Proceedings of the 24th International Conference on Data Engineering (ICDE),
p. 566-575, April, 2008. This version contains a correction (third page, second column).

Top-k Spatial Joins of Probabilistic Objects

Vebjorn Ljosa [#1], Ambuj K. Singh [*2]

[#]*Broad Institute of MIT and Harvard*
7 Cambridge Center, Cambridge, MA 02142, U.S.A.
[1]ljosa@broad.mit.edu

[*]*Dept. of Computer Science, University of California*
Santa Barbara, CA 93106-5110, U.S.A.
[2]ambuj@cs.ucsb.edu

***Abstract*— Probabilistic data have recently become popular in applications such as scientific and geospatial databases. For images and other spatial datasets, probabilistic values can capture the uncertainty in extent and class of the objects in the images. Relating one such dataset to another by spatial joins is an important operation for data management systems.**

We consider probabilistic spatial join (PSJ) queries, which rank the results according to a score that incorporates both the uncertainties associated with the objects and the distances between them. We present algorithms for two kinds of PSJ queries: Threshold PSJ queries, which return all pairs that score above a given threshold, and top-k PSJ queries, which return the k top-scoring pairs.

For threshold PSJ queries, we propose a plane sweep algorithm that, because it exploits the special structure of the problem, runs in $O(n \log n + k)$ time, where n is the number of points and k is the number of results. We extend the algorithms to 2-D data and to top-k PSJ queries. To further speed up top-k PSJ queries, we develop a scheduling technique that estimates the scores at the level of blocks, then hands the blocks to the plane sweep algorithm. By finding high-scoring pairs early, the scheduling allows a large portion of the datasets to be pruned. Experiments demonstrate speed-ups of two orders of magnitude.

I. Introduction

In its basic form, a spatial join is a query which, given a set of red points and a set of blue points, finds all pairs consisting of one red and one blue point within a certain distance of each other. Such queries have been studied extensively, and many efficient techniques exist for answering them [1], [2], [3], [4], [5], [6]. A new look at spatial joins is mandated, however, by the large geographical and biomedical image datasets that are becoming available.

The interesting feature of these datasets is not that they contain many objects, but that the objects are *probabilistic* in nature. For instance, satellite imagery is being annotated both by automated systems and by unreliable humans: Machine learning applications recognize airports, golf courses, residential areas, different crops, and other land uses, and community-based systems like Wikimapia allow anyone to annotate regions. Both lead to large numbers of annotations, but of varying quality: Some objects are accurately annotated, whereas many others are not. The accuracy of an annotation is approximated by the segmentation algorithm and/or classifier (for an automated system) or from a reputation network (for a community-based system).

As a second example of probabilistic spatial datasets, consider annotations of biomedical images. Microscopy is the main source of data for many fields of biology. New, high-throughput microscopes are producing images much faster than before; in addition, emerging databases of micrographs [7], [8], [9] make it possible for the first time to analyze large collections of existing images. The size of the datasets rules out the traditional approach of identifying objects and relationships manually. Instead, we must rely on automatic techniques. It is often impossible to interpret the image unequivocally, but analysis techniques can manage the uncertainty by producing *probabilistic values* for the extent and class of the objects identified. This gives subsequent analysis steps more information to work with, and thus a better chance of gaining new biological insight [10].

How do probabilistic datasets change the nature of spatial joins? First of all, not all result pairs are equally desirable: A pair consisting of a high-confidence red point and a high-confidence blue point is more important than a pair of low-confidence points. Add to this that the datasets generally have many low-confidence points and few high-confidence points, and it becomes clear that the spatial join is a form of top-k query: Given two sets of probabilistic points, find the k top-scoring pairs according to a ranking function that takes into account the confidence values of the points (and perhaps also the distance between them).

In the examples above, the probabilistic objects are points with deterministic position; what is probabilistic is their class. For example, the satellite imagery classifier could say that it is 80 % certain that the object at 34°26′N 119°50′W is an airport, but leave no doubt that there is *something* at exactly that location. Representing an object as a point is acceptable when the uncertainty in position and extent of the object is small compared to the distances between objects. However, for many real datasets this is not the case, so the extent of the object is also probabilistic—perhaps derived from a probabilistic segmentation [10]. As an example, the uncertain extent of a horizontal cell is shown in Figure 1. To be widely useful, a spatial join method for probabilistic datasets should handle uncertainty both in class and extent.

This paper studies spatial joins on probabilistic datasets. Spatial joins are a natural next step in the progression of database and data mining techniques that have been devel-

Fig. 1. Probabilistic mask representing uncertainty in a horizontal cell's extent. Darker pixels have higher probability of belonging to the cell. A second source of uncertainty is the class of the cell: A classifier could decide, for instance, that there is a 70 % probability of this cell being a B-type horizontal cell.

oped over the last few years: Probabilistic network analysis [11], range and k-NN queries [12], [13], [14], [15], top-k queries [16], equality queries [17], segmentation [10], and clustering [18] techniques have started to appear, motivated by the desire to find patterns that have so far eluded discovery in protein interaction, moving object, sensor, and image data.

This paper makes the following major contributions:

- *Query definitions.* We define threshold spatial joins and top-k spatial joins of objects with probabilistic extent.
- *Spatial join algorithms for probabilistic points.* By exploiting the geometry of the problem and the score function, we can escape the curse of $O(n^2)$-time join and compute a threshold spatial join of 1-D or 2-D datasets in $O(n(\log n + k))$ time, where n and k are the number of points and results, respectively.
- *Predictive scheduling.* Although the geometry-based join algorithm extends to top-k probabilistic spatial join, it is beneficial to find good matches early so most of the remaining data can be pruned. Our index-based scheduling algorithm predicts high-scoring pairs of pages, then calls the geometry-based algorithm to join them.

The rest of the paper is organized as follows. Section II defines the problem and query types. Section III presents our geometry-based algorithm for threshold spatial joins of probabilistic datasets. Section IV extends the algorithm to top-k probabilistic spatial joins and develops the index-based scheduling technique. Section V evaluates the algorithms experimentally. Section VI discusses related work before Section VII concludes the paper.

II. Problem Statement

What kind of spatial joins must a database support in order to be able to help in biomedical and geographical applications?

First, objects often cannot be segmented reliably. We must therefore work with probabilistic values of the objects' extents, i.e., probabilistic masks [10] (see Figure 1 for an example).

Definition 1 (from [10]): A **probabilistic mask** M_a for an object a is a set of tuples $\{\langle \vec{x}, p\rangle\}$ such that each point $\vec{x}$ belongs to a with probability p.

Second, even after segmentation, the objects cannot be identified reliably. Our technique must therefore work with *confidence values*. A confidence value is the classifier's estimate of the probability that the object belongs to the class.

Definition 2: A **probabilistic object** a is a pair $\langle M_a, p_a\rangle$, where M_a is a's probabilistic mask and p_a is the confidence value of a's class.

Third, it is not clear how close two objects must be in order to constitute a pair of interest. Rather than only including pairs within a certain distance, we should therefore incorporate the distance in our ranking function so that two objects that are closer together are ranked higher (other things equal). We adopt a similarity measure that decays exponentially with increased distance. This mapping is appealing because of its simplicity and because it is sensitive to small variations in small distances. It has been validated for applications in cognitive science, biogeography, and ecology [19], [20].

