# On Computing Top-$t$ Most Influential Spatial Sites

Tian Xia Donghui Zhang* Evangelos Kanoulas Yang Du

College of Computer and Information Science
Northeastern University
Boston, MA 02115
{*tianxia, donghui, ekanou, duy*}*@ccs.neu.edu*

## Abstract

Given a set $O$ of weighted objects, a set $S$ of sites, and a query site $s$, the bichromatic RNN query computes the *influence set* of $s$, or the set of objects in $O$ that consider $s$ as the nearest site among all sites in $S$. The *influence* of a site $s$ can be defined as the total weight of its RNNs. This paper addresses the new and interesting problem of finding the top-$t$ most influential sites from $S$, inside a given spatial region $Q$. A straightforward approach is to find the sites in $Q$, and compute the RNNs of every such site. This approach is not efficient for two reasons. First, all sites in $Q$ need to be identified whatsoever, and the number may be large. Second, both the site R-tree and the object R-tree need to be browsed a large number of times. For each site in $Q$, the R-tree of sites is browsed to identify the *influence region* – a polygonal region that may contain RNNs, and then the R-tree of objects is browsed to find the RNN set. This paper proposes an algorithm called *TopInfluentialSites*, which finds the top-$t$ most influential sites by browsing both trees once systematically. Novel pruning techniques are provided, based on a new metric called *minExistDNN*. There is no need to compute the influence for all sites in $Q$, or even to visit all sites in $Q$. Experimental results verify that our proposed method outperforms the straightforward approach.

*This work was partially supported by NSF CAREER Award IIS-0347600.



**Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005**

## 1 Introduction

Since its introduction by [KM00], the reverse nearest neighbor (RNN) query has received considerable attention in recent spatial database research field. The RNN query can be roughly classified into two cases: the (traditional) monochromatic RNN query and the bichromatic RNN query. In the monochromatic case, there is only one spatial dataset $O$ of objects. We want to find the objects that are closer to a given location than to other objects. In the bichromatic case, there is another dataset $S$ of sites. The RNNs of a given site $s$ are the objects $o \in O$ such that $\forall s_i \in S$, $d(o, s) \leq d(o, s_i)$. Here $d()$ is the Euclidean distance function. In other words, the RNNs of $s$ are the objects that consider $s$ as the nearest site. The bichromatic RNN query has many practical applications. For instance, retrieve the residence buildings that consider a give supermarket as the nearest, or retrieve the mobile users that consider a given wireless station as the closest. However, the bichromatic case is intrinsically more difficult than the monochromatic case. For instance, the number of RNNs is limited in the monochromatic case (e.g. at most 6 in the 2D space) [Smi97], but unbounded in the bichromatic case.

This paper proposes and solves the problem of finding top-$t$ most influential sites, which is an extension to the bichromatic RNN problem. There are several differences. First, instead of taking as input a single site, we take as input a spatial region $Q$. Second, instead of computing the set of RNNs for a given site, we are interested in the influence, or the total weight of RNNs, of a site. In particular, we are interested in efficiently identifying the sites in $Q$ with the largest influence. Example applications are: find the two most influential supermarkets in Boston, or find the most influential wireless station in central USA.

**Definition 1** *Given a set of sites $S$, a set of weighted objects $O$, a spatial region $Q$, and an integer $t$, the* **top-$t$ most influential site query** *retrieves $t$ sites in $Q$ with the largest influences. Here the* **influence** *of a site $s \in S$ is the total weight of objects in $O$ that have $s$ as the nearest site.*

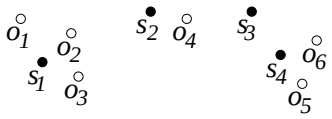


Figure 1: The most influential site is $s_1$, if all objects have weight=1 and $Q$ is the whole space.

For example, consider Figure 1. Let $t = 1$, and let $Q$ be the whole space. Further, let the weight of every object be 1. The most influential site is $s_1$, whose influence is 3. There are three objects $o_1$, $o_2$ and $o_3$ that have $s_1$ as the nearest site.

One approach to solve this problem is to use pre-computation. Each site stores its influence. At query time, we find the sites in $Q$ and return the top-$t$ most influential ones. This approach has two drawbacks. First, it is time consuming to maintain the pre-computed results upon updates. Second, the pre-computation binds a set of sites closely with a set of objects. Suppose a set of sites (e.g. supermarkets) may be queried against multiple sets of objects (e.g. the set of all residential buildings, or the set of store members' addresses). A site needs to maintain multiple influence values. For these reasons, we focus on algorithms which apply directly to two plain R-trees, one for sites and the other for objects, without any pre-computation.

