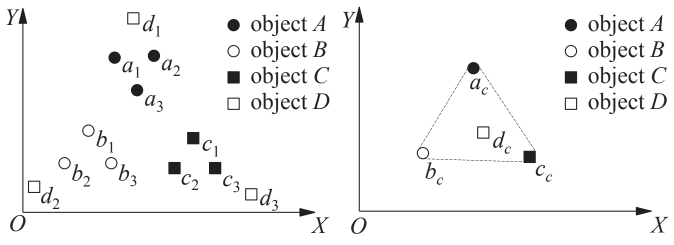


Fig. 3. Visualization of the Habitats of Two Species



(a) Uncertain Object Database (b) Convex Hull of Centroids

Fig. 4. Problem with Centroid-Based Method

the second possible world, object A is on the convex hull and object D is not. ■

We study how to compute the probability that an object o is on the convex hull, denoted as $\Pr^{CH}(o)$. Given a user-specified probability threshold α, we also study how to find all the objects with $\Pr^{CH}(o) \geq \alpha$, which compose the PCH.

Motivations. As a fundamental geometric operation, computing convex hull in the presence of data uncertainty is an important research problem in its own right. While this problem is not well explored yet, many other fundamental geometric operations have already been studied over uncertain data, such as range queries [9], nearest neighbor (NN) queries [8], [9], group nearest neighbor queries [10], reverse nearest neighbor queries [11], skyline queries [1], [14], Voronoi diagram [12] and clustering [31].

Besides, PCH query is especially useful for spatial applications involving 2D location uncertainty, since it finds objects that are very likely to be on the boundary of an object set. By posing PCH queries over objects satisfying different constraints, we are able to visualize the spatial relationship between different sets of objects. We have developed a system for PCH visualization in the application of animal tracking [15], where animals on PCHs are highlighted so that the boundary of a region that a species spans becomes apparent.

Figure 3 shows two PCH results displayed by the animal tracking system [15], for the animal tracking data of two species, Pacific bluefin tuna and Northern elephant seal, collected from GTOPP[1]. The locations of a tuna (or respectively, a seal) are marked green (or respectively, blue) if $\Pr^{CH}(o) = 0$, and they are marked pink (or respectively, yellow) if $\Pr^{CH}(o) > 0$. As Figure 3 shows, the habitats of both species exhibit a high spatial correlation. This observation helps zoologists to explore the potential relationships between the two species, such as the predator-prey relationship.

In the above example, the PCH nicely characterizes the boundary of animal habitats. As a comparison, we also show the *minimum bounding rectangle* (MBR) of all locations of the tuna species (and respectively, the seal species) in Figure 3. Apparently, MBR covers a much larger region than the actual habitats, and is not sufficiently descriptive on the habitat shape. Moreover, even without data uncertainty, there are at most four points on the MBR boundary in most cases, one on each edge of the MBR (see Figure 1(b)). When data uncertainty exists, the MBR is even less expressive as the few points on the MBR boundary are not even 100% certain. In our Flickr photo filtering application, MBR finds at most four photos in most case, and some of them may not be about natural landscapes.

Also note that the simple method of (1) finding the centroid of the instances for each object, and then (2) computing the convex hull of the centroids, cannot serve the same purpose as PCH queries. We now show a counterexample using the object set presented in Figure 4(a): no matter where the location of D is (i.e. at d_1, d_2 or d_3), it is always on the convex hull; however, as shown in Figure 4(b), the instance centroid of object D is not on the convex hull. Thus, probability plays an important role in expressing the boundary of an uncertain object set.

In addition to visualizing the spatial distribution of an object set selected by non-spatial constraints, PCH queries can also be employed to find useful non-spatial information from an object set selected by spatial constraints. An example of such an application is Flickr photo filtering, which we describe next.

Flickr Photo Filtering. Suppose that one wants to collect some photos about the landscape of Hong Kong from the Web. Searching by non-spatial constraints like keywords is not very effective. For example, keywords "Hong Kong" may

1. http://gtopp.org

TABLE 1
Flickr Photo Tags on the Convex Hull

Prob. on Convex Hull	Keywords in Tag
100%	alone, hiking, stairs
55%	disney
45%	butterfly, insect
42.7%	flower, park
39.7%	outlyingislands, weatherbug, sea, fishmen
34.6%	bike
25%	beach, sunrise
23.3%	lantauisland, hiking, fruit

end up returning photos about stylish shopping malls, while expanding the search by "landscape" may have a poor recall, as many relevant photos are tagged by other words like "hiking".

We observe that natural landscapes are mostly found in the outskirt of Hong Kong, and successfully find many landscape photos using an PCH query over the geo-tagged photos crawled from Flicker[1]. Specifically, we use Flickr API to obtain the collection of geo-tagged photos in Hong Kong, using a circular query window with Victoria Peak as the center and a 25km radius.

Flickr users usually tag a collection of photos they take in a trip with the same set of keywords. Moreover, the locations geo-tagged by these photos exhibit spatial locality. Therefore, we regard each tag as an uncertain object, and regard its corresponding geo-tagged photos as its instances. Our proposed algorithm computes all those objects o with $Pr^{CH}(o) > 0$ in seconds. Table 1 illustrates some obtained objects in the result. For example, the first tag has $Pr^{CH}(o) = 100\%$ and all its corresponding photos are about mountain trails for hiking; while the last tag has $Pr^{CH}(o) = 23.3\%$ and most of its corresponding photos are about the landscapes in an island called Lantau. ■

Contributions. We summarize our contributions as follows:

- We propose and formally define the concept of *probabilistic convex hull* (PCH) over a set of uncertain objects.
- A polynomial-time algorithm is proposed to compute PCH over an uncertain database of size N, whose time complexity is $O(N^3)$. The algorithm is able to tackle the problem of "possible world explosion", and its efficiency is further boosted by our effective pruning techniques.
- A batch-evaluation technique is developed that improves the time complexity to $O(N^2 \log N)$.
- To support fast response, we develop an approximation algorithm based on Gibbs sampling, which efficiently finds the PCH with reasonable accuracy.

Organization. The rest of the paper is organized as follows: we review the related work in Section 2. Our uncertain data model and the concept of PCH are formally defined in Section 3. In Section 4, we present our baseline algorithm for computing PCH. Then, several effective pruning techniques are introduced in Section 5. Our batch-evaluation technique is described in Section 6, and the Gibbs sampling method is presented in Section 7. Finally, we report the experimental results on efficiency in Section 8, and conclude our paper in Section 9.

1. http://www.flickr.com

2 Related Work

2.1 Queries over Uncertain Data

Recent research proposes to consider uncertainty as a "first-class citizen" in a DBMS. Various probabilistic DBMSs have already been developed to support the storage and querying of these uncertain data, including MystiQ[18], Trio[19], ORION[20], MayBMS[21]. Two models are popular in representing uncertain data: tuple-level uncertainty model where each database tuple has an occurrence probability, and attribute-level uncertainty model where the value of each data object is specified by a probability distribution. The data model considered in this paper conforms to the attribute-level model, where each data object o has a set of possible instance values $s \in o$, each with occurrence probability $p(s)$.

One of the most fundamental types of queries over uncertain data are spatial queries, since location uncertainty is common in real world applications. Consider, for example, the data collected by GPS devices and sensors, where measurement errors are inevitable. Existing spatial queries that are studied in the context of uncertain data include range queries [9], nearest neighbor (NN) queries [8], [9], group nearest neighbor queries [10], reverse nearest neighbor queries [11], skyline queries [1], [14], Voronoi diagram [12] and clustering [31]. Like in this paper, the data model of uncertain objects adopted by those works are also defined in Euclidean space (often 2D space), and the query semantics are also defined following the possible world semantics. The work most related to this paper is [13], which proposes to find the *most likely convex hull* (MLCH). However, the most likely possible world is sometimes not robust [23], in the sense that its occurrence probability can still be very small as there are many possible worlds. Our definition of PCH is more robust, since we require that each object in the result has a reasonable amount of probability to be on the convex hull.

Another important branch of queries over uncertain data are top-k queries. Different semantics are proposed based on the possible world model, such as U-Top$k$ [22], PT-k [23], U-pop$k$ [25] and PRF [24], where the interplay between high score and high occurrence probability is defined differently. Interestingly, U-Top$k$ returns the most probable top-k tuples that belong to a valid possible world, which is similar to the idea of defining MLCH. In contrast, PT-k returns all tuples whose probability values of being in the top-k answers in possible worlds are above a threshold. The idea of imposing a probability threshold is also adopted in our definition of PCH.

2.2 The R-Tree Index

Since our algorithms use the R-tree as a spatial index, we briefly review the concept of R-tree [27].

Figure 5 shows a 2D point set $P = \{p_1, p_2, \ldots, p_{12}\}$ indexed by an R-tree assuming a capacity of three entries

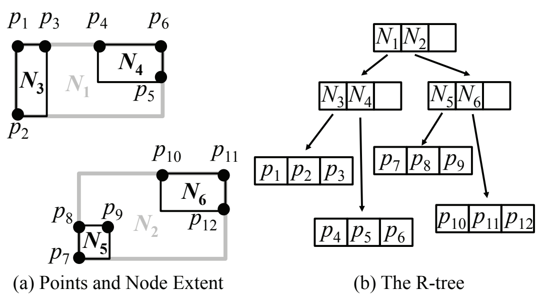


Fig. 5. R-Tree Illustration

per node. Points that are close in space (e.g., p_1, p_2, p_3) are clustered in the same leaf node (N_3). Nodes are then recursively grouped together with the same principle until the top level, which consists of a single root. An intermediate index entry contains the minimum bounding rectangle (MBR) of its child node, together with a pointer to the node. A leaf entry stores the coordinates of a data point.

3 Problem Definition

In this section, we present our uncertain data model and formally define the concept of PCH based on the possible world semantics.

Uncertain Data Model. We adopt the *multi-instance data model* to represent the uncertain objects, which is also used in previous studies such as [1]. We assume that a database is composed of a set of uncertain objects $O = \{o_1, o_2, \ldots, o_n\}$, and each object o_i is represented by a set of m_i instances, which we denote as $o_i = \{s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(m_i)}\}$. Since we only consider 2D objects, each instance is represented by 2D coordinates (x, y).

Each object instance $s_i \in o_i$ is associated with an occurrence probability $p(s_i)$. To keep our presentation simple, we assume that (1) the uncertain objects are independent of each other, and that (2) for each uncertain object, its instances are mutually exclusive and $\sum_{\ell=1}^{m_i} p(s_i^{(\ell)}) = 1$.