We can now derive an appropriate score function for ranking pairs of objects. First, consider two point objects a and b with exactly known location, but with probabilities p_a and p_b of belonging to their respective classes. Their probabilistic masks have only one point. Because it is the only point, its probability of belonging to the object must be one. In other words, $M_a = \langle \vec{x}_a, 1\rangle$, and similarly for b. It is now clear that the score function for a pair of point objects $\langle a, b\rangle$ with known location should be a monotonely increasing function of their confidence values (p_a and p_b) and the inverse exponential of their distance. Because these are all non-negative, we choose their product as our score function.

Definition 3: The **score** s' between two point objects $a = \langle\langle \vec{x}_a, 1\rangle, p_a\rangle$ and $b = \langle\langle \vec{x}_b, 1\rangle, p_b\rangle$ is

$$s'(p_a, \vec{x}_a, p_b, \vec{x}_b) = p_a\, p_b\, \lambda \mathrm{e}^{-\lambda d(\vec{x}_a, \vec{x}_b)}, \tag{1}$$

where λ is a positive, domain-specific parameter that determines the relative importance of probability and distance, and d is a suitable distance function.

The fourth and last requirement concerns the score between objects that are not single points: The distance that is important is the *smallest* distance between two pixels that with high probability belong to objects a and b, respectively. It is not important that there are many other high-probability pixels that belong to the two objects, but are farther apart. The simplest way to satisfy this requirement is to define the score between two objects as the maximum of all scores between constituent points, weighted by the probability that both points belong to their respective objects:

Definition 4: The **score** $s(a, b)$ of a pair of objects $a =$

$\langle M_a, p_a \rangle$ and $b = \langle M_b, p_b \rangle$ is

$$s(a,b) = \max_{\substack{\langle \vec{x}_a, q_a \rangle \in M_a \\ \langle \vec{x}_b, q_b \rangle \in M_b}} p_a\, p_b\, s'(q_a, \vec{x}_a, q_b, \vec{x}_b) \qquad (2)$$
$$= p_a\, p_b \max_{\substack{\langle \vec{x}_a, q_a \rangle \in M_a \\ \langle \vec{x}_b, q_b \rangle \in M_b}} q_a\, q_b\, \lambda e^{-\lambda d(\vec{x}_a, \vec{x}_b)}.$$

This object-level score function—or variations, such as the 90th percentile of the point-level scores between the two objects—can easily be used in concert with any technique designed for point-level scores (Definition 3). To simplify the presentation, the algorithms in this paper are therefore presented in terms of the point-level score function.

We can now proceed to define two kinds of useful PSJ queries.

Definition 5: Given two sets A and B of probabilistic objects and a score threshold τ, a **threshold probabilistic spatial join query** ("threshold PSJ query") finds all pairs $\langle a, b \rangle \in A \times B$ such that $s(a,b) \geq \tau$.

Definition 6: Given two object sets A and B and a natural number k, a **top-k probabilistic spatial join query** ("top-k PSJ query") finds a set $R \subseteq A \times B$ of size k such that other pairs in $A \times B$ score no higher than the lowest-scoring pair in R.

These general query types have many applications. Neuroscientists want to know, for instance, whether the axons of bipolar cells in the retina extend in order to maintain the connection to the photoreceptor synaptic terminals [21], whether Müller cells hypertrophy toward macrophages [22], and whether proliferating horizontal cell dendrites grow toward subretinal glial scars [23]. All of these questions translate naturally into spatial joins.

III. Threshold Probabilistic Spatial Join

Now that we have defined threshold PSJ queries, we can discuss algorithms for answering them. The basic solution is a nested loop join, which computes the scores between every object in A and every object in B and checks whether it exceeds τ. If A and B contain n and m points, respectively, this takes $O(nm)$ time, which precludes datasets of even moderate size. Our algorithm exploits the geometry of the score function to find the solution more efficiently.

We first present our algorithm for 1-D data (i.e., probabilistic points on a line). Then we extend it to 2-D data.

A. Threshold Probabilistic Spatial Join in 1D

For a point $a = \langle x_a, p_a \rangle \in A$, what can we say about where a corresponding point $b = \langle x_b, p_b \rangle$ must be in order to yield a score of at least τ? From the score function, we have that

$$p_a\, p_b\, \lambda e^{-\lambda |x_a - x_b|} \geq \tau \qquad (3)$$

and, consequently, that

$$|x_a - x_b| \leq \frac{1}{\lambda} (\ln p_a + \ln p_b + \ln \lambda - \ln \tau). \qquad (4)$$

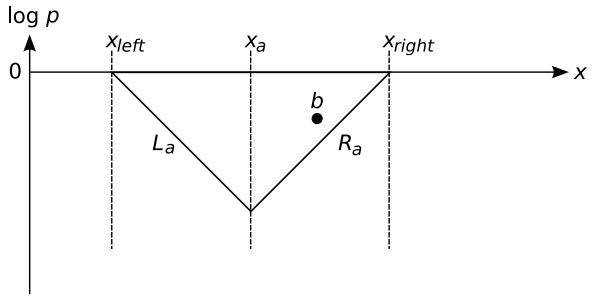

Fig. 2. Each point $a = \langle x_a, p_a \rangle$ defines a triangle. The score between two points a and $b = \langle x_b, p_b \rangle$ exceeds the threshold τ only if b is inside a's triangle. See Ineq. (4).

In a plot with x on the first axis and $\ln p$ on the second, Ineq. (4) describes an inverted triangle. The left and right side are $2(\ln p_a + \ln \lambda - \ln \tau)/\lambda$ apart at $\ln p = 0$ (which corresponds to $p = 1$, the highest probability any point can have). They eventually meet at $x = x_a$ and $\ln p = \ln \tau - \ln p_a - \ln \lambda$. Figure 2 shows an example of such a triangle.

Our algorithm is a plane sweep algorithm that slides (conceptually) a vertical line from $x = -\infty$ to $x = \infty$. It maintains a data structure of all the triangles currently intersecting the sweep line. When the sweep line meets a point $\langle x_b, \ln p_b \rangle \in B$, we search the data structure for all triangles in which the point is contained. Graphically, this can be done by following the $x = x_b$ line from $\langle x_b, \ln p_b \rangle$ toward $\langle x_b, -\infty \rangle$; for any line segment we intersect on the way, we know that the corresponding a scores high enough together with b, so the pair $\langle a, b \rangle$ can be added to the result set.

The time complexity of this search operation is critical. In the worst case, there are n triangles in the data structure at the same time, and we perform one search for each point in B, so the search operation must be sublinear in order to avoid $O(nm)$ complexity. Insertion and deletion must also be sublinear in order to avoid quadratic complexity in n.

For arbitrary triangles, this seems to be a tall order. Bentley and Ottmann's algorithm [24] comes close by maintaining a balanced binary tree of the segments that are currently intersecting the sweep line. In order to keep the tree sorted, however, two neighboring segments in the tree must be swapped when they intersect—and in the worst case, every triangle can intersect with every other. That is no problem for Bentley and Ottmann's purpose, which is precisely to find line segment intersections, but for our application, $O(n^2)$ time complexity from intersection processing is a show stopper.