Another approach, which does not use pre-computation, is to extend an existing solution to the bichromatic RNN query [SRAE01]. The idea of [SRAE01] to find the RNNs of a site $s$, is to dynamically construct the *influence region* (or Voronoi cell), by examining the R-tree of sites. Here, an influence region is defined as a polygon in space which encloses the locations that are closer to $s$ than to any other site. Once the influence region is computed, a range query in the R-tree of objects is performed to locate RNNs of $s$.

To find top-$t$ most influential sites in $Q$, we can perform a range query on $S$ to find the sites in $Q$, and then for each located site $s$, compute its RNNs using [SRAE01]. An optimization is to index the set of objects using an R-tree whose index entries store the total weight of objects in their sub-trees. If a sub-tree MBR is contained in an influence region, there is no need to examine the sub-tree.

Nevertheless, this approach is not efficient for the following two reasons. First, all sites in $Q$ need to be identified whatsoever. In practice, $Q$ may be large (e.g. "central USA") and thus may contain many sites. We know that out of the many sites in $Q$, only $t$ sites (for some small $t$) are needed. So, intuitively, pruning techniques should be possible. That is, without going all the way to the leaf level of the site R-tree, it may be possible to determine that a sub-tree (intersecting with $Q$) only contains sites with relatively small influences and therefore can be pruned. A bigger efficiency issue is that the two R-trees need to be browsed many times. In more detail, for every site in $Q$, the R-tree of sites is browsed to identify the influence region, and then the R-tree of objects is browsed to compute the corresponding influence.

This paper presents an algorithm called *TopInfluentialSites*, which finds the top-$t$ most influential sites by browsing both R-trees (aR-trees [1] to be more precise) once. Novel pruning techniques are provided, based on a new metric called *minExistDNN*. There is no need to compute the influence for all sites in $Q$, or even to locate all sites in $Q$.

Here $minExistDNN_{S_1}(O_1)$ is a distance defined between two minimum bounding rectangles (MBRs) of R-trees. It is the smallest distance which is guaranteed to be an upper bound of the distance between any object in $O_1$ to its nearest site in $S_1$. Here 'DNN' stands for *distance to nearest neighbor*. 'Exist' represents the fact that from any object in $O_1$, within this distance there must exist a site in $S_1$. And 'min' corresponds to the fact that this metric is the smallest upper bound. It reminds us the metrics $minDist$, $maxDist$ and $minMaxDist$ [RKV95]. One difference is that this new metric is directional. That is, switching $S_1$ with $O_1$ will get a different distance. This is a useful metric. We have: if $minExistDNN_{S_1}(O_1) < minDist(O_1, S_2)$, no object in $O_1$ will consider a site in $S_2$ to be the nearest site. It can be used to prune the search space for NN and/or RNN related queries, such as the problem of finding top-$t$ most influential sites. Nevertheless, it is NOT straightforward how to compute it. In Section 3 we formally define the metric and provide an algorithm to compute it.

For the top-$t$ most influential site query, the algorithm and the data structures used are outlined below. For ease of presentation, let's use $O$ and $S$ to also represent the R-tree of objects in $O$ and the R-tree of sites in $S$, respectively. We keep three queues. One queue, $queue_{SIN}$ (reads Q-ESS-IN), keeps the (index or leaf) entries from $S$ that intersect (or are inside) $Q$. Each entry keeps a lower bound and an upper bound for the influence of sites in the entry's spatial range. Another queue keeps the entries from $O$ that *affects* some entries in $queue_{SIN}$. Here an entry $O_i$ *affects* an entry $S_j$, if and only if there does not exist another entry $S_k$ in the queues, such that $minExistDNN_{S_k}(O_i) < minDist(O_i, S_j)$. Intuitively, if some object in $O_i$ considers some site in $S_j$ as the closest site, $O_i$ affects $S_j$. The third queue keeps the entries from $S$ that are outside $Q$, but are affected by some entries in the queue of $O$. Initially, the entries in the root nodes of $O$ and $S$ are selectively loaded into the three queues. The metric $minExistDNN$ is

[1] The aR-tree [PKZT01], is an R-tree in which each index entry stores the total weight (or some other aggregate) of the objects in the subtree. The aggregate values can be maintained easily along with each update.

used to determine which $O_i$ does not affect which $S_j$. The algorithm then progressively chooses some entries from one of the three queues to *expand*, and updates the content of the queues. Here, to *expand* an index entry means to retrieve the referenced node from disk and visit its content. It terminates when there are $t$ sites in $queue_{SIN}$ whose lower bounds are no less than the upper bounds of all remaining entries. These $t$ sites are reported as the most influential sites.