The multi-instance data model is a popular representation of real-world objects with uncertain attribute values. For example, in the animal tracking application, an animal may be detected by different sensors at different locations. This is because the detection range of a sensor is sensitive to environmental factors such as changes in temperature and humidity. In this case, the animal is an object o_i whose location attribute is depicted by the locations $s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(m_i)}$ of the sensors that detect the animal. If we follow the idea of probabilistic databases, and regard each object instance as a tuple, a dataset that conforms to the multi-instance data model is actually a block-independent-disjoint (BID) database [26]. For those applications where the attribute values are modeled by a continuous *probability density function* (pdf), our model is still applicable: we can draw a certain number of samples (as instances) for each object according to the pdf to approximate the true distribution. These samples have the same occurrence probability $p(s_i) = \frac{1}{m_i}$.

From now on, we assume that for any two different objects o_i and o_j, there do not exist two instances $s_i \in o_i$ and $s_j \in o_j$ referring to exactly the same location. This is because convex hull is defined over points with different coordinates. However, this may not hold in real world applications. For example, different geo-tagged photos on Flickr may have the same GPS coordinates. In this case, we perturb the location of each instance by a very small random noise, which is chosen independently for each instance from the same noise distribution such as Gaussian. Perturbation is a common technique in computational geometry to handle degeneracies [6], and it is unlikely to have two instances with the same location after the pertubation. On the other hand, since the noise is very small, its impact on the application semantics is negligible.

Query Semantics. We denote by $\Pr^{CH}(o)$ the probability that object o occurs on the convex hull, and define the PCH query as follows:

Definition 1: Given a set of objects O, the probabilistic convex hull query with probability threshold α returns the set of objects $PCH_\alpha(O) = \{o \in O \,|\, \Pr^{CH}(o) \geq \alpha\}$.

PCH is a natural extension of traditional convex hull to the context of uncertain data: when O is deterministic, $\Pr^{CH}(o)$ is either 0 or 1, and thus $PCH_\alpha(O)$ contains exactly those objects on the convex hull of O for arbitrary $\alpha \in (0, 1]$.

One might have the concern that PCH does not capture the geometric shape of convex hulls. However, the problem of finding all possible convex hulls, even without the corresponding probabilities, is $\#\mathcal{P}$-hard. To circumvent this high complexity, the work [13] opts to find the most likely convex hull (MLCH). The problem with this approach is that, the MLCH has very small occurrence probability and many objects with large $\Pr^{CH}(o)$ may not appear in the MLCH. In contrast, PCH computes, for each uncertain object, the probability that the object is part of the convex hull, which gives rise to a more robust result. Our definition of PCH shares a similar spirit with the previous work in probabilistic skyline [1], [14].

4 Baseline Algorithm

In this section, we consider how to compute $\Pr^{CH}(o)$ for an object $o \in O$. A naïve approach is by evaluating

$$\Pr^{CH}(o) = \sum_{pw} \Pr^{CH}(o|pw) \cdot p(pw),$$

where pw is a possible world of O with occurrence probability $p(pw)$, and $\Pr^{CH}(o|pw)$ is the probability that o is on the convex hull in pw. Note that $\Pr^{CH}(o|pw)$ is either 0 or 1.

However, this naïve approach is intractable, since there are $\prod_{i=1}^{n} m_i$ possible worlds. We develop a polynomial-time algorithm to compute PCH, based on a simple observation about 2D convex hull illustrated by Figure 1(a): each point p_i on the convex hull has a unique successor p_j clockwise, and any other point p' is either on segment $p_i p_j$ or in the half plane bounded by line $\overline{p_i p_j}$ that makes $\angle p_i p_j p'$ clockwise.

We now describe an $O(N^3)$-time algorithm that computes the PCH in 2D Euclidean space, where $N = \sum_{i=1}^{n} m_i$.

4.1 Baseline Algorithm

To compute the PCH, we compute $\Pr^{CH}(o)$ for all $o \in O$, which consists of three levels to be described next. We will describe pruning rules that avoid unnecessary probability computation in Section 5.

Level 1: o_i-level. Let us denote by $\Pr^{CH}(s_i)$ the (conditional) probability that instance $s_i \in o_i$ appears on the convex hull, under the condition that o_i occurs as s_i. By the law of total probability, we obtain the following expression:

$$\Pr^{CH}(o_i) = \sum_{s_i \in o_i} \Pr^{CH}(s_i) \cdot p(s_i) \overset{\circ}{=} \frac{1}{m_i} \sum_{s_i \in o_i} \Pr^{CH}(s_i), \tag{1}$$

where the expression after "$\overset{\circ}{=}$" refers to the following special case: for any object, all its instances carry the same occurrence probability. For each equation hereafter, we will show the expression for the special case, right after the expression for the general case that uses instance probabilities $p(s)$ given by the data.

Equation (1) decomposes the computation of $\Pr^{CH}(o_i)$ into the computation of $\Pr^{CH}(s_i)$ for all instances $s_i \in o_i$, which we describe next.

Level 2: s_i-level. We now consider how to compute $\Pr^{CH}(s_i)$. Let us denote by $\Pr^{CH}(s_i \to s_j)$ the (conditional) probability that $s_j \in o_j$ occurs as the successor of $s_i \in o_i$ clockwise on the convex hull, under the condition that o_i occurs as s_i and o_j occurs as s_j. Since the events $\{s_i \to s_j \mid s_j \in o_j, o_j \in O - \{o_i\}\}$ form a *collectively exhaustive* and *mutually exclusive* partition of the event $\{o_i$ occurs as $s_i \wedge s_i$ is on the convex hull$\}$, we obtain

$$\begin{aligned} \Pr^{CH}(s_i) &= \Pr(s_i \to s_j | o_i = s_i) \\ &= \sum_{s_j \in o_j, o_j \in O - \{o_i\}} \Pr^{CH}(s_i \to s_j) \cdot p(s_j) \quad (2) \\ &\overset{\circ}{=} \sum_{s_j \in o_j, o_j \in O - \{o_i\}} \frac{1}{m_j} \Pr^{CH}(s_i \to s_j). \quad (3) \end{aligned}$$

Equation (2) decomposes the computation of $\Pr^{CH}(s_i)$ into the computation of $\Pr^{CH}(s_i \to s_j)$ for all $s_j \in o_j$, for all $o_j \in O - \{o_i\}$.

We describe how to compute $\Pr^{CH}(s_i \to s_j)$ next.

Level 3: s_j-level. Figure 6 illustrates how to compute $\Pr^{CH}(s_i \to s_j)$. Here, s_i refers to instance a_3 of object $o_i = A$, and s_j refers to instance f_3 of object $o_j = F$.

For $a_3 \to f_3$ to be true, it is obvious that objects B, C, D and E cannot occur above line $\overline{a_3 f_3}$. Specifically, D can only occur as d_4 but not the other three instances, while there is no requirement for B, C and E, since all their instances are below line $\overline{a_3 f_3}$. Therefore, assume that for any object, all its instances are equally likely to occur, we have $\Pr^{CH}(a_3 \to f_3) = 1 \times 1 \times \frac{1}{4} \times 1 = 0.25$.

We now formalize the above discussion and present the equation for computing $\Pr^{CH}(s_i \to s_j)$ in $O(N)$ time. We first introduce the *ccw indicator* that decides the validity of an instance given $s_i \to s_j$.

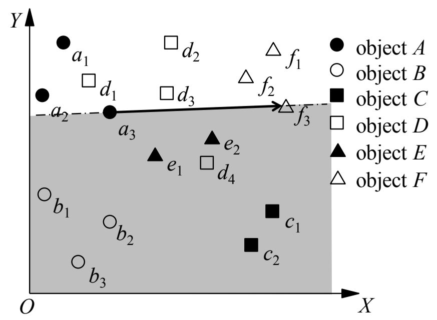


Fig. 6. Illustration of Computing $\Pr^{CH}(s_i \to s_j)$

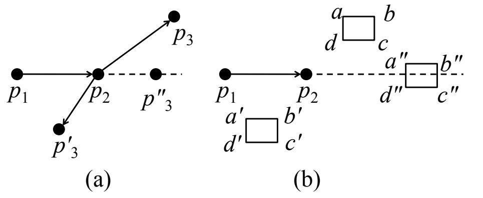


Fig. 7. Illustration of CCW Indicator

Definition 2: Given three points $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ and $p_3 = (x_3, y_3)$, the ccw indicator of p_1, p_2 and p_3 is defined as $ccw(p_1, p_2, p_3) = (x_2 - x_1) \cdot (y_3 - y_1) - (x_3 - x_1) \cdot (y_2 - y_1)$.

The ccw indicator has the following property [6]:

Theorem 1: Given three points p_1, p_2 and p_3, they are in counter-clockwise order if $ccw(p_1, p_2, p_3) > 0$, in clockwise order if $ccw(p_1, p_2, p_3) < 0$, and on the same line if $ccw(p_1, p_2, p_3) = 0$.

Figure 7(a) illustrates the underlying idea of Theorem 1, where line $\overline{p_1 p_2}$ divides the whole space into two half-planes. For a point p_3 in the upper half-plane, $ccw(p_1, p_2, p_3) > 0$; for a point p'_3 in the lower half-plane, $ccw(p_1, p_2, p'_3) < 0$; for a point p''_3 on line $\overline{p_1 p_2}$, $ccw(p_1, p_2, p''_3) = 0$.

We also extend the concept of ccw indicator to deal with the MBR of a set of points, which is used for R-tree [27] node pruning.

Definition 3: Given two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, and an MBR M with four vertices a, b, c and d, the *ccw indicator* of p_1, p_2 and M is defined as follows:

- $ccw(p_1, p_2, M) = 1$, if $ccw(p_1, p_2, v) > 0$ for all $v \in \{a, b, c, d\}$.
- $ccw(p_1, p_2, M) = -1$, if $ccw(p_1, p_2, v) < 0$ for all $v \in \{a, b, c, d\}$.
- Otherwise, $ccw(p_1, p_2, M) = 0$.

Figure 7(b) illustrates the idea of Definition 3: for an MBR $M = \square abcd$ which is totally contained in the upper half-plane, $ccw(p_1, p_2, M) = 1 > 0$; for an MBR $M = \square a'b'c'd'$ which is totally contained in the lower half-plane, $ccw(p_1, p_2, M) = -1 < 0$; for an MBR $M = \square a''b''c''d''$ which intersects with line $\overline{p_1 p_2}$, $ccw(p_1, p_2, M) = 0$.