The key to a better solution is the slope of the line segments that make up our triangles. From Ineq. (4), we see that the left and right edges of the triangle (which we call L_a and R_a, respectively) are the lines defined by

$$L_a: \quad \ln p = -\lambda x + (\ln \tau - \ln p_a - \ln \lambda + \lambda x_a) \qquad (5)$$

and

$$R_a: \quad \ln p = \lambda x + (\ln \tau - \ln p_a - \ln \lambda - \lambda x_a), \qquad (6)$$

respectively. The slopes of these lines are always $-\lambda$ and λ. They do not depend on p_a or x_a, only on the constant λ.

This insight does not help Bentley and Ottmann's algorithm, which can still encounter $O(nm)$ intersections. But what if we keep *two* trees, one for downsloping segments and one for upsloping segments? Then the segments in each tree are parallel, and there is never a need to reorder segments.

Our join algorithm for 1-D points incorporates this idea, and is shown in Figure 3. The algorithm starts by initializing the event queue: A *Point* event is added for each point in B, and for most points in A, we add a *Start*, *Mid*, and *End* event. A point a in A is skipped, however, if

$$\frac{1}{\lambda}(\ln p_a + \ln\lambda - \ln\tau) < 0, \tag{7}$$

for in that case, p_a is so low that no point from b can possibly yield as score of τ or more. The x-coordinates of the *Start* and *End* events are computed by subtracting and adding, respectively, the right side of Ineq. (4) from x_a. The x-coordinate of the *Mid* event is simply x_a.

In the main event loop, the algorithm responds to *Start*, *Mid*, and *End* events by inserting a segment into and/or removing a segment from the two trees. At any time, all downsloping segments intersected by the sweep line are present in $T_{-\lambda}$, and all upsloping segments intersected by the sweep line are present in T_λ. The key for each entry in the tree is the x-coordinate of the segment's intersection point with the $\ln p = 0$ axis. In addition, each entry contains the identifier of the point in A it corresponds to.

When the algorithm encounters a *Point* event for a point b, it first computes the two lines (slope $\pm\lambda$) that intersects b. It then finds their intersections x_{left} and x_{right} with the $\ln p = 0$ axis. Finally, it searches the two trees. In $T_{-\lambda}$ (the tree of downsloping segments), it finds all entries with keys not exceeding x_{left}. The points they refer to are paired with b and reported as results. Similarly, T_λ is searched for all entries with keys of x_{right} or greater.

The algorithm uses $O(n+m)$ space, but what is its time complexity? Sorting the events is $O(n\log n)$ and there are $O(n+m)$ events to process. Processing an event is either $O(\log n)$ (for *Start*, *Mid*, and *End* events) or $O(\log n + k)$ (for *Point* events). (Here, k is the number of results—for threshold PSJ queries an unavoidable cost.) In summary, the time complexity of our algorithm is $O((n+m)(\log n+k))$—or, if $n = m$, $O(n(\log n+k))$.

The following theorems prove that the algorithm finds a pair if and only if its score is at least the threshold.

Theorem 1 (Completeness): If $a \in A$, $b \in B$, and $s(a,b) \geq \tau$, then $\langle a,b\rangle$ is in the result of the 1-D plane sweep algorithm.

Proof: Suppose $s \geq \tau$ and assume (without loss of generality because the other case is symmetric) that $x_b \geq x_a$. It follows from Ineq. (4) that

$$\ln p_b \geq \lambda x_b + (\ln\tau - \ln p_a - \ln\lambda - \lambda x_a), \tag{8}$$

which means (see Eq. (6)) that b is above R_a, the line that is the right side of a's triangle. Because p_b is a probability, it cannot exceed 1, so $x_b \leq x_{\text{right}}$, where x_{right} is where R_a intersects the $\ln p = 0$ line. Because $x_a \leq x_b \leq x_{\text{right}}$, we know that at the time the plane sweep algorithm receives b's *Point* event, it will have already processed a's *Mid* event and not yet seen a's *End* event, so R_b is in T_λ. Consequently, a is in T_λ (the tree of upsloping line segments) and in the result of the query T_λ.range(x_right, ∞), and $\langle a,b\rangle$ is returned as a result. ■

Theorem 2 (Soundness): If $\langle a,b\rangle$ is in the result of the 1-D plane sweep algorithm, then $a \in A$, $b \in B$, and $s(a,b) \geq \tau$.

Proof: If the algorithm returns $\langle a,b\rangle$, then a must be returned by one of the queries $T_{-\lambda}$.range(-∞,x_left) and T_λ.range(x_right, ∞). Assume that it is in the second query (the other case is symmetric). Then $x_a \leq x_b \leq x_{\text{end}}$ and Ineq. (8) holds. It follows by Eq. (4) that $s \geq \tau$. ■

B. Threshold Probabilistic Spatial Join in 2D

The plane sweep algorithm can be adapted to probabilistic points in the plane if the distance metric is L_1. We write out $\vec{x}_i$ as $\langle x_i, y_i\rangle$. The score function then becomes

$$s = p_a\, p_b\, \lambda e^{-\lambda(|x_a-x_b|+|y_a-y_b|)}, \tag{9}$$

so

$$\ln s = \ln p_a + \ln p_b + \ln\lambda - \lambda|x_a-x_b| - \lambda|y_a-y_b|. \tag{10}$$

In $(x,y,\ln p)$-space, we have a pyramid instead of a triangle for each point a in A. The base of the pyramid (a square) is in the xy-plane, and the sides of the base make 45° angles with the x and y axes. The four triangular sides meet at a point with x and y coordinates equal to those of a, and with

$$\ln p = \ln\tau - \ln p_a - \ln\lambda. \tag{11}$$

We refer to the four sides as quadrants Q_1–Q_4, starting northeast of $\langle x_a, y_a\rangle$ and going counterclockwise. The sides are in planes described by the following four equations:

$$Q_1 : \ln p = \ln\tau - \ln p_a - \ln\lambda + \lambda(x-x_a) + \lambda(y-y_a) \tag{12}$$
$$Q_2 : \ln p = \ln\tau - \ln p_a - \ln\lambda - \lambda(x-x_a) + \lambda(y-y_a) \tag{13}$$
$$Q_3 : \ln p = \ln\tau - \ln p_a - \ln\lambda - \lambda(x-x_a) - \lambda(y-y_a) \tag{14}$$
$$Q_4 : \ln p = \ln\tau - \ln p_a - \ln\lambda + \lambda(x-x_a) - \lambda(y-y_a) \tag{15}$$

Because all Q_1 planes are parallel, each can be represented by a single number $G_1(x_a,y_a,p_a)$, and they can be kept in sorted order. (Likewise for Q_2–Q_4.) As G_i, we choose the intersection point between the plane and the $(\log p)$-axis, which can be computed by setting $x_i = y_i = 0$ in Eq. (12). For Q_1 (the other quadrants are similar):

$$G_1(x_a,y_a,p_b) = \ln\tau - \ln p_a - \ln\lambda + \lambda(-x_a) + \lambda(-y_a) \tag{16}$$