What's crucial is: how do we choose the next entry to expand? It is not a good choice to expand all entries from one queue (e.g. $queue_{SIN}$) to the leaf level before expanding other queues. It is also not a good choice to expand in a round-robin manner. A good strategy should be to expand according to the actual data, satisfying certain goals. For example, (a) The lower bound and upper bound estimations should be as accurate as possible. (b) An $O_i$ should affect as few $S_j$ as possible, and vice versa. (c) The expansion should quickly reveal some highly influential sites and work on increasing their lower bounds.

The key results of this paper are:

- We propose a new metric called *minExistDNN*, and an algorithm to calculate it. It can be used to prune search space when computing the top-$t$ most influential sites. In the future we expect this metric to be used in other NN/RNN related spatial database problems.
- We propose and solve the top-$t$ most influential site query. Our algorithm works directly on the datasets indexed by R-trees, without the need to pre-compute the influence of all sites or the distance from every object to its closest site. Our algorithm systematically chooses the next entry to expand, according to the actual data distribution. Experimental results are provided which illustrates the efficiency of our algorithm.

The rest of the paper is organized as follows. Section 2 reviews related work on RNN queries. Section 3 proposes the metric *minExistDNN*, with an algorithm to compute it. Section 4 presents our algorithm to compute top-$t$ most influential sites. Section 5 presents experimental results. Finally, Section 6 concludes the paper.

## 2 Background

The RNN problem was first studied in [KM00]. The idea is to pre-compute, for each object, the distance *dnn* to its nearest site. Thus each object corresponds to a circle, whose center is the object and whose radius is the *dnn* of it. Besides the R-tree which indexes the original objects, a separate R-tree is maintained which indexes the set of such circles. The problem of finding RNNs (influence set) is then reduced to finding the circles that contain the query point.

To eliminate the need of storing two R-trees, Yang and Lin [YL01] proposed to store some additional information in the original R-tree, so that it logically stores a circle per object. The extension is: every leaf record stores *dnn*, and every index record also stores *dnn* – the max *dnn* for all objects in the sub-tree. They call the extended structure the Rdnn-tree. The benefit for the RNN search is as follows. Let an index node be $N$, and let the query location be $l$. If the distance between $l$ and the MBR of $N$ is bigger than $N.dnn$, there is no need to search the sub-tree rooted by $N$. Lin *et al.* [LNY03] proposed a method to bulk-insert the Rdnn-tree.

The work based on storing circles [KM00, YL01], either physically or logically, can be used to answer both the monochromatic RNN query and the bichromatic RNN query. However, the pre-computation incurs extra update cost to maintain the correct *dnn* for each object or index entry. Furthermore, for the bichromatic RNN case, the solution binds a set of objects closely with a set of sites. Below we review some algorithms without pre-computation.

For the monochromatic RNN query, there are two approaches without using pre-computation. An earlier approach [SAE00] divides the space into six 60° regions centered by the query location $l$. It was proved that the only candidates of the RNNs are the six nearest neighbors (NN) of $l$ in each region. So [SAE00] finds the six NNs, and then check to see if each of them really considers $l$ as NN. A recent result which is more efficient was introduced by [TPL04]. Given the query location $l$, we first find its NN, say $o_1$. Consider the bisector of $l$ and $o_1$. All objects on the side of $o_1$ (except $o_1$ itself) can be pruned, since their distances to $o_1$ is no more than the distances to $l$. Next, in the unpruned space, the NN to $l$ is found, and the space is further pruned. Finally, the unpruned space does not contain any object. The only candidates of RNNs are the identified NNs. The refinement step, which removes false positives, uses the previously pruned MBRs so that no tree node is visited twice throughout the algorithm.