Now, we are ready to present the computation of $\Pr^{CH}(s_i \to s_j)$. Figure 8(a) shows all the three possible cases for an object $o \in O - \{o_i, o_j, o_k\}$ when computing

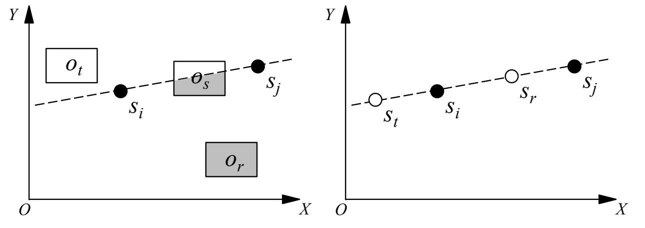


Fig. 8. Different Cases of $\Pr^{CH}(s_i \to s_j)$ Computation

$\Pr^{CH}(s_i \to s_j)$, where we temporarily ignore the possibility that some object instance lies on line $\overline{s_is_j}$.

Let us define the following set

$$V_{(s_i \to s_j)}(o) = \{s \in o | ccw(s_i, s_j, s) < 0\}, \quad (4)$$

and accordingly, $\overline{V_{(s_i \to s_j)}(o)} = \{s \in o | ccw(s_i, s_j, s) > 0\}$. We omit the subscript $s_i \to s_j$ and use lighter notations $V(o)$ and $\overline{V(o)}$ when s_i and s_j are clear from the context.

Intuitively, $V(o)$ is the set of instances $s \in o$ below line $\overline{s_is_j}$ in Figure 8(a), and $\overline{V(o)}$ is the set of instances $s \in o$ above the line.

We now analyze the three cases in Figure 8(a) as follows:

- If there exists an object $o_t \in O - \{o_i, o_j\}$, such that $\forall s_t \in o_t, s_t \in \overline{V(o_t)}$, then $\Pr^{CH}(s_i \to s_j) = 0$ since no matter which instance of o_t occurs, $s_i \to s_j$ is impossible as $\angle s_ts_is_j$ forms a concave angle already. We define this case as $V(o_t)$***-empty***.
- If $o_r \in O - \{o_i, o_j\}$ satisfies $\forall s_r \in o_r, s_r \in V(o_r)$, then no matter which instance of o_r occurs, it does not contradict with the event $s_i \to s_j$. We define this case as $V(o_r)$***-full***.
- If $o_s \in O - \{o_i, o_j\}$ satisfies $V(o_s) \neq \emptyset \;\wedge\; \overline{V(o_s)} \neq \emptyset$, then in order to make $s_i \to s_j$ hold, o_s can only occur as an instance in $V(o_s)$. We define this case as $V(o_s)$***-valid***.

Now, let us take into consideration the special cases when there exists some instance $s \in o$ on line $\overline{s_is_j}$. Figure 8(b) illustrates such cases: instance s_r is on segment s_is_j, and its occurrence does not contradict $s_i \to s_j$; on the other hand, instance s_t occurs on line $\overline{s_is_j}$, but outside of segment s_is_j, and its occurrence disqualifies s_i from being on the convex hull. Let us define the predicate $onSeg_{p_1p_2}(p_3)$ to indicate whether p_3 is on segment p_1p_2. Then, we extend the definition of $V(o)$ in Equation (4) to include the special cases as follows:

$$V(o) = \{\ s \in o \mid ccw(s_i, s_j, s) < 0 \vee onSeg_{s_is_j}(s)\}. \quad (5)$$

Finally, under the condition that s_i and s_j occur, we can see that $s_i \to s_j$ occurs iff $\forall o \in O - \{o_i, o_j\}$, o occurs as an instance in $V(o)$. As a result, we compute $\Pr^{CH}(s_i \to s_j)$ by the following formula:

$$\begin{aligned}\Pr^{CH}(s_i \to s_j) &= \prod_{o_t \in O - \{o_i, o_j\}} \Pr(s_t \in V(o_t)) \\ &= \prod_{o_t \in O - \{o_i, o_j\}} \left(\sum_{s_t \in V(o_t)} p(s_t) \right) \quad (6) \\ &\doteq \prod_{o_t \in O - \{o_i, o_j\}} \frac{|V(o_t)|}{m_t}. \quad (7)\end{aligned}$$

According to Equation (6), $\Pr^{CH}(s_i \to s_j)$ can be computed in $O(N)$ time. Specifically, for each object o_t with m_t instances, $V(o_t)$ can be obtained by checking each instance of o_t using Equation (5). Each checking takes $O(1)$ time since both $ccw(.)$ and $onSeg(.)$ operations in Equation (5) take constant time. Therefore, in Equation (6), the summation in the parentheses takes $O(m_t)$ time (note that $p(s_t)$ is given in the data). As a result, Equation (6) requires computing the summation for each object which takes $\sum_t O(m_t) = O(\sum_t m_t) = O(N)$ time, while computing the product of the summations of the $O(n)$ objects only requires $O(n)$ multiplication operations. Totally, the cost is $O(N) + O(n) = O(N)$.

Complexity Analysis. According to Equation (1), computing $\Pr^{CH}(o_i)$ requires computing $\Pr^{CH}(s_i)$ for m_i times. By Equation (2), computing $\Pr^{CH}(s_i)$ requires computing $\Pr^{CH}(s_i \to s_j)$ for N times. Finally, by Equation (6), computing each $\Pr^{CH}(s_i \to s_j)$ takes $O(N)$ time. Thus, it takes $O(m_i \times N^2)$ time to compute each $\Pr^{CH}(o_i)$. Since the computation of $PCH_\alpha(O)$ requires computing $\Pr^{CH}(o_i)$ for at most all objects $o_i \in O$, the time complexity is $\sum_{i=1}^{n} O(m_i \cdot N^2) = O(N^3)$.

5 Four-Corner Pruning & Bounding

In this section, we first present our spatial indices on the data. Then, we present our *four-corner pruning* techniques that prune objects and instances with zero probability to occur on the convex hull. The techniques can be employed to prune the majority of the search space. Finally, we propose our *four-corner upper bounding* technique to derive the upper bounds of $\Pr^{CH}(s_i)$ and $\Pr^{CH}(o_i)$, which is effective in search space pruning when computing $PCH_\alpha(O)$.

5.1 Spatial Indices

We first build main-memory spatial indices on the dataset, to support efficient spatial operations used in our algorithms. Besides R-tree, we also use aggregate R-tree (aR-tree) [7], which we describe next. An aR-tree is an R-tree extended with a specific aggregate function (e.g. *MAX, SUM, COUNT*). Each node $\mathcal{N}$ of an aR-tree maintains the aggregated value computed over all the data indexed under $\mathcal{N}$.

To support efficient PCH evaluation, we build the following spatial indices on the data:

- **Object R-tree** T_O**.** Let us denote the MBR of all the instances $s \in o$ as $o.M$. The global object R-tree T_O is bulk-loaded over $o.M$ for all $o \in O$.
- **Instance aR-tree** aR_o**.** For each object $o \in O$, we bulk-load an aggregate R-tree aR_o on all the instances $s \in o$, where the aggregate function on node $\mathcal{N}$ is $\sum_{s \in \mathcal{N}} p(s)$.

When the instances of any object are equally likely to occur, we use *COUNT* as the aggregate function of aR_o instead. We choose aR-trees to index object instances, since the computation of $\Pr^{CH}(s_i \to s_j)$ in Equation (6) (or (7)) requires the value of $\sum_{s_t \in V(o_t)} p(s_t)$ (or $|V(o_t)|$), and therefore, if we know that all instances in node $\mathcal{N}$ belong to $V(o_t)$, we can use the aggregate value without accessing the children of $\mathcal{N}$.

We now discuss how to compute $\Pr^{CH}(s_i \to s_j)$ efficiently using our spatial indices. The algorithm is composed of three steps:

1) Traverse T_O to obtain a candidate object set C containing all objects $o_t \in O - \{o_i, o_j\}$ such that $V(o_t)$-valid holds. If we ever find an object o_t such that $V(o_t)$-empty holds, we set $\Pr^{CH}(s_i \to s_j) = 0$ directly and terminate.
2) For each $o_t \in C$, traverse aR_{o_t} to compute $\sum_{s_t \in V(o_t)} p(s_t)$ (or $|V(o_t)|$).
3) Compute $\Pr^{CH}(s_i \to s_j)$ using the following formula:

$$\Pr^{CH}(s_i \to s_j) = \prod_{o_t \in C} \left(\sum_{s_t \in V(o_t)} p(s_t) \right) \quad (8)$$

$$\stackrel{\circ}{=} \prod_{o_t \in C} \frac{|V(o_t)|}{m_t}. \quad (9)$$

Equations (8) and (9) are derived from Equations (6) and (7). Specifically, Step 1 guarantees that for any object $o_t \in O - \{o_i, o_j\}$, $V(o_t)$-empty does not hold. If $V(o_t)$-full holds, we have $\sum_{s_t \in V(o_t)} p(s_t) = 1$ (and $\frac{|V(o_t)|}{m_t} = 1$) and thus, the term can be omitted in the product evaluation in Equations (6) and (7). Otherwise, $V(o_t)$-valid holds and $o_t \in C$, which is counted in Equations (8) and (9).

5.2 Four-Corner Pruning

We now present our *four-corner pruning* techniques that are used to prune objects and instances with zero probability to occur on the convex hull. The effectiveness of the pruning techniques are established by the following two observations:

- If $\Pr^{CH}(o_l) = 0$ $(l = i, j)$, then $\Pr^{CH}(s_i \to s_j) = 0$ for any $s_l \in o_l$;
- If $\Pr^{CH}(s_l) = 0$ $(l = i, j)$, then $\Pr^{CH}(s_i \to s_j) = 0$.