Two points $a \in A$ and $b \in B$ yield a score of at least the threshold τ if and only if b is contained in a's pyramid. We can find all pyramids that contain b by starting at b and following the $x = x_b \wedge y = y_b$ line away from the xy-plane, reporting all pyramid sides we encounter. This is equivalent to computing the intersection point $H_i(x_b,y_b,p_b)$ between the $(\ln p)$-axis and

```
Algorithm THRESHOLD-PSJ(A, B, λ, τ)
q ← make-event-queue(A, B, λ, τ)
while q.size > 0:
  e ← q.pop()
  if e.type = Point:
    b ← e.point
    x_left ← b.x + ln b.p /λ
    if x_left ≤ b.x:
      x_right ← b.x − ln b.p /λ
      for (x, id) in T_{−λ}.range(−∞, x_left):
        report result (id, b.id)
      for (x, id) in T_λ.range(x_right, ∞):
        report result (id, b.id)
  else:
    a ← e.point
    d ← (1/λ) * (ln a.p + ln λ − ln τ)
    x_left ← a.x − d
    x_right ← a.x + d
    case type(e):
      Start:
        T_{−λ}.insert(x_left, a.id)
      Mid:
        T_{−λ}.delete(x_left, a.id)
        T_λ.insert(x_right, a.id)
      End:
        T_λ.delete(x_right, a.id)

Function MAKE-EVENT-QUEUE(A, B, λ, τ)
q.events ← array of |A| + |B| events
q.size ← |A|
j ← 0
for a in A:
  d ← (1/λ)(ln a.p + ln λ − ln τ)
  if d < 0: /* τ too high for a match to be possible */
    q.size ← q.size −1
  else:
    q.events[j] ← Event(Start, a, a.x − d)
    q.events[j + 1] ← Event(Mid, a, a.x)
    q.events[j + 2] ← Event(End, a, a.x + d)
    j ← j + 3
for b in B:
  q.events[j] ← Event(Point, b, x_b)
q.events ← q.events[0..(n + |B| − 1)]
sort q.events by x
q.next ← 0
return q

Function POP(q)
q.next ← q.next +1
q.size ← q.size −1
return q.events[q.next −1]
```

Fig. 3. Our join algorithm for 1-D probabilistic points avoids quadratic time complexity by maintaining two trees, one for each possible slope.

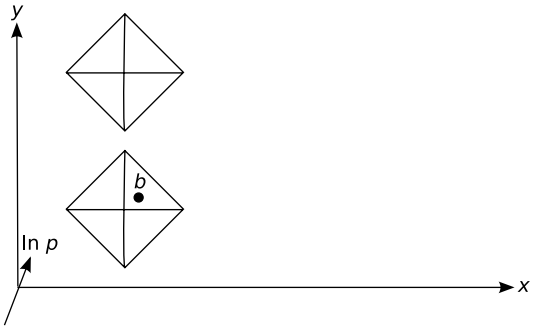

Fig. 4. The point is inside the lower pyramid, but a direct adaptation of the 1-D plane sweep algorithm finds the Q_1 plane of the upper pyramid as well.

the plane that goes through b and is parallel with G_i, then finding every Q_i-plane the G_i of which is no more than H_i. For quadrant Q_1, the corresponding function H_1 is

$$H_1(x_b, y_b, p_b) = \ln p - \lambda x_b - \lambda y_b. \tag{17}$$

The challenge is how to find the correct pyramid sides. In 1D, this was not an issue because the plane sweep ensured that only the relevant segments were in the trees. In 2D, however, planes at some other y position can cause false positives. As an example, Figure 4 shows two pyramids and a point b. The pyramids are identical but translated in the y-direction. The point is inside the lower pyramid, directly under its Q_1 face. What happens in the example if we directly adapt the algorithm developed in the previous section? Suppose we keep a balanced binary tree (e.g., a red-black tree) for each quadrant and search the tree of Q_1-planes for $G_1(x_a, y_a, p_a) \leq H_1(x_b, y_b, p_b)$. We find the lower pyramid, which is correct, but also the upper pyramid, which is not.

The essence of the problem is that the second plane we find is a superset of the Q_1-face of the upper pyramid. The plane is only valid in the first quadrant of the pyramid (i.e., where $x > x_a$ and $y > y_a$), but we find that plane for a point that is elsewhere. False positives such as these can be avoided by checking, every time we find a plane, whether our point is in the proper quadrant of the pyramid the plane came from. Such a check can lead to performance problems, however: In the worst case, there could be n pyramids in the trees, and for each point we could end up checking all of them, resulting in $O(nm)$ time complexity.

A better solution is to replace the red-black tree with a data structure that allows us to incorporate the quadrant check in the search predicate. For each plane we store three numbers: its intersection point $G_i(x_a, y_a, p_a)$ with the $(\log p)$-axis, and the x and y-coordinates of its corresponding point from A. When we encounter a point $\langle x_b, \log p_b \rangle$ from B, we perform a range query on the data structure that contains Q_1 planes for all planes that satisfy the following predicate (the other quadrants are analogous):

$$G_1(x_a, y_a, p_b) \leq H_1(x_b, y_b, p_b) \wedge y_a \geq y_b \wedge x_a \geq x_b. \tag{18}$$

This approach can be extended beyond two dimensions.

Like other plane sweep algorithms, it is unlikely to perform well on very high-dimensional datasets.

Because we need insert, delete, and search to be sublinear in n, we adopt the skip quadtree [25], which performs insert and delete in $O(\log n)$ time and search in $O(\log n + k)$ time for data of any dimensionality. It follows that the time complexity for the 2-D plane sweep is the same as that of 1-D plane sweep, i.e., $O((n+m)(\log n + k))$.

The following theorems proves that the 2-D plane sweep algorithm finds a pair if and only if its score is at least the threshold.

Theorem 3 (Completeness): If $a \in A$, $b \in B$, and $s(a,b) \geq \tau$, then $\langle a,b \rangle$ is in the result of the 2-D plane sweep algorithm.

Proof: Suppose $s \geq \tau$ and assume (without loss of generality because the other three cases are analogous) that $x_b \geq x_a$ and $y_b \geq x_a$. By Eq. (10),

$$\ln p_a + \ln p_b + \ln \lambda - \lambda x_b + \lambda x_a \geq \ln \tau, \tag{19}$$

e so $G_1(x_a, y_a, p_a) \leq H_1(x_b, y_b, p_b)$, and Ineq. (18) is satisfied. Because a's Q_1-plane is in the tree when the sweep line reaches $x = x_b$ (same reasoning as the 1-D case), $\langle a,b \rangle$ is included in the result. ■

Theorem 4 (Soundness): If $\langle a,b \rangle$ is in the result of the 2-D plane sweep algorithm, then $a \in A$, $b \in B$, and $s(a,b) \geq \tau$.

Proof: If the algorithm returns $\langle a,b \rangle$, then a must be returned by one of the four queries. Assume that it is returned by the query for the first quadrant (see Ineq. 18). By Eqs. (10), (16), and (17), $s(a,b) \geq \tau$. ■

IV. Top-k Probabilistic Spatial Joins

In Section III, we have discussed techniques for answering threshold spatial joins on probabilistic datasets. In this section, we consider top-k spatial join queries, which are more useful than their threshold counterparts, but more difficult to answer.