An algorithm which computes bichromatic RNNs, without pre-computation, was proposed by [SRAE01]. The idea is to dynamically construct the *influence region* of the query location $l$. Here, the influence region is defined as a polygon in space which encloses and only encloses all possible RNNs of $l$. This is equivalent to the *Voronoi cell* enclosing $l$ [BKOS97]. Conceptually, if we draw a bisector line between $l$ and a site $s$, any object located on the $l$ side of the bisector will have smaller Euclidean distance to $l$ than to $s$. The $l$ side of the bisector is a half plane. If we compare $l$ against all sites and take the intersection of these $l$-side half planes, we get the Voronoi cell containing $l$.

Of course, to compare with all sites is expensive. As proved in [SRAE01], we only need to examine sites in a certain rectangle which is computed as follows

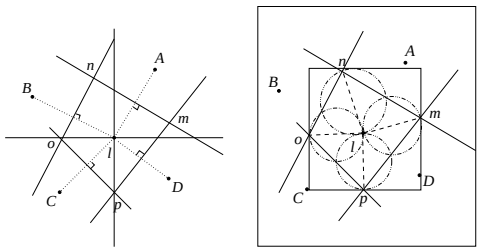


Figure 2: Identifying the rectangle which is guaranteed to contain all sites needed for computing a Voronoi cell.

(Figure 2). First, we find the nearest site of $l$ in each quadrant formed by axes parallel to the original axes that pass through $l$. This is a constrained NN query [FSAE01]. Let the four NNs be $A$, $B$, $C$ and $D$ in the first, second, third and fourth quadrant respectively. By using the four NNs we form an approximate influence region, *mnop* (Figure 2(a)). Then, we draw four circles. Each one passes through $l$ and one of the four vertices of the approximate influence region, e.g. $m$, with diameter being the distance between $l$ and that vertex, e.g. $d(l, m)$ (Figure 2(b)). Next, we draw the MBR of the four circles. Finally, each edge of the MBR is expanded away from $l$ by the same distance from $l$ to the edge. And we get the rectangle that is guaranteed to contain all sites needed for computing the Voronoi cell of $l$. A range query is performed using this rectangle on the R-tree of sites. The Voronoi cell of $l$ can be computed by only examining these identified sites.

An object is located in the Voronoi cell of $l$ if and only if it is closer to $l$ than to any site. So to find the RNNs of $l$, a range query using the Voronoi cell is performed on the R-tree of objects.

There is some other RNN work as well. A computational geometry work [MVZ02] on finding RNNs also uses the idea of storing circles. The method focuses on asymptotic behavior rather than experimental evaluation. Singh *et al.* [SFT03] proposed an approximate RNN solution. The idea is to first find $k$ (for some system parameter $k$) NNs of the query location $l$, then eliminate false positives. The problem is that the method may have false negatives. The NN query and RNN query for moving objects was discussed in [BJKv02]. Finding RNNs and RNN aggregations over a data stream was proposed by [KMS02]. The RNN aggregation queries they considered are different from our problem. Finding RNNs in a graph was discussed in [YPMT05].

## 3 The New *minExistDNN* Metric

### 3.1 Motivation

To find the top-$t$ most influential sites in a query region $Q$, a reasonable algorithm should start with pushing index entries from the root node of $S$, that intersect with $Q$, into a processing queue. Then the entries should be selectively expanded to reveal the $t$ most influential sites. Let's name such a queue as $queue_{SIN}$. An optimal algorithm should focus on expanding the entries that contain these $t$ sites.

To differentiate the entries in $queue_{SIN}$, we propose to use a priority queue, where the sorting key is some upper bound of the influence for every location in the entry and it can be computed in the following way. Consider an entry $O_i$ in the root node of $O$, and an entry $S_j$ in $queue_{SIN}$. If some object in $O_i$ may consider some site in $S_j$ as NN, we say $O_i$ *affects* $S_j$, and the total weight of objects in $O_i$ is added to the upper bound of $S_j$. (This total weight is maintained along with the index entry which stores $O_i$.) Here in the *affect* relationship, one or both entries can be leaf entries (i.e. sites or objects). For example, an object $o$ affects $S_j$, if it is possible that $o$'s NN is in $S_j$.