This is because, if either s_i or s_j is not on the convex hull, then $s_i \to s_j$ is impossible. As a result, if we define $\widetilde{O} = \{o \in O | \Pr^{CH}(o) > 0\}$ and $\widetilde{o} = \{s \in o | \Pr^{CH}(s) > 0\}$, then Equation (1) can be reformulated as

$$\Pr^{CH}(o_i) = \sum_{s_i \in \widetilde{o_i}} \Pr^{CH}(s_i) \cdot p(s_i) \stackrel{\circ}{=} \frac{1}{m_i} \sum_{s_i \in \widetilde{o_i}} \Pr^{CH}(s_i), \quad (10)$$

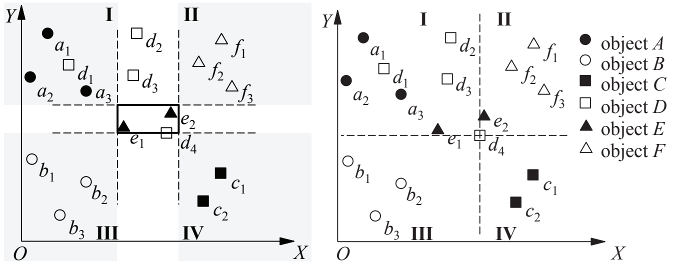
and Equations (2) and (3) can be reformulated as

$$\Pr^{CH}(s_i) = \sum_{s_j \in \widetilde{o_j}, o_j \in \widetilde{O} - \{o_i\}} \Pr^{CH}(s_i \to s_j) \cdot p(s_j) \quad (11)$$

$$\stackrel{\circ}{=} \sum_{s_j \in \widetilde{o_j}, o_j \in \widetilde{O} - \{o_i\}} \frac{1}{m_j} \Pr^{CH}(s_i \to s_j). \quad (12)$$

In a nutshell, we only need to consider the objects and instances with non-zero probability to occur on the convex hull when computing $\Pr^{CH}(o_i)$ and $\Pr^{CH}(s_i)$. Moreover, those objects o with $\Pr^{CH}(o) = 0$ does not belong to $PCH_\alpha(O)$ for any $\alpha > 0$, and can thus be safely ignored when computing $PCH_\alpha(O)$.

Four-Corner Pruning Rules. Given a rectangle with lower-left corner (x_1, y_1) and upper-right corner (x_2, y_2), we define four regions determined by its four corners as follows:



(a) Object-Level Pruning (b) Instance-Level Pruning

Fig. 9. Object/Instance-Level Four-Corner Pruning

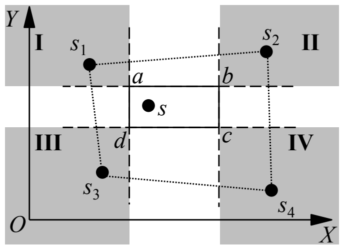


Fig. 10. Proof of Theorem 2

- Region I $= \{p \in \mathbb{R}^2 \mid p.x \le x_1 \ \wedge \ p.y \ge y_2\}$;
- Region II $= \{p \in \mathbb{R}^2 \mid p.x \ge x_2 \ \wedge \ p.y \ge y_2\}$;
- Region III $= \{p \in \mathbb{R}^2 \mid p.x \le x_1 \ \wedge \ p.y \le y_1\}$;
- Region IV $= \{p \in \mathbb{R}^2 \mid p.x \ge x_2 \ \wedge \ p.y \le y_1\}$.

We allow the degenerated case where the rectangle becomes a point. Figure 9(a) (and Figure 9(b)) illustrates the idea of *object-level (and instance-level) four-corner pruning*. In Figure 9(a) the MBR of object E (denoted $E.M$) is shown along with the four regions determined by it. It is clear that $A.M$, $F.M$, $B.M$ and $C.M$ are totally contained in Regions I, II, III and IV, respectively, and thus any point within the region of $E.M$ cannot be on the convex hull. This implies that $\Pr^{CH}(E) = 0$. Similar reasoning for the instance case shown in Figure 9(b) leads to the conclusion that $\Pr^{CH}(d_4) = 0$.

These observations are formalized by Theorem 2 below:

Theorem 2: Given an object o (or respectively, an instance $s \in o$), if for each of the four regions determined by $o.M$ (or respectively, s), there exists an object $o' \neq o$ such that $o'.M$ is totally contained in the region, then $\Pr^{CH}(o) = 0$ (or respectively, $\Pr^{CH}(s) = 0$).

Proof: Given an object o, its MBR $o.M$ determines four regions as shown in Figure 10. Suppose that there exist four objects o_1, o_2, o_3 and o_4, such that $o_1.M$, $o_2.M$, $o_3.M$ and $o_4.M$ are totally contained in Regions I, II, III and IV, respectively. Then, in any possible world pw, s_1, s_2, s_3 and s_4 are totally contained in Regions I, II, III and IV, respectively.

We now prove that the convex hull in pw is a polygon that contains $o.M$, so that any instance $s \in o$ is not on the convex hull. Due to the arbitrariness of pw, we would have $\Pr^{CH}(o) = 0$.

Let us define $O' = \{o_1, o_2, o_3, o_4, o\}$. Since $O' \subseteq O$, the convex hull of O' in pw is a polygon that is contained in the polygon defined by the convex hull of O in pw. Therefore, it is sufficient to prove that the polygon defined by the convex hull of O' in pw contains $o.M$.

Algorithm 1 Preprocessing by Four-Corner Pruning

```
1: for each object $o \in O$ do
2:     $o.pruned \leftarrow FourCornerPrune(o)$
3:     if $o.pruned = FALSE$ then
4:         for each instance $s \in o$ do
5:             $s.pruned \leftarrow FourCornerPrune(s)$
```

We now prove that the convex hull of O' in pw, i.e. polygon $s_1 s_2 s_3 s_4$ in Figure 10, contains $o.M$, i.e. rectangle $abcd$ in Figure 10. Without loss of generality, we only need to prove that polygon edge $s_1 s_2$ is above $o.M$, or equivalently, above rectangle edge ab. This holds because both $s_1.y$ and $s_2.y$ are at least y_2, which accomplishes the proof.

The case of instance-level four-corner pruning can be similarly proved. □

Our four-corner pruning operation is based on four range queries on the object R-tree T_O. We call such range queries as the *containment queries*, which return whether there exists an object whose MBR is totally contained in the query region. As long as one of the four *containment queries* gives a negative answer ("no contained object"), we return a negative answer ("cannot prune") immediately without evaluating the remaining queries.

In our implementation, *four-corner pruning* is designed as a preprocessing step executed only once for each dataset O, the algorithm of which is shown in Algorithm 1. Our experiments show that Algorithm 1 is able to prune the majority of the objects and instances.

5.3 Four-Corner Upper Bounding

For those non-pruned objects o (or instances s), we can still bound $\Pr^{CH}(o)$ (or $\Pr^{CH}(s)$) by using the four-corner technique. Let us first consider how to compute the upper bound of $\Pr^{CH}(s)$ for instance $s \in o$. Given the four regions defined by an instance $s \in o$, we assume that (1) for each $t \in \{\text{I, II, III, IV}\}$, there exists an object o_t with n_t of its m_t instances in Region t, that (2) o_{I}, o_{II}, o_{III} and o_{IV} are different from each other, and that (3) none of them is o. Then, instance s is not on the convex hull as long as o_t is contained in Region t (we denote this fact by $o_t \in$ Region t) for all $t \in \{\text{I, II, III, IV}\}$. Therefore, we have:

$$\begin{aligned}
\Pr^{CH}(s) &= 1 - \overline{\Pr^{CH}(s)} \\
&\leq 1 - \prod_{t \in \{\text{I, II, III, IV}\}} \Pr(o_t \in \text{ Region } t) \quad (13) \\
&= 1 - \prod_{t \in \{\text{I, II, III, IV}\}} \left(\sum_{s_t \in \text{Region } t} p(s_t) \right) \quad (14) \\
&\doteq 1 - \frac{n_{\text{I}}}{m_{\text{I}}} \times \frac{n_{\text{II}}}{m_{\text{II}}} \times \frac{n_{\text{III}}}{m_{\text{III}}} \times \frac{n_{\text{IV}}}{m_{\text{IV}}}. \quad (15)
\end{aligned}$$

Equations (14) and (15) give the upper bound of $\Pr^{CH}(s)$, which we denote by $UB^{CH}(s)$. In order to make $UB^{CH}(s)$ tight, for each Region $t \in \{\text{I, II, III, IV}\}$, we choose objects o_t to maximize $\sum_{s_t \in \text{Region } t} p(s_t)$ (or n_t/m_t). We do not choose an object that has been chosen before.

Algorithm 2 Evaluation of $\Pr^{CH}(o_i)$ for Object $o_i \in \widetilde{O}$

```
1: $\Pr^{CH}(o_i) \leftarrow 0$, $bound \leftarrow UB^{CH}(o_i)$
2: for each non-pruned instance $s_i \in \widetilde{o_i}$ do
3:     Compute $\Pr^{CH}(s_i)$ using Equations (8) and (11)
4:     $\Pr^{CH}(o_i) \leftarrow \Pr^{CH}(o_i) + p(s_i) \cdot \Pr^{CH}(s_i)$
5:     $bound \leftarrow bound - p(s_i) \cdot UB^{CH}(s_i)$
6:     if $bound + \Pr^{CH}(o_i) < \alpha$ then
7:         return $o_i \notin PCH_\alpha(O)$
8: Add $\langle o_i, \Pr^{CH}(o_i) \rangle$ to $PCH_\alpha(O)$
```

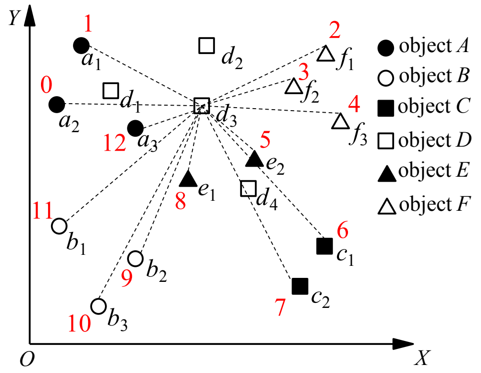


Fig. 11. Oracle A_{d_3}

Once $UB^{CH}(s_i)$ is computed for all $s_i \in o_i$, according to Equation (1), we compute the upper bound of $\Pr^{CH}(o_i)$ as:

$$UB^{CH}(o_i) = \sum_{s_i \in o_i} UB^{CH}(s_i) \cdot p(s_i) \doteq \frac{1}{m_i} \sum_{s_i \in o_i} UB^{CH}(s_i). \quad (16)$$

We call the process of computing $UB^{CH}(o)$ for all $o \in \widetilde{O}$ using Equation (16) as *four-corner upper bounding*, which is also implemented as a preprocessing step, executed immediately after *four-corner pruning* (cf. Algorithm 1).