The basis for top-k spatial joins is that each pair in the join result has a score. The score depends on the probabilities of the two points as well as the distance between them (see Eq. (1)). The pairs can be ranked by their scores, and in most cases only the highest-scoring pairs are of interest. If we know an appropriate threshold value, we can use the algorithm for threshold PSJ, but in general this is not the case.

A. Plane sweep algorithm for top-k

Our plane sweep algorithm for threshold queries can be adapted to answer top-k queries. The threshold is initialized to a small positive value (e.g., 10^{-6}), which remains unchanged until k results have been found. Then, after each new result, the threshold is updated to the score of the k-th best result found so far. Note that the threshold is monotonically nondecreasing.

The effect of a change in threshold is to shrink all triangles. From Eqs. (5) and (6), we have that

$$x_{\text{left}} = x_a + \frac{1}{\lambda}(\ln \tau - \ln p_a - \ln \lambda) \tag{20}$$

and

$$x_{\text{right}} = x_a - \frac{1}{\lambda}(\ln \tau - \ln p_a - \ln \lambda), \tag{21}$$

so when τ increases by $\Delta\tau$, x_{left} and x_{right} move right and left, respectively, by

$$\frac{1}{\lambda} \ln \frac{\tau + \Delta\tau}{\tau}.$$

The bottom of the triangle, at $x = x_a$ and

$$\ln p = \ln \tau - \ln p_a - \ln \lambda, \tag{22}$$

moves upward by $\ln[(\tau + \Delta\tau)/\tau]$.

Adjusting the *Start* and *End* events in the queue to account for this shrinkage is a complex affair because one must account for which line segments are already in the index structure. Notice, however, that the shrinking of the triangles is equivalent to shifting them upward by $\ln[(\tau + \Delta\tau)/\tau]$—or, equivalently, by shifting the points in B downward by the same amount! Therefore, when the threshold increases, we simply increase an offset variable, the value of which is subtracted from the $(\ln p)$-dimension whenever a *Point* event is processed.

When the threshold increases, the remaining number of candidate pairs generally decreases because a pair may score above the old threshold but below the new threshold.

This adapted plane sweep algorithm works, but its performance depends on the order in which results are found. If some high-scoring pairs happen to be found early, the threshold will increase beyond the scores of most pairs, and so they will be pruned by the plane sweep. Conversely, if the good pairs are not found until late in the search, we end up considering many pairs that will not be in the final result.

B. Global scheduling algorithm

We have developed a global scheduling technique for finding good pairs early. We start by partitioning each point set into a number of subsets so that there are approximately r (a parameter) points in each subset and the points in a subset have similar spatial coordinates and probabilities. We write A_i for the i-th subset of A and $\mathcal{A}$ for the set of all subsets of A. We then make global scheduling decisions based on the minimum bounding rectangles (MBRs) of these subsets.

The scheduling algorithm works incrementally, by repeatedly deciding which pair of MBRs to join next. The next pair of MBRs to join should be the one that has the highest likelihood of yielding at least one pair of points that scores above the current threshold. We should aim to find the two MBRs $A_i \in \mathcal{A}$ and $B_i \in \mathcal{B}$ such that

$$s_{\max}(A_i, B_i) = \max_{a \in A_i} \max_{b \in B_i} p_a p_b \lambda e^{-d(\vec{x}_a, \vec{x}_b)} \tag{23}$$

is maximized. The score distribution for a pair of MBRs is difficult to compute, however, for several reasons. First, we do not know how the points are distributed in space or in probability. Second, even if we assume that the points are distributed uniformly within the MBRs, we cannot find a closed-form solution, but must resort to sampling the MBRs, which is too time consuming.

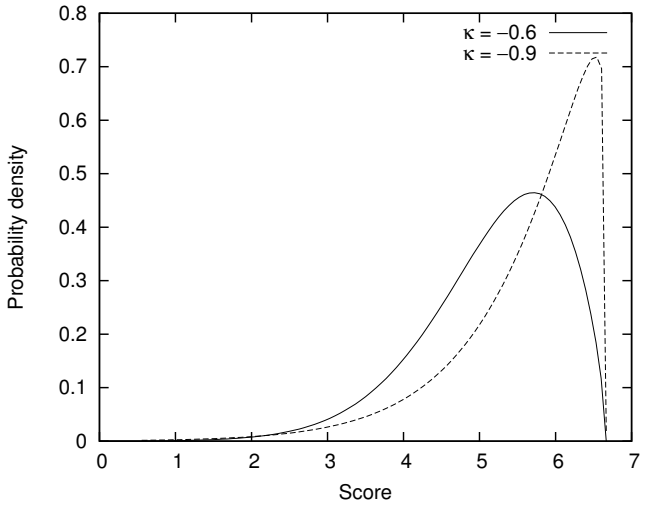

Fig. 5. The effect of increasing the number of points in each MBR is to increase the magnitude of the (negative) shape parameter κ of the extreme value distribution that approximates the maximum score, s_{max}. When there are many points, the distribution becomes extremely left-skewed, so the upper bound of the score between two MBRs is a good approximation of s_{max}.

The fact that a pair of MBRs represents a very large number of point pairs comes to our rescue, however. Because $s_{\max}$ is the maximum of many terms, it can be approximated well by a generalized extreme value distribution (EVD) with pdf of the form (see [26, p. 64])

$$f(s;\kappa,\mu,\sigma) = \frac{1}{\sigma} e^{-\left(1+\kappa\frac{s-\mu}{\sigma}\right)^{-1/\kappa}} \left(1+\kappa\frac{s-\mu}{\sigma}\right)^{-1-1/\kappa}, \quad (24)$$

where μ is the mean, σ is the standard deviation, and κ is a shape parameter. Because the points are constrained by the MBR boundaries, their score distribution also has finite domain. Consequently, the EVD is a Weibull-type (or type III) EVD, and the shape parameter κ in Eq. (24) is negative. The cdf is then (see [26])

$$F(s;\kappa,\mu,\sigma) = e^{-\left(1+\kappa\frac{s-\mu}{\sigma}\right)^{-1/\kappa}}. \quad (25)$$

When the number of points in each MBR (and thus the number of terms to maximize over) increases, the shape parameter κ decreases, the distribution becomes more left-skewed, as shown in Figure 5. The upper bound s_{ub} for the scores between two boxes is therefore a good approximation of $s_{\max}$ when there are many points in each box. This increasingly becomes the case as the datasets grow—which is precisely when we need our scheduling to be effective.

The upper bound s_{ub} can be found by plugging the maximum probabilities of the two MBRs and the minimum distance between them into Eq. (1). In addition to scheduling, the upper bound is also used for pruning: If the upper bound for a pair of MBRs is below τ, we can prune the pair without even looking at the points they contain.

In conclusion, we order our joins by s_{ub}. The MBRs are constructed by repeatedly splitting the dimension with the highest variance (as when constructing a VAMsplit R-tree [27]). We compute s_{ub} for all pairs of MBRs from the two datasets. These values are then sorted, and we start to join MBR pairs in decreasing order of their upper bounds. The algorithm, which is shown in Figure 6, terminates when the next s_{ub} is below the current threshold. The scheduling algorithm terminates when all remaining pairs of MBRs have been processed or pruned.