Clearly, we should be able to identify the set of MBRs from $O$ that affect each MBR of $S$. One intuition is that if $O_1$ is close to $S_1$ but is far from $S_2$, it is likely that $O_1$ does not affect $S_2$. To be more precise, $O_1$ does not affect $S_2$ if for every location $l$ in $O_1$, it is guaranteed that the distance from $l$ to its nearest site in $S_1$ is smaller than the minimum distance from $l$ to $S_2$.

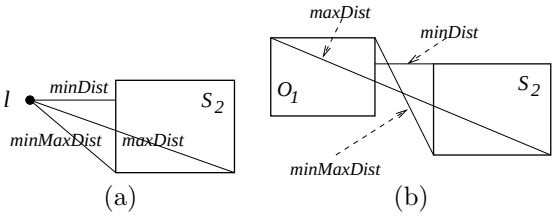


Figure 3: The three metrics defined in [RKV95], between a point and a rectangle. The metrics can also be defined similarly between two rectangles.

This reminds us the three metrics defined in [RKV95], as shown in Figure 3(a). Given a point $l$ and a rectangle $S_2$, $minDist(l, S_2)$ and $maxDist(l, S_2)$ are the lower bound and upper bound, respectively, of the distance between $l$ and any possible site in $S_2$. The third metric, $minMaxDist(l, S_2)$, is the smallest distance which guarantees that, within this distance from $l$, there exists a site in $S_2$. The computation of $minMaxDist$ is as follows. For each edge of $S_2$, compute the maximum distance between $l$ and any point on the edge. This must be between $l$ and one of the two end point of the edge. Due to the property of the (R-tree) MBR, each edge of $S_2$ must contain at least one site, and the maximum distance from $l$ to each edge is an upper bound of the $minMaxDist(l, S_2)$. Therefore, out of the four distances, pick the minimum one. Figure 3(b) shows that the three metrics can be straightforwardly extended to the case of two rectangles.

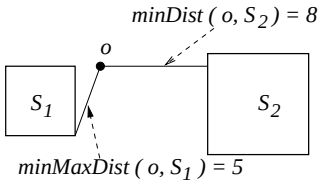


Figure 4: Illustrating the case when an object $o$ does not affect $S_2$, due to the existence of $S_1$.

**Theorem 1** *An object $o$ does not affect $S_2$, if there exists $S_1$ such that*

$$minMaxDist(o, S_1) < minDist(o, S_2)$$

Due to the space limitations, the proofs of theorems and lemmas in this paper are omitted.

What we really need is an condition which determines that an MBR $O_1$ does not affect $S_2$, given $S_1$. What if we simply replace $o$ in Theorem 1 with $O_1$? Well, it does not work. As shown in Figure 5, although $minMaxDist(O_1, S_1) < minDist(O_1, S_2)$, it is still possible that on object $o$ in $O_1$ considers some site in $S_2$ as its nearest site.

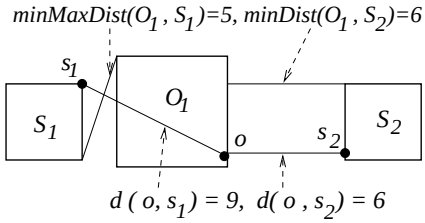


Figure 5: Replacing $o$ with an MBR $O_1$ in Theorem 1 does not work.

Instead of using $minMaxDist(O_1, S_1)$, the correct condition should use some distance which (i) guarantees that, within this distance, every location in $O_1$ can reach a site in $S_1$; and (ii) is the smallest one among all distances satisfying (i). We introduce a metric called $minExistDNN$ which satisfy these two conditions.

### 3.2 The Metric *minExistDNN*

**Definition 2** *Given two MBRs $O_1$ and $S_1$, the minimum upper bound of the distance from an object in $O_1$ to its nearest site in $S_1$ is defined as:*

$$minExistDNN_{S_1}(O_1) =$$
$$\max\{minMaxDist(l, S_1) \mid \forall \texttt{location}\ l \in O_1\}$$

Note that unlike the existing three metrics (minMaxDist, etc.), in $minExistDNN$ we cannot swap the input parameters. In other words, $minExistDNN_{S_1}(O_1) \neq minExistDNN_{O_1}(S_1)$. This is why we put $S_1$ as subscript.