Baseline Algorithm for Computing $PCH_\alpha(O)$. After *four-corner pruning & upper bounding* are performed, for each non-pruned object $o_i \in \widetilde{O}$ with $UB^{CH}(o_i) \geq \alpha$, we use Algorithm 2 to compute $\Pr^{CH}(o_i)$ and check whether $o_i \in PCH_\alpha(O)$. In Algorithm 2, we compute $\Pr^{CH}(o_i)$ by accumulating the results of $\Pr^{CH}(s_i)$ for all $s_i \in \widetilde{o_i}$ in Line 4, which is according to Equation (10). We also use $UB^{CH}(o_i)$ and $UB^{CH}(s_i)$ to prune o_i in Lines 6–7 whenever $\Pr^{CH}(o_i) < \alpha$ holds.

6 Batch Evaluation Technique

In Section 4, we introduced our baseline algorithm that computes $\Pr^{CH}(s_i)$ by computing $\Pr^{CH}(s_i \to s_j)$ for $O(N)$ times, each time requiring $O(N)$ time. In this section, we show how to compute $\Pr^{CH}(s_i)$ in totally $O(N)$ time rather than $O(N^2)$, which implies that the amortized cost of computing $\Pr^{CH}(s_i \to s_j)$ is $O(1)$.

Assume hereafter that s_i is fixed. Suppose that we have an oracle A_{s_i} that returns $\Pr^{CH}(s_i \to s_j)$ in $O(1)$ time given s_j, then according to Equations (2) and (3), we can obtain $\Pr^{CH}(s_i)$ in $O(N)$ time. We now consider how to construct and maintain the oracle.

Oracle Construction. Given s_i, its oracle is actually a circular array A_{s_i} which contains all the instances $s \in o$ for all the objects $o \in O - \{o_i\}$. All the $O(N)$ instance elements s are radially ordered such that line $\overline{s_i s}$ rotates clockwise around s_i. As an example, Figure 11 illustrates the element ordering in circular array A_{d_3}. Note that A_{d_3} does not contain the instances of object D.

The key to the construction of A_{s_i} is to radially sort the instance elements clockwise. To achieve this, we divide the whole 2D space into two half-planes using line $y = s_i.y$. The instance elements of A_{s_i} are divided into two sets: those in the upper half-plane H_u and those in the lower half-plane H_ℓ. Referring to Figure 11 again, the instance elements of A_{d_3} are divided into two sets $\{a_2, a_1, f_1, f_2\}$ and $\{f_3, e_2, c_1, c_2, e_1, b_2, b_3, b_1, a_3\}$. In each half plane, we define a strict total order among the instance elements of A_{s_i}:

Definition 4: $\forall s_r, s_t \in H_u$ (or H_ℓ), $s_r \prec s_t$ iff $ccw(s_r, s_i, s_t) > 0 \vee onSeg_{s_i s_t}(s_r)$.

Definition 4 actually defines the clockwise radial order for all the instance elements in each half-plane. For example, in Figure 11, a_1 is before f_1 clockwise in H_u because $ccw(a_1, d_3, f_1) > 0$.

After the instance elements in both half-planes are radially sorted, they are concatenated to form A_{s_i}. Thus, the time complexity of constructing A_{s_i} is $O(N \log N)$. In Figure 11, we mark the ID of each instance in array A_{d_3} besides that instance.

Active Domain $V(o_t)$**.** To compute $\Pr^{CH}(s_i \to s_j)$ for all s_j, we iterate s_j from $A_{s_i}[0]$ to $A_{s_i}[|A_{s_i}| - 1]$. We denote by pt_{s_j} the ID of the current s_j in A_{s_i}.

Given s_i and $s_j = A_{s_i}[pt_{s_j}]$, for any object $o_t \in O - \{o_i, o_j\}$, its instance s_t is in $V_{s_i \to s_j}(o_t)$ iff $ccw(s_i, s_j, s_t) < 0 \vee onSeg_{s_i s_j}(s_t)$. Let us iterate A_{s_i} from pt_{s_j} clockwise until reaching the last instance $s_{last} = A_{s_i}[pt_{last}]$ with $ccw(s_i, s_j, s_{last}) < 0 \vee onSeg_{s_i s_j}(s_{last})$. Then, $s_t \in V(o_t)$ iff s_t is within the range wiping from $s_i s_j$ clockwise to $s_i s_{last}$. For example, in Figure 11, when $s_j = f_1$, we have $s_{last} = b_1$ and $pt_{last} = 11$.

Initialization of pt_{last}**.** Initially, $pt_{s_j} = 0$ and we find the corresponding pt_{last} using binary search over $A_{s_i}[0, \cdots, |A_{s_i}| - 1]$. Specifically, in each iteration, we find the two clockwise consecutive elements s_{m_1} and s_{m_2} in the middle, and check $ccw_1 = ccw(s_i, s_j, s_{m_1})$ and $ccw_2 = ccw(s_i, s_j, s_{m_2})$ as follows: (1) if both values are negative, we rule out the elements before s_{m_2}; (2) if both values are non-negative, we rule out the elements after s_{m_1}; and (3) otherwise, we set $s_{last} = s_{m_1}$. Therefore, it takes $O(\log N)$ time to find pt_{last}.

If s_{last} is not found, we know that no instance is within $V(o_t)$ (even for $A_{s_i}[1]$). In this case, $\Pr^{CH}(s_i \to s_j) = 0$ and we set $pt_{last} = 0$.

In later computation, as pt_{s_j} moves clockwise, pt_{last} also moves clockwise and we need no more binary search.

Product. Let us temporarily assume that $|V(o_t)| > 0$ for all $o_t \in O - \{o_i, o_j\}$. If we define

$$\mathbb{P} = \prod_{t \neq i} \left(\sum_{s_t \in V(o_t)} p(s_t) \right) \doteq \prod_{t \neq i} \frac{|V(o_t)|}{m_t}, \tag{17}$$

then according to Equations (6) and (7), we have

$$\Pr^{CH}(s_i \to s_j) = \mathbb{P} / \sum_{s_j \in V(o_j)} p(s_j) \doteq \mathbb{P} / \frac{|V(o_j)|}{m_j}. \tag{18}$$

In other words, we can obtain $\Pr^{CH}(s_i \to s_j)$ in $O(1)$ time if $\mathbb{P}$ is always available. However, in reality, an object o_t may exists such that $V(o_t)$-empty holds. In this case, by Equation (17) we have $\mathbb{P} = 0$ and Equation (18) is no longer valid. We circumvent this problem by maintaining all objects with $V(o_t) = \emptyset$ in a set S_0, and redefine $\mathbb{P}$ as the product of $\sum_{s_t \in V(o_t)} p(s_t)$ for those objects that are not $V(o_t)$-empty:

$$\begin{aligned} \mathbb{P} &= \prod_{o_t \in O - \{o_i\} : |V(o_t)| > 0} \left(\sum_{s_t \in V(o_t)} p(s_t) \right) & (19) \\ &\doteq \prod_{o_t \in O - \{o_i\} : |V(o_t)| > 0} \frac{|V(o_t)|}{m_t}, & (20) \end{aligned}$$

Main Algorithm. To compute $\Pr^{CH}(s_i \to s_j)$ for the next s_j, we move pt_{s_j} clockwise by setting $pt_{s_j} \leftarrow (pt_{s_j} + 1) \mod |A_{s_i}|$. In this case, the old s_j **exits** the active domain.

After the new $s_j = A_{s_i}[pt_{s_j}]$ is updated, we need to set the corresponding pt_{last} properly. We achieve this by moving the old pt_{last} clockwise one position at a time. Let $pt_{next} = (pt_{last} + 1) \mod |A_{s_i}|$ and $s_{next} = A_{s_i}[pt_{next}]$, then we stop moving pt_{last} if $ccw(s_i, s_j, s_{next}) < 0 \vee onSeg_{s_j s_{next}}(s_i)$. Whenever we update pt_{last}, the new s_{last} **enters** the active domain.

Obtaining Probability. Suppose S_0 and $\mathbb{P}$ are up-to-date, we obtain $\Pr^{CH}(s_i \to s_j)$ as follows. **Case (1):** if $|S_0| > 1$, $\Pr^{CH}(s_i \to s_j) = 0$ since there exists an object $o_t \neq o_j$ such that $V(o_t)$-empty holds. **Case (2):** if $S_0 = \{o_j\}$, $\Pr^{CH}(s_i \to s_j) = \mathbb{P}$ which involves all objects other than o_j and o_i. **Case (3):** if $S_0 = \{o_t\}$ but $o_t \neq o_j$, $\Pr^{CH}(s_i \to s_j) = 0$ since $V(o_t)$-empty holds. **Case (4):** if $S_0 = \emptyset$, compute $\Pr^{CH}(s_i \to s_j)$ using Equation (18).

Active Domain Maintenance. We now present how to maintain S_0 and $\mathbb{P}$ up-to-date. Specifically, we maintain an array V such that $V[t] = \sum_{s_t \in V(o_t)} p(s_t)$. We initialize V by scanning through A_{s_i} starting from $pt_{s_j} = 0$.

When an instance s_j exits the active domain, we update $V[o_j] \leftarrow V[o_j] - p(s_j)$. (1) if $V[o_j] = 0$, we add o_j to S_0 and set $\mathbb{P} \leftarrow \mathbb{P}/p(s_j)$ to rule out the old factor $\sum_{s_{j'} \in V(o_j)} p(s_{j'}) = p(s_j)$ from $\mathbb{P}$. (2) otherwise, we first rule out the old factor of o_j by setting $\mathbb{P} \leftarrow \mathbb{P}/(V[o_j] + p(s_j))$, and then incorporate the new factor by setting $\mathbb{P} \leftarrow \mathbb{P} \cdot V[o_j]$.

When an instance s_t enters the active domain, we update $V[o_t] \leftarrow V[o_t] + p(s_t)$. (1) if $o_t \in S_0$, we remove o_t from S_0 and set $\mathbb{P} \leftarrow \mathbb{P} \cdot p(s_t)$. (2) otherwise, we first rule out the old factor of o_t by setting $\mathbb{P} \leftarrow \mathbb{P}/(V[o_t] - p(s_t))$, and then incorporate the new factor by setting $\mathbb{P} \leftarrow \mathbb{P} \cdot V[o_t]$.