Algorithm TOPK-PSJ-SCHEDULING(A, B, λ, r)
$\tau \leftarrow 10^{-6}$
$\mathcal{A} \leftarrow$ VAMSPLIT(A, R)
$\mathcal{B} \leftarrow$ VAMSPLIT(B, R)
for i from 0 below $|\mathcal{A}|$:
  for j from 0 below $|\mathcal{B}|$:
    $d_{\text{lb}} \leftarrow$ minimum distance between MBRs A_i and B_j
    $s_{\text{ub}} \leftarrow A_i.\text{max\_p} \times B_j.\text{max\_p} \times \lambda \times e^{-\lambda d_{\text{lb}}}$
    estimate$[i|\mathcal{B}|+j] \leftarrow \langle i, j, s_{\text{ub}}\rangle$
sort "estimate" array by s_{ub}
for $\langle i, j, s_{\text{ub}}\rangle$ in estimate:
  if $s_{\text{ub}} < \tau$:
    terminate
  join A_i and B_i, updating result set and τ as appropriate

Fig. 6. Our scheduling algorithm.

If each MBR has r points, sorting the MBR pairs takes

$$O\left(\frac{nm}{r^2}\log\frac{nm}{r^2}\right) \quad (26)$$

time. If the pruning ratio is ρ, then $\rho nm/r^2$ MBR pairs must be joined, at a cost of $O(r\log r + rk)$ each. The total complexity is therefore

$$O\left(\frac{nm}{r^2}\left(\log\frac{nm}{r^2} + \rho r\log r + \rho rk\right)\right). \quad (27)$$

In our experiments, the pruning rate is at the order of 10^{-10}, so for large datasets, the running time is dominated by sorting. Thus, we expect that absolute running time can be improved by first finding some of the best pairs by Hoare's Find algorithm [28] or another selection algorithm, thereby increasing the threshold to a level where most of the MBR pairs can be pruned. We did not pursue this approach.

C. A Space Partitioning Algorithm

As an alternative to the plane sweep algorithms, we examine a divide-and-conquer (D&C) method. The method can be used both for threshold PSJs and top-k PSJs.

Given two MBRs A_i and B_j, the algorithm first checks whether they can be pruned based on their s_{ub}. If not, each MBR is split into 2^d quadrants (where d is the number of dimensions, including the probability dimension), and processes each pair of quadrants recursively. When there are only a few points left, the subdivision stops, and a nested loop finishes the join.

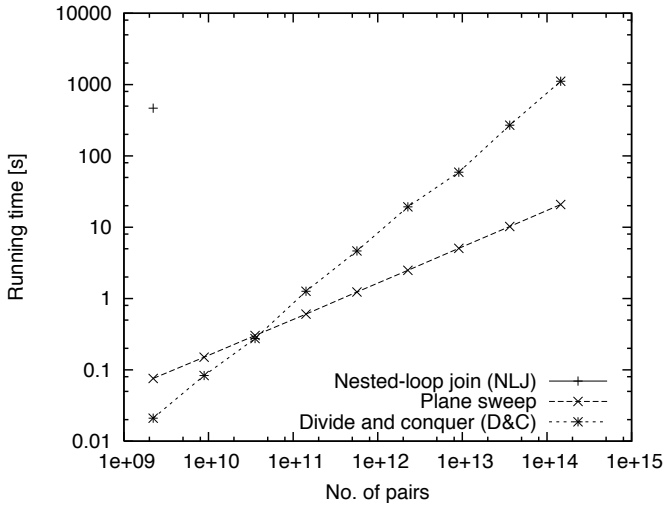

Fig. 7. For large datasets, the plane sweep algorithm for threshold probabilistic spatial joins is orders of magnitude faster than the divide and conquer algorithm.

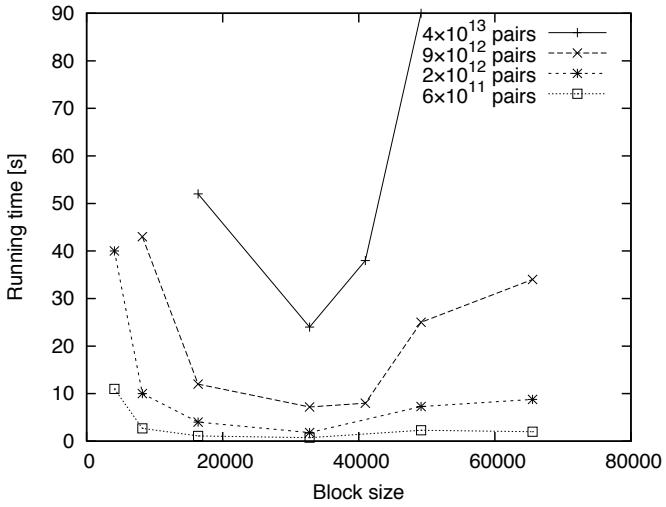

Fig. 8. The ideal block size balances the work to decide which pair of MBRs to join next with the work to actually join them.

Although its worst-case performance is not ideal, the algorithm works for any L_p distance metric, and its performance is good for small datasets.

V. Experimental Evaluation

This section evaluates our algorithms through experiments on real data. All experiments were run on a computer with Intel Pentium D 3.2 GHz CPU, 2 MB cache, 2 GB RAM, and Linux 2.6.18.

We used two datasets of 43k and 52k probabilistic points, respectively, derived from probabilistic segmentation [10] of wholemount micrographs of horizontal cells. For scalability experiments, the dataset was artificially enlarged by randomly copying cells and translating them in space. The implementation and data are available for download from www.ljosa.com.

To assess the performance of the plane sweep algorithm for threshold PSJ, we varied the number of pairs from 10^9 to 10^{14} and measured the running time of the three join methods (nested loop join, D&C, and plane sweep). The threshold was chosen so the queries returned 50–100 pairs. Nested-loop join was too slow to finish for the large datasets, but the running times for plane sweep and D&C are plotted in Figure 7 on a log-log scale. We see that the plane sweep algorithm is faster than the divide and conquer method, except for very small datasets (which can already be joined in less than 0.3 s). For larger datasets, the plane sweep is significantly faster. For instance, two sets of 10^{14} points can be joined 100 times faster with our plane sweep algorithm.

The next experiment explores the block size parameter of the scheduling algorithm. Figure 8 shows the running time for datasets of several sizes, and for different block sizes. We see that increasing the block size improves the speed of the algorithm. This is because the prediction (sorting) step dominates the running time when there are many small blocks. Increasing the block size too much, however, causes the running time to go up because the time to join two blocks becomes significant. Based on this experiment, we decided to use a block size of 2^{15} points for the remaining experiments.

Next, we investigated the scheduling algorithm's ability to prune pairs of MBRs. Figure 9 shows how the threshold increased during an execution of the top-k algorithms (plane sweep with and without scheduling) on a dataset of 9×10^{12} pairs. We set k to 20. Without scheduling, pairs are examined in an order based solely on the x-coordinates of the points. As a result, it took the algorithm almost 10 s to find good pairs. Thereafter, the threshold increased gradually. As long as the threshold was low, the algorithm continued to process candidate pairs that scored too low to affect the final result. In contrast, the scheduling algorithm invested about 2 s in ordering the pairs of MBRs. It was then able to find some high-scoring pairs very quickly, and thereby raise the threshold to just below its final value. As a result, most of the remaining pairs of MBRs were pruned.