Complexity Analysis. We organize S_0 as a balanced binary search tree. As a result, each enter (or exit) operation takes at most $O(\log N)$ time. Since pt_{s_j} (or pt_{last}) moves for at most N times, there are at most N exit (or enter) operations, and thus $O(N \log N)$ time in total. For each value of pt_{s_j}, we obtain the probability $\Pr^{CH}(s_i \to s_j)$ once which takes $O(1)$ time (in Case (4) we compute $\mathbb{P}/V[o_j]$), and $O(N)$ time in total. Finally, constructing A_{s_i} takes $O(N \log N)$ time. Therefore, we can compute $\Pr^{CH}(s_i)$ in totally $O(N \log N)$ time using Equations (2) and (3), and thus $PCH_\alpha(O)$ in $O(N^2 \log N)$ time. Note that we only need to construct oracles for non-pruned instances.

7 Gibbs Sampling Method

Different applications have different requirements on the performance of PCH computation. For Flickr photo filtering, the quality of the photos is more important than the response time. On the other hand, for animal tracking, short response time is critical since the readings are collected continuously, and the PCH should be kept up to date. To support fast response, we propose to estimate $\Pr^{CH}(o)$ for all objects $o \in O$ using Gibbs sampling [16], which usually takes only several minutes.

Gibbs Sampling. Gibbs sampling is a Markov chain Monte Carlo (MCMC) algorithm for obtaining a sequence of random samples from a multivariate probability distribution $p(o_1, o_2, \ldots, o_n)$. The samples can be used to approximate the joint distribution. Suppose that we aim to obtain k samples. Gibbs sampling works as follows, where we denote by $o_j^{(i)}$ the value of variable o_j in the i-th sample.

1) We randomly determine the initial values for all variables: $O^{(0)} = \{o_1^{(0)}, o_2^{(0)}, \ldots, o_n^{(0)}\}$;
2) The i-th sample is obtained by sampling $o_j^{(i)}$ from the conditional distribution $p(o_j^{(i)} | o_1^{(i-1)}, \ldots, o_{j-1}^{(i-1)}, o_{j+1}^{(i-1)}, \ldots, o_n^{(i-1)})$ for $j = i \mod n$, and set $o_\ell^{(i)} = o_\ell^{(i-1)}$ for all $\ell \neq j$. This is repeated until k samples are obtained.

Our Algorithm. In our data model, each variable o_i is now an uncertain object associated with a probability mass function (pmf): $\Pr\{o_i = s_i\} = p(s_i)$. Furthermore, since objects are independent of each other, the conditional distribution $p(o_j^{(i)} | o_1^{(i-1)}, \ldots, o_{j-1}^{(i-1)}, o_{j+1}^{(i-1)}, \ldots, o_n^{(i-1)}) = p(o_j^{(i)})$.

As a result, we obtain a simple Gibbs sampler as follows. The i-th sample is obtained by sampling the instance of o_j using its pmf for $j = i \mod n$, while the instances of the other objects remain unchanged.

To estimate $\Pr^{CH}(o)$ for all objects $o \in O$, we maintain a counter $cnt(o)$ for each object o. Whenever we obtain a new sample $O^{(i)}$, we compute its convex hull and increase the counters of all the objects on the convex hull by one. When k samples are obtained, we estimate $\Pr^{CH}(o)$ as $cnt(o)/k$.

However, it is not efficient to compute the convex hull of each sample from scratch. Since a new sample $O^{(i)}$ is obtained from the previous sample $O^{(i-1)}$, by deleting point $o_j^{(i-1)}$ and inserting point $o_j^{(i)}$ for $j = i \mod n$, we propose to dynamically maintain the convex hull. We adopt the R-tree based approach proposed in [17] for convex hull maintenance[1].

Accuracy Estimation. While $cnt(o)/k$ is an unbiased estimator of $\Pr^{CH}(o)$, the accuracy depends on the variance of the estimation. Clearly, the larger the number of samples k is, the smaller the variance. One method of variance estimation is described in [28], which applies the theory of time series. However, the method requires computing an estimate of lag-k autocovariance, which is not only expensive to compute, but also requires storing previously sampled dataset instances. We adopt a much faster convergence check as follows. We sample 10 non-pruned objects o and compute their exact value of $\Pr^{CH}(o)$. This takes just several seconds. Then, during Gibbs sampling, we periodically (every 1M samples) check the median of the 10 estimation errors of $\Pr^{CH}(o)$ for those samples. Sampling terminates once the estimated average error is smaller than a user-specified error threshold τ.

8 Experiments

In this section, we evaluate the performance of our algorithms for computing $PCH_\alpha(O)$ using synthetic datasets.

For those applications where the datasets O do not change frequently, such as Flickr photo filtering, it is desirable to pre-compute $\Pr^{CH}(o)$ for all non-pruned objects $o \in O$, so that $PCH_\alpha(O)$ can be efficiently obtained for arbitrary α later on. This problem is equivalent to setting α as the infinitesimal positive number ε, since $PCH_\varepsilon(O) = \{o \in O | \Pr^{CH}(o) \geq \varepsilon\} = \{o \in O | \Pr^{CH}(o) > 0\}$. A similar problem was already studied in the context of skyline [14].

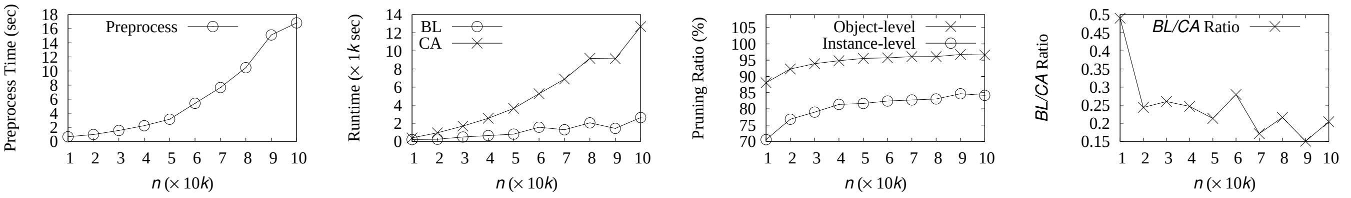
Since we find that the performance of our algorithms is insensitive to instance weights, in all our experiments, we simply assume that for any object, all its instances have equal occurrence probability. All our programs were written in JAVA, and run on a computer with a 2.13GHz Intel CPU and 2GB memory.

Data Generator. To test the scalability of our algorithms, we designed a data generator with parameters (n, m, c), similar to the one used in [1]:

1) For each of the n objects $o_i \in O$ to generate, we first uniformly pick a center $c(o_i)$ in an area of $[0, 1] \times [0, 1]$.
2) Then, a rectangular region $R(o_i)$ centered at $c(o_i)$ is generated where the instances of o_i appear. The length of each edge of $R(o_i)$ is generated from the Guassian distribution with $\mu = c/2$ and $\sigma = c/8$, and if the generated length falls out of $[0, c]$, we repeat its generation until it falls in $[0, c]$.
3) Finally, we generate m_i instances uniformly in $R(o)$, where m_i is picked uniformly from $\{1, 2, \ldots, m\}$.

The expected number of instances for each object is $m/2$, and the expected size $N = \sum_{i=1}^{n} m_i$ of the generated data is $nm/2$. The parameters (n, m, c) of our data generator are summarized as follows: (1) n specifies the number of uncertain objects, (2) m specifies the average number of instances per object, and (3) c specifies how scattered the instances of an object are over the space.

1. We remark that Algorithm 3 in [17] is not correct unless the "if" branch in Line 11 is expanded with an "else" branch: **else** $minVdist_{u_i} = 0$



(a) Preprocessing Time (b) Runtime (c) Pruning Percentage (d) $\mathcal{BL}$ v.s. $\mathcal{CA}$

Fig. 12. Experimental Results with Varying n

To eliminate the bias of each generated dataset, we generate 10 datasets for each parameter configuration (n, m, c) in our experiments, and all the results are reported based on the average of the 10 runs.

8.1 Performance of Exact Algorithms

From now on, we call Algorithm 2 the *Baseline* ($\mathcal{BL}$) algorithm, whose time complexity is $O(m_i \cdot N^2)$. By replacing Line 3 of Algorithm 2 with "Compute $\Pr^{CH}(s_i)$ using the batch evaluation technique", we obtain our algorithm that computes $\Pr^{CH}(o_i)$ in $O(m_i \cdot N \log N)$ time, which we call the *Circular Array* ($\mathcal{CA}$) algorithm.

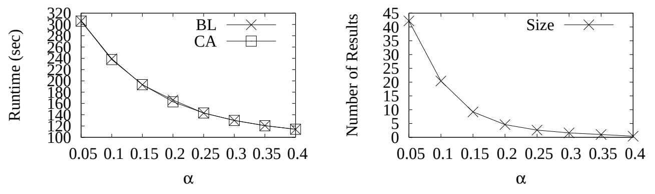
Measures. For each data configuration (n, m, c, α), we evaluate the following measures.

1) Preprocessing time for R-tree (aR-tree) bulk-loading and four-corner pruning & upper bounding.
2) Runtime of both $\mathcal{BL}$ and $\mathcal{CA}$ for evaluating $PCH_\varepsilon(O)$.
3) Percentage of objects pruned.
4) Percentage of instances pruned over the instances of all the non-pruned objects.
5) The runtime ratio of $\mathcal{BL}$ to $\mathcal{CA}$.

Effect of n on scalability. In this set of experiments, we fix $(m, c, \alpha) = (20, 0.2, \varepsilon)$ and study the scalability of $\mathcal{BL}$ and $\mathcal{CA}$ as n increases, the results of which is shown in Figure 12. Figure 12(a) shows the preprocessing time for R-tree (aR-tree) bulk-loading and four-corner pruning & upper bounding, and Figure 12(b) shows the runtime of $\mathcal{BL}$ and $\mathcal{CA}$ for evaluating $PCH_\varepsilon(O)$. From these figures we can see that the preprocessing time is negligible compared with the time of evaluating $PCH_\varepsilon(O)$, which verifies the efficiency of our four-corner pruning & upper bounding techniques.

From Figure 12(b), we see that $\mathcal{BL}$ is much faster than $\mathcal{CA}$ despite the fact that $\mathcal{CA}$ has lower time complexity. This is because our setting $(m, c) = (20, 0.2)$ is favorable to R-tree pruning. Specifically, c is small and thus the instances of an object tends to cluster together. Furthermore, as m is small, the chance of generating a biased sample is small. Therefore, object MBRs are small and R-tree pruning is very effective in this case. On the other hand, when $\mathcal{CA}$ processes a non-pruned instance, it requires a pass over each object instance in the circular array no matter whether it is pruned or not. Therefore, $\mathcal{CA}$ does not fully utilize the pruning power of our R-tree index.

Figures 12(c) shows the pruning effectiveness of our *object-level & instance-level four-corner pruning* techniques, where usually 88%–97% objects and 70%–85% instances are pruned.



(a) Runtime (b) Result Size

Fig. 15. Experimental Results with Varying α

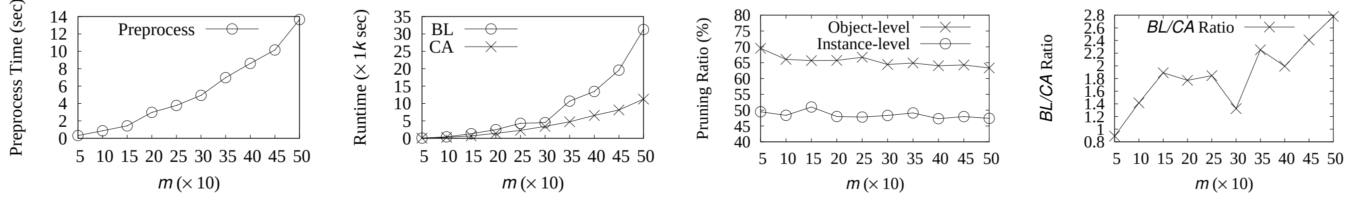
Both object-level and instance-level pruning ratios increase as n increases, since more objects provide more chances for four-corner pruning.

Figures 12(d) shows the runtime ratio of $\mathcal{BL}$ to $\mathcal{CA}$, which decreases as n increases. This indicates that the advantage of $\mathcal{BL}$ over $\mathcal{CA}$ is more prominent for large n. This is because more objects provide more chance for four-corner pruning.

Effect of m on scalability. In this set of experiments, we fix $(n, c, \alpha) = (10^3, 0.2, \varepsilon)$ and study the scalability of $\mathcal{BL}$ and $\mathcal{CA}$ as m increases, the results of which is shown in Figure 13. Since the values of m are much larger now, the runtime is no longer favorable to $\mathcal{BL}$. As Figure 13(b) shows, the runtime of $\mathcal{BL}$ is now longer than that of $\mathcal{CA}$, and the better time complexity of $\mathcal{CA}$ becomes visible. In fact, as Figure 13(d) shows, the runtime ratio of $\mathcal{BL}$ to $\mathcal{CA}$ increases as m increases, which indicates that the advantage of $\mathcal{CA}$ over $\mathcal{BL}$ is more prominent for large m.

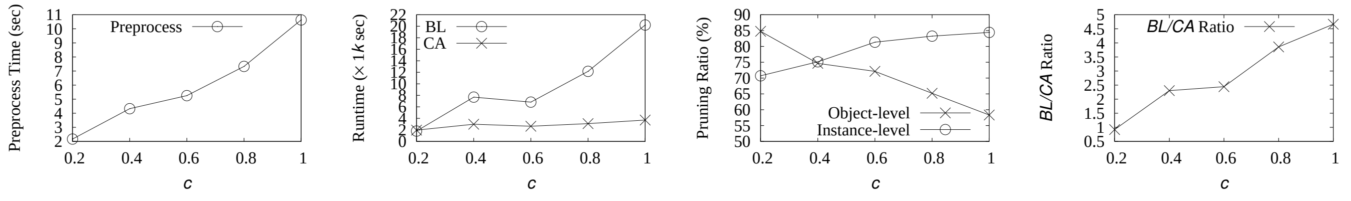
Figures 12(c) shows the pruning effectiveness of our *object-level & instance-level four-corner pruning* techniques, where usually 63%–70% objects and 47%–51% instances are pruned. Both object-level and instance-level pruning ratios decrease as m increases, since more instances per object imply larger object MBRs, which in turn imply less chance for four-corner pruning.

Effect of c on scalability. In this set of experiments, we fix $(n, m, \alpha) = (10^4, 40, \varepsilon)$ and study the scalability of $\mathcal{BL}$ and $\mathcal{CA}$ as c increases, the results of which is shown in Figure 14. Since the values of c are much larger now, the runtime is no longer favorable to $\mathcal{BL}$. As Figure 14(b) shows, the runtime of $\mathcal{BL}$ is now much longer than that of $\mathcal{CA}$, and the better time complexity of $\mathcal{CA}$ becomes quite prominent. In fact, CA does not change too much as c increases, but BL changes sharply. This becomes clearer in Figure 14(d), where the runtime ratio of $\mathcal{BL}$ to $\mathcal{CA}$ increases quickly as C increases. This indicates that the advantage of $\mathcal{CA}$ over $\mathcal{BL}$ is prominent for large c.



(a) Preprocessing Time (b) Runtime (c) Pruning Percentage (d) $\mathcal{BL}$ v.s. $\mathcal{CA}$

Fig. 13. Experimental Results with Varying m



(a) Preprocessing Time (b) Runtime (c) Pruning Percentage (d) $\mathcal{BL}$ v.s. $\mathcal{CA}$

Fig. 14. Experimental Results with Varying c

Figures 14(c) shows the pruning effectiveness of our *object-level & instance-level four-corner pruning* techniques, where usually 58%–85% objects and 70%–85% instances are pruned. Interestingly, while the object-level pruning ratio decreases as c increases due to larger object MBRs, the instance-level one increases. This is because, for larger c, an instance of a non-pruned object has more chance to fall in the central region of the data space, which increases the chance of its pruning. This positive effect outweighs the negative one caused by larger object MBRs.

The datasets in our experiments are already reasonably large. In particular, the largest dataset in the first set of experiments has expected size $nm/2 = 100k \times 20/2 = 1M$. From the figures, we can see that evaluating $PCH_\varepsilon(O)$ on large datasets may take hours. However, these cases involve quite large m and c that are rare in real life applications, and in our *Flickr photo filtering* example, $PCH_\varepsilon(O)$ is computed in less than two seconds. Exact evaluation is acceptable if the data do not change frequently, such as in the Flickr photo filtering application.

Effect of α on performance. We also studied the effect of the threshold parameter α, by fixing $(n, m, c) = (10^4, 20, 0.2)$ and varying α. Figure 15(a) shows the runtime of $\mathcal{BL}$ and $\mathcal{CA}$ for evaluating $PCH_\alpha(O)$, and Figure 15(b) shows the number of objects in the result. Both measures decrease superlinearly as α increases, which demonstrates that there are more low-probability objects than high-probability ones. Besides, the runtime shown in Figure 15(a) is relatively short, which shows the effectiveness of our *four-corner upper bounding* technique and the practicality of $\mathcal{CI}$ for reasonably large threshold values. In fact, we find that most non-pruned objects have very small occurrence probabilities (in the order of 10^{-10} or even 10^{-20}). Besides, Figure 15(a) also shows that $\mathcal{BL}$ and $\mathcal{CA}$ have similar performance for reasonably large threshold values.

8.2 Performance of the Gibbs Sampling Algorithm

We now study the performance of our Gibbs sampling algorithms. We set $\alpha = \varepsilon$ in all the subsequent experiments. We consider two stop conditions for Gibbs sampling. The first one fixes the number of samples to k, and the second one adopts the error estimation approach described at the end of Section 7 using the error threshold $\tau = 5\%$.

We define the following metrics to evaluate the accuracy of Gibbs sampling. Let $\widehat{\Pr^{CH}(o)}$ be the value of $\Pr^{CH}(o)$ estimated by our Gibbs sampling method, the relative error is given by

$$|\widehat{\Pr^{CH}(o)} - \Pr^{CH}(o)| \;/\; \max\{\Pr^{CH}(o), \delta\}. \qquad (21)$$

Note that if $\Pr^{CH}(o) \geq \delta$, Equation (21) is exactly the relative error in traditional sense. Our definition of relative error reduces the influence of objects with very small $\Pr^{CH}(o)$ in error evaluation. For example, consider an object o with $\Pr^{CH}(o) = 10^{-10}$, and suppose that it happens to appear on the convex hull in one sample of totally $k = 10^7$ samples. Then, the traditional relative error is $(10^{-7} - 10^{-10})/10^{-10} \approx 10^3$ which is quite large. However, this is due to the small sample problem (e.g. only one occurrence) and an object with $\Pr^{CH}(o)$ as small as 10^{-10} is usually not interesting. By using Equation (21) and setting $\delta = 0.1\%$, the error contribution of that object is mitigated as $(10^{-7} - 10^{-10})/10^{-3} \approx 10^{-4}$. This definition of relative error is widely used in existing work such as [30].

In all the experiments, we set $\delta = 0.1\%$ (i.e., we are only interested in objects o with $\Pr^{CH}(o) \geq 0.1\%$), and the reported relative error is averaged over all non-pruned objects. We do not incorporate pruned objects since $\Pr^{CH}(o) = 0$ anyway, and thus, no such sample will be obtained. For those objects that have 100% accuracy, incorporating them into the averaged relative error will significantly decrease the error value.

Clearly, the relative error decreases as more samples are considered. To show the trend of error decrement with the

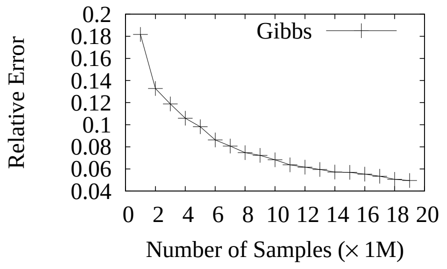


Fig. 16. Number of Samples v.s. Relative Error

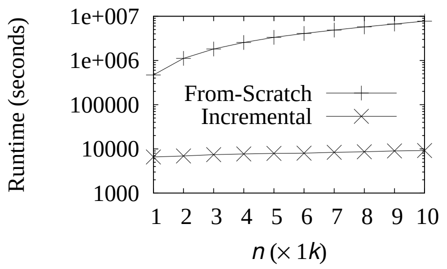
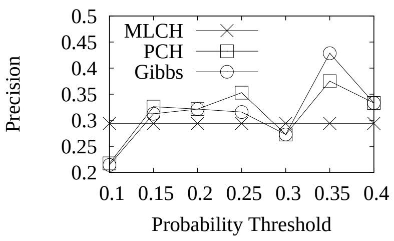


Fig. 17. Comparison of Gibbs Sampling Algorithms

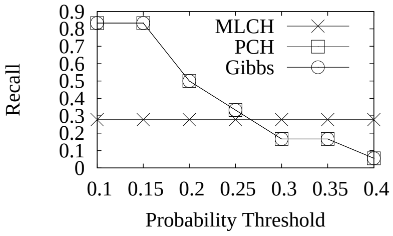
number of samples, we generate a dataset with $(m, c, k) = (20, 0.2, 10^4)$ and compute the average relative error after every 1M samples are obtained. Figure 16 presents the results, where we can see that the relative error decreases quickly as the number of samples increases and it is already below 5% when 19M samples are obtained.