Figure 10 shows how the techniques for top-k PSJ scale with the number of pairs. The first, third, and fourth curves show the running time of the scheduling algorithm using NLJ, divide and conquer, and plane sweep, respectively, to join the MBRs. Plane sweep and D&C lead to an order of magnitude improvement compared to NLJ, and this speedup is independent of dataset size. The difference between plane sweep and D&C is small because there are only 33k points in each MBR. The second curve shows the running time for plane sweep without scheduling. We see that as datasets grow very large, the lower time complexity of the pure plane sweep algorithm prevails, although the scheduling algorithms is as much as an order of magnitude faster for datasets of more moderate size.

VI. Related Work

Our score function for objects of uncertain extent (Definition 4) mirrors the *minimum matching distance*, which is used

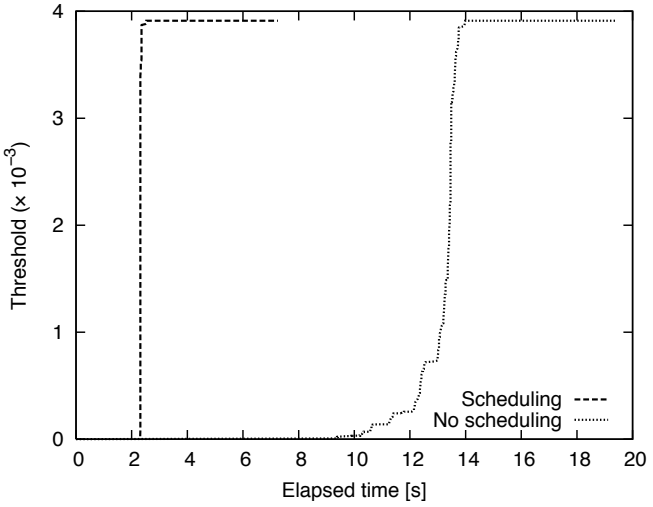

Fig. 9. Without scheduling, the threshold for the plane sweep algorithm for top-k PSJ stays at zero for a long time, then increases gradually. The best results are not found until late in the search. The scheduling algorithm, in contrast, invests some time on preprocessing; then, some good pairs are found quickly, raising the threshold and pruning most of the remaining candidate pairs.

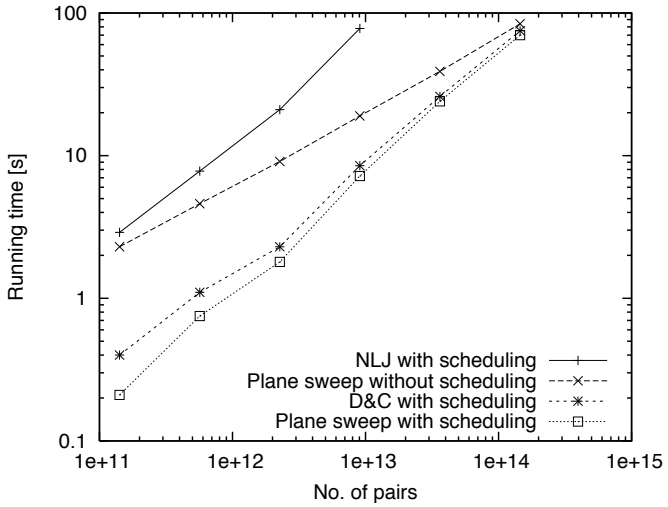

Fig. 10. With scheduling, plane sweep is an order of magnitude faster than NLJ, regardless of database size. As datasets grow very large, the lower time complexity of the pure plane sweep algorithm prevails, although the scheduling algorithms is as much as an order of magnitude faster for datasets of more moderate size.

for CAD objects represented as vector sets [29], [30].

Spatial joins have been studied extensively, and many authors [2], [3], [4], [5] employ some form of plane sweep. There are significant differences, however, because their sweep algorithms detect intersections between two sets of rectangles, where each rectangle is a minimum bounding rectangle (MBR) of an object. Their objects have exact extent, so as long as the rectangles fit reasonably tightly around the objects, the distance between two rectangles will be a relatively tight lower bound for the distance between the objects they contain. Intersection between rectangles (possibly after dilation [31] in order to find objects within a certain distance of each other) is therefore a good pruning step. In contrast, our objects consist of probabilistic points, so the distance within which another point must be in order to match this object varies with the point's probability. Thus, techniques based on rectangle intersection would produce many false positives.

Others have also scheduled spatial joins and used MBRs for pruning. Brinkhoff, Kriegel, and Seeger [3] compute spatial joins of deterministic sets of rectangles recursively on R-trees. Huang, Jing, and Rundensteiner [4] also perform spatial joins on R-trees, but traverse them breadth-first in order to optimize I/O better. Kahveci, Lang, and Singh [32] use a prediction matrix to schedule I/O-bound joins. Because these techniques are for non-probabilistic spatial joins, there is a certain set of page pairs that will need to be joined. Our situation is different because we are answering a *top-k* PSJ query. Thus, the results are not all equally important but have scores. This flavors the way we do the scheduling: For us, finding a high-scoring pair soon is more important than reducing I/O in the future because the joins we plan for the future may be pruned if we find a high-scoring pair soon.

Abel et al. [1] discuss caching strategies for spatial join. Faloutsos et al. [33] use power laws to estimate the selectivity of spatial joins. Ravada et al. [6] take a graph partitioning approach to scheduling spatial joins.

Fagin's algorithm [34] and the threshold algorithm [35] are top-k algorithms for joins, and several top-k algorithms have been proposed for probabilistic data under different models and assumptions [36], [37], [16]. These algorithms do not work for spatial joins, for they rely on primary key lookups: After finding an object in one dataset (source), they search the other datasets for the same object so its score can be determined. In a spatial join, however, we are not looking for the same object in two datasets, but for all spatially proximate objects.

The only previous PSJ algorithm of which we are aware is that of Kriegel et al. [38]. The model is different in that the objects have no extent, but are point objects. Samples of the position are clustered using k-means and stored in an R-tree-like index structure, which can then be used to answer forms of threshold and top-1 PSJ queries in which pairs are ordered by probability of being within a fixed distance threshold. The implementation and datasets were not available for experimental comparison.

Patel and DeWitt [5] perform spatial joins on large datasets that have no index structure. They partition the dataset so two partitions fits in memory, then join pairs of partitions by plane sweep. Arge et al. [2] take a similar approach, but partitions the dataset along one axis. Their algorithm has optimal time and I/O complexity.

VII. Conclusion

Spatial joins of probabilistic objects have many applications in geographical information systems and biomedical image analysis. They are technically challenging because the decision of whether to include a pair of points in the result depends on not only their distance from each other, but also both

their probabilities. In addition, finding the top-ranking results requires that we solve a spatial join and a top-k query at the same time.

We have proposed threshold probabilistic spatial joins (PSJs) and top-k PSJs and presented efficient algorithms for answering them. Our plane sweep algorithm for threshold PSJs joins n points with m other points in $O((n+m)(\log n+k))$ time, where k is the number of results above the threshold. The algorithm extends to multidimensional data.

We have presented a variant of the plane sweep algorithm that can answer top-k PSJs. Our index-based scheduling method finds high-scoring results early, thereby speeding up the search by an order of magnitude.