Next, we compare the performance of our sampling approach that maintains the convex hull incrementally, with the naïve approach that computes the convex hull of each sample from scratch. Specifically, we generate datasets with $m = 20$ and $c = 0.2$, and vary n from 10^3 to 10^4. In this set of experiments, we fix the number of samples to 20M, and 10 datasets are generated for each setting of (n, m, c). Figure 17 shows the runtime of both algorithms, which are averaged over the 10 datasets generated. As the figure shows, our algorithm that dynamically maintains the convex hull is consistently 2 to 3 orders of magnitude faster than the naïve approach. In the rest of this subsection, we only consider the algorithm that dynamically maintains the convex hull when referring to our Gibbs sampling algorithm.

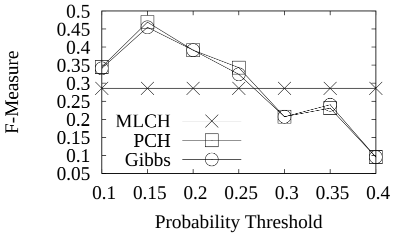
We now evaluate the effectiveness of different definitions of convex hull over uncertain data, i.e. PCH defined by us and MLCH defined in [13] (cf. Section 3). To achieve this goal, we first generate an uncertain dataset with $(n, m, c) = (10^4, 20, 0.2)$, and sample a deterministic object set from it as the ground truth. We also compute the PCH and MLCH of this dataset, and then compute the precision, recall and F-measure of them over the ground truth (i.e. objects on the PCH or MLCH v.s. objecs on the convex hull of the ground truth data), which are shown in Figures 18(a)–(c), respectively. For PCH, we consider $PCH_\alpha(O)$ for different values of α computed from both our exact algorithm and our Gibbs sampling algorithm. On the other hand, MLCH has no concept of α and is thus a constant in the figures. As we can



(a) Precision



(b) Recall



(c) F-Measure

Fig. 18. Accuracy of PCH and MLCH

see from Figure 18(a), PCH generally has a better precision than MLCH. Moreover, as Figure 18(b) shows, PCH achieves a recall much higher than 80% for $\alpha < 0.15$, more than twice that of MLCH. We remark that a high recall is critical in real life applications as more objects of interest are covered by the result. Figure 18(c) shows that PCH achieves a much higher F-measure than MLCH when $\alpha < 0.25$, which further verifies that PCH is more effective than MLCH in terms of both precision and recall.

We also studied the scalability of our Gibbs sampling algorithm with parameters n, m and c, where we fix the number of samples k to 20M. We put the experiments in our online appendix[1] due to the space limitation. Our results show that the algorithm is up to tens of times faster than the exact algorithm, and it achieves small relative error for all objects o with non-negligible $\Pr^{CH}(o)$ (e.g. $\Pr^{CH}(o) > 0.1\%$). Also, when the number of samples is fixed, the relative error does not change much with varying m and c, while the error increases almost linearly with n.

9 CONCLUSION

In this paper, we studied the concept of convex hull over uncertain data, and proposed the *probabilistic convex hull* query. We presented a baseline algorithm with $O(N^3)$ time complexity to answer the query, and developed the *four-corner*

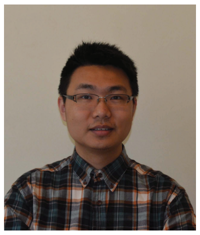
1. http://www.cse.ust.hk/~wilfred/Gibbs/gibbs_appendix.pdf

pruning & upper bounding techniques that prune the majority of the search space. We further improved the time complexity to $O(N^2 \log N)$ using a batch evaluation technique. Experiments show that the baseline algorithm is favorable when n is large and m and c are small, while the batch evaluation technique is more efficient when m and c becomes large. Finally, we presented our Gibbs sampling algorithm which dynamically maintains the convex hull, and demonstrated that it achieves small relative error for all objects o with non-negligible $\Pr^{CH}(o)$ (i.e. $\Pr^{CH}(o) > 0.1\%$). The algorithm is able to answer PCH queries of various settings in just a couple of minutes, which is a reasonable enough response time to support many real-life applications.

References

[1] J. Pei, B. Jiang, X. Lin and Y. Yuan. "Probabilistic Skylines on Uncertain Data". In *VLDB*, 2007.
[2] S. G. Akl and G. T. Toussaint. "Efficient Convex Hull Algorithms for Pattern Recognition Applications". In *Int. Joint Conf. on Pattern Recognition*, 1978.
[3] J. Sander, M. Ester, H.-P. Kriegel and X. Xu. "Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Aplications". *Data Mining and Knowledge Discovery*, Vol. 2, No. 2, 1998.
[4] Y.-C. Chang, L. D. Bergman, V. Castelli, C.-S. Li, M.-L. Lo and J. R. Smith. "The Onion Technique: Indexing for Linear Optimization Queries". In *SIGMOD*, 2000.
[5] S. Borzsony, D. Kossmann and K. Stocker. "The Skyline Operator". In *ICDE*, 2001.
[6] F. P. Preparata and M. I. Shamos. "Computational Geometry: An Introduction". *Springer-Verlag*, 1985.
[7] I. Lazaridis and S. Mehrotra. "Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure". In *SIGMOD*, 2001.
[8] R. Cheng, D. V. Kalashnikov and S. Prabhakar. "Querying Imprecise Data in Moving Object Environments". In *TKDE*, 2004.
[9] R. Cheng, D. V. Kalashnikov and S. Prabhakar. "Evaluating Probabilistic Queries over Imprecise Data". In *SIGMOD*, 2003.
[10] X. Lian and L. Chen. "Probabilistic Group Nearest Neighbor Queries in Uncertain Databases". In *TKDE*, 2008.
[11] X. Lian and L. Chen. "Effcient Processing of Probabilistic Reverse Nearest Neighbor Queries over Uncertain Data". *VLDB Journal*, 2009.
[12] R. Cheng, X. Xie, M. L. Yiu, J. Chen and L. Sun. "UV-Diagram: A Voronoi Diagram for Uncertain Data". In *ICDE*, 2010.
[13] S. Suri, K. Verbeek and H. Yıldız. "On the Most Likely Convex Hull of Uncertain Points". In *ESA*, 2013.
[14] M. Atallah and Y. Qi. "Computing All Skyline Probabilities for Uncertain Data". In *PODS*, 2009.
[15] Z. Zhao, D. Yan and W. Ng. "A Probabilistic Convex Hull Query Tool for Animal Tracking". In *EDBT*, 2012.
[16] C. M. Bishop. "Pattern Recognition and Machine Learning". *Springer New York*, 2006.
[17] B. Yao, F. Li, P. Kumar. "Reverse Furthest Neighbors in Spatial Databases". In *ICDE*, 2009.
[18] N. Dalvi and D. Suciu. "Efficient Query Evaluation on Probabilistic Databases". *VLDB Journal*, vol. 16, no. 4, pp. 523–544, 2007.
[19] P. Agrawal, O. Benjelloun, A. Das Sarma, C. Hayworth, S. Nabar, T. Sugihara and J. Widom. "Trio: A System for Data, Uncertainty, and Lineage". in *VLDB*, 2006.
[20] R. Cheng, D. Kalashnikov and S. Prabhakar. "Evaluating Probabilistic Queries over Imprecise Data". In *SIGMOD*, 2003.
[21] L. Antova, C. Koch and D. Olteanu. "From Complete to Incomplete Information and Back". In *SIGMOD*, 2007.
[22] M. A. Soliman, I. F. Ilyas and K. C.-C. Chang. "Top-k Query Processing in Uncertain Databases". In *ICDE*, 2007.
[23] M. Hua, J. Pei, W. Zhang and X. Lin. "Ranking Queries on Uncertain Data: A Probabilistic Threshold Approach". In *SIGMOD*, 2008.
[24] J. Li, B. Saha and A. Deshpande. "A Unified Approach to Ranking in Probabilistic Databases". In *VLDB*, 2009.
[25] D. Yan and W. Ng. "Robust Ranking of Uncertain Data". In *DASFAA*, 2011.
[26] D. Suciu, D. Olteanu, C. Ré and C. Koch. "Probabilistic Databases". *Synthesis Lectures on Data Management*. Morgan & Claypool Publishers, 2011.
[27] A. Guttman. "R-Trees: A Dynamic Index Structure for Spatial Searching". In *SIGMOD*, 1984.
[28] B. Walsh. "Markov Chain Monte Carlo and Gibbs Sampling". *Lecture Notes for EEB 581*, 2004.
[29] D. Yan, Z. Zhao and W. Ng. "Leveraging Read Rates of Passive RFID Tags for Real-Time Indoor Location Tracking". In *CIKM*, 2012.
[30] X. Xiao, G. Bender, M. Hay and J. Gehrke. "iReduct: Differential Privacy with Reduced Relative Errors". In *SIGMOD*, 2011.
[31] W. K. Ngai, B. Kao, C. K. Chun, R. Cheng, M. Chau and K. Y. Yip. "Efficient Clustering of Uncertain Data". In *ICDM*, 2006.

Da Yan received his B.S. degree in Computer Science from Fudan University, Shanghai, in 2009; and received his Ph.D. degree in Computer Science from the Hong Kong University of Science and Technology. He is currently a post-doctoral fellow in the Department of Computer Science and Engineering, the Chinese University of Hong Kong. His research interests include big data, spatial data management, uncertain data management and data mining.



Zhou Zhao received his B.S. degree in Computer Science from the Hong Kong University of Science and Technology (HKUST), in 2010. He is currently a Ph.D. student in the Department of Computer Science and Engineering, HKUST. His research interests include data cleansing and data mining.

Wilfred Ng received his MS.c. (Distinction) and Ph.D. in Computer Science from the University of London. Currently he is an Associate Professor of Computer Science and Engineering at the Hong Kong University of Science and Technology, where he is a member of the database research group. His research interests are in the areas of databases, data mining and information Systems, which include Web data management and XML searching. Further Information can be found at the following URL: http://www.cs.ust.hk/faculty/wilfred/index.html.

Steven Liu received his B.Eng degree in Computer Science from the Hong Kong University of Science and Technology, in 2012. He is currently a Ph.D. student in the Department of Computer Science, Stony Brook University. His research interests include wireless sensor network and computational topology.