Acknowledgements

This work was supported in part by grant no. ITR-0331697 from the NSF. Horizontal cell micrographs were provided by Geoffrey P. Lewis from the laboratory of Steven K. Fisher at UCSB.

References

[1] D. J. Abel, V. Gaede, R. A. Power, and X. Zhou, "Caching strategies for spatial joins," *GeoInformatica*, vol. 3, no. 1, pp. 33–59, 1999.

[2] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. S. Vitter, "Scalable sweeping-based spatial join," in *Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB)*, 1998, pp. 570–581.

[3] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, "Efficient processing of spatial joins using R-trees," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 1993, pp. 237–246.

[4] Y.-W. Huang, N. Jing, and E. A. Rundensteiner, "Spatial joins using R-trees: Breadth-first traversal with global optimizations," in *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB)*, 1997, pp. 396–405.

[5] J. M. Patel and D. J. DeWitt, "Partition based spatial merge join," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, 1996, pp. 259–270.

[6] S. Ravada, S. Shekhar, C. tien Lu, and S. Chawla, "Optimizing join index based spatial join processing: A graph partitioning approach," in *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, 1998, pp. 302–308.

[7] M. Martone, S. Zhang, A. Gupta, X. Qian, H. He, D. Price, M. W. M, S. Santini, and M. Ellisman, "The Cell-Centered Database: A database for multiscale structural and protein localization data from light and electron microscopy," *Neuroinformatics*, vol. 1, no. 3, pp. 379–396, 2003.

[8] A. K. Singh, B. Manjunath, and R. F. Murphy, "A distributed database for bio-molecular images," *SIGMOD Record*, vol. 33, no. 2, pp. 65–71, 2004.

[9] J. R. Swedlow, I. Goldberg, E. Brauner, and P. K. Sorger, "Informatics and quantitative analysis in biological imaging," *Science*, vol. 300, pp. 100–102, 2003.

[10] V. Ljosa and A. K. Singh, "Probabilistic segmentation and analysis of horizontal cells," in *Proceedings of the Sixth IEEE International Conference on Data Mining (ICDM)*, Dec. 2006.

[11] O. Camoglu, T. Can, and A. K. Singh, "Integrating multi-attribute similarity networks for robust representation of the protein space," *Bioinformatics*, vol. 22, no. 13, pp. 1585–1592, 2006.

[12] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, "Querying imprecise data in moving object environments," *IEEE Transactions on Knowledge Engineering*, vol. 16, no. 9, pp. 1112–1127, Sept. 2004.

[13] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter, "Efficient indexing methods for probabilistic threshold queries over uncertain data," in *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, 2004.

[14] V. Ljosa and A. K. Singh, "APLA: Indexing arbitrary probability distributions," in *Proceedings of the 23nd International Conference on Data Engineering (ICDE)*, 2007.

[15] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar, "Indexing multi-dimensional uncertain data with arbitrary probability density functions," in *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*, 2005.

[16] C. Ré, N. Dalvi, and D. Suciu, "Efficient top-k query evaluation on probabilistic data," in *Proceedings of the 23nd International Conference on Data Engineering (ICDE)*, 2007, pp. 886–895.

[17] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. Hambrusch, "Indexing uncertain categorical data," in *Proceedings of the 23nd International Conference on Data Engineering (ICDE)*, 2007, pp. 616–625.

[18] W. K. Ngai, B. Kao, C. K. Chui, R. Cheng, M. Chau, and K. Y. Yip, "Efficient clustering of uncertain data," in *Proceedings of the Sixth IEEE International Conference on Data Mining (ICDM)*, 2006, pp. 436–445.

[19] J. C. Nekola and P. S. White, "The distance decay of similarity in biogeography and ecology," *Journal of Biogeography*, vol. 26, pp. 867–878, 1999.

[20] R. N. Shepard, "Toward a universal law of generalization for psychological science," *Science*, vol. 237, pp. 1317–1323, 1987.

[21] G. Lewis, K. Linberg, and S. Fisher, "Neurite outgrowth from bipolar and horizontal cells after experimental retinal detachment," *Investigative Ophthalmology & Visual Science*, vol. 39, no. 2, pp. 424–434, Feb. 1998.

[22] S. K. Fisher and G. P. Lewis, "Müller cell and neuronal remodeling in retinal detachment and reattachment and their potential consequences for visual recovery: a review and reconsideration of recent data," *Vision Research*, vol. 43, pp. 887–897, 2003.

[23] G. P. Lewis, K. A. Linberg, and S. K. Fisher, "Neurite outgrowth from bipolar and horizontal cells after experimental retinal detachment," *Investigative Ophthalmology & Visual Science*, vol. 39, no. 2, pp. 424–434, Feb. 1998.

[24] J. L. Bentley and T. Ottmann, "Algorithms for reporting and counting geometric intersections," *IEEE Transactions on Computing*, vol. C-28, pp. 643–647, 1979.

[25] D. Eppstein, M. T. Goodrich, and J. Z. Sun, "The skip quadtree: A simple dynamic data structure for multidimensional data," in *Proceedings of the Twenty-First Annual Symposium on Computational Geometry (SCG 2005)*, 2005, pp. 296–305.

[26] E. Castillo, A. S. Hadi, N. Balakrishnan, and J. M. Sarabia, *Extreme Value and Related Models with Applications in Engineering and Science*. Wiley, 2005.

[27] D. A. White and R. Jain, "Similarity indexing: Algorithms and performance," *Proceedings of SPIE*, vol. 2670, pp. 62–73, Mar. 1996.

[28] C. Hoare, "Algorithm 65: Find," *Communications of the ACM*, vol. 4, no. 7, pp. 321–322, 1961.

[29] H.-P. Kriegel, S. Brecheisen, P. Kröger, M. Pfeifle, and M. Schubert, "Using sets of feature vectors for similarity search on voxelized CAD objects," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, 2003.

[30] H. Yu and W. Niu, "Efficient similarity search on vector sets," Department of Computer Science, University of California, Santa Barbara, Tech. Rep. 2004-13, 2004.

[31] R. C. Gonzales and R. E. Woods, *Digital Image Processing*. Addison-Wesley, 1993.

[32] T. Kahveci, C. A. Lang, and A. K. Singh, "Joining massive high-dimensional datasets," in *Proceedings of the 19th International Conference on Data Engineering (ICDE)*, 2003, pp. 265–276.

[33] C. Faloutsos, B. Seeger, A. Traina, and J. Caetano Traina, "Spatial join selectivity using power laws," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 2000, pp. 177–188.

[34] R. Fagin, "Combining fuzzy information from multiple systems," *Journal of Computer and System Sciences*, vol. 58, no. 1, pp. 83–99, Feb. 1999.

[35] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," in *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2001, pp. 102–113.

[36] P. Agrawal and J. Widom, "Confidence-aware joins in large uncertain databases," Stanford University, Tech. Rep. 2007-14, 2007.

[37] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "Top-k query processing in uncertain databases," in *Proceedings of the 23nd International Conference on Data Engineering (ICDE)*, 2007, pp. 896–905.

[38] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz, "Probabilistic similarity join on uncertain data," in *Proceedings of the 11th International Conference on Database Systems for Advanced Applications*, 2006, pp. 295–309.