Let's see why this metric satisfies the two requirement given at the end of the previous sub-section. (i) For every location $l \in O_1$, $minMaxDist(l, S_1)$ is an upper bound of the distance between $l$ and its nearest site in $S_1$. So within $minExistDNN_{S_1}(O_1)$, which is the maximum of all such $minMaxDist$, every location in $O_1$ will meet a site in $S_1$. (ii) Consider any distance $d < minExistDNN_{S_1}(O_1)$. According to Definition 2, there must exist a location $l_M \in O_1$ s.t. $d < minMaxDist(l_M, S_1)$. So $d$ does not satisfy (i).

**Theorem 2** *An MBR $O_1$ does not affect $S_2$, if there exists $S_1$ such that*

$$minExistDNN_{S_1}(O_1) < minDist(O_1, S_2)$$

Figure 6 illustrates $minExistDNN_{S_1}(O_1)$ in two scenarios. In Figure 6(a), the distance is between a corner point of $O_1$ and a corner point of $S_1$. The correctness is easy to see. However, the correctness of Figure 6(b) is not intuitive. The next section presents an algorithm that calculates this metric.

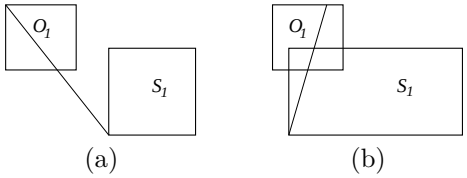


Figure 6: Illustration of $minExistDNN_{S_1}(O_1)$.

### 3.3 Calculating *minExistDNN*

Let's ***associate*** every location $l$ in space with a corner point of $S_1$ – the corner whose distance to $l$ is equal to $minMaxDist(l, S_1)$. If all locations in $O_1$ were associated with the same corner of $S_1$, the computation of $minExistDNN_{S_1}(O_1)$ would become an easy task. That is, we simply compute the maximum distance between that corner of $S_1$ with $O_1$. However, different locations in $O_1$ may be associated with different corner points of $S_1$.

In Step 1 below, we present a space partitioning scheme such that every location in the same partition is associated with the same corner of $S_1$. This immediately suggests an algorithm for calculating $minExistDNN_{S_1}(O_1)$. That is, we divide $O_1$ into multiple sub-regions, one in each partition. Then, for each sub-region, we calculate the maximum distance from the associated corner of $S_1$ to the sub-region.

In Step 2, based on the space partitioning, we develop a more efficient algorithm which computes $minMaxDist(l, S_1)$ for up to eight locations in $O_1$. The maximum of these eight distances is the required $minExistDNN$.

#### 3.3.1 Step 1: space partitioning

**Lemma 1** *Given a location $l$ and an MBR $S_1$, $minMaxDist(l, S_1)$ is the distance between $l$ and the second closest corner point of $S_1$.*

Suppose every location in space is associated with the second closest corner of $S_1$. Figure 7 shows a partitioning of space, such that all locations in the same partition are associated with the same corner. (Our partitions are different from the Voronoi diagram of order 2 of the four corner points, whose partitions are associated with two closest corner point.)

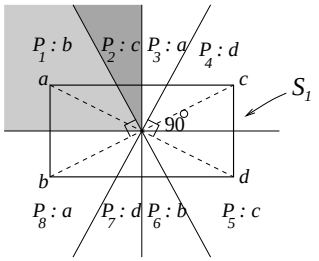


Figure 7: A partitioning of space such that in every partition, all locations take the same corner of the rectangle as the second closest corner.

The space partitioning is created by four lines intersecting at the same location: the center of $S_1$. The four lines are the perpendicular bisectors (in short, bisectors) of $S_1$'s edges and diagonals. Notice that every two edges, e.g. top and bottom, have the same bisector. The associated corner for locations in each partition is shown in Figure 7. For instance, $P_1 : b$ means all locations in partition $P_1$ are associated with $b$.

Let's see why this partitioning is correct. Consider a quadrant of space: the union of $P_1$ and $P_2$ (Figure 7). Any location in this quadrant considers the upper-left corner $a$ as the closest corner. We know the second closest corner is either $b$ or $c$, but not $d$. So if we draw a bisector of diagonal $\overline{bc}$, any location on the $b$ side of the bisector ($P_1$) considers $b$ to be closer than $c$, and thus takes $b$ as the second closest corner. Similarly, any location in $P_2$ considers $c$ as the second closest corner. Same with the other partitions.

### 3.3.2 Step 2: the algorithm

We argue that the location in $O_1$, whose distance to its associated corner of $S_1$ is the $minExistDNN_{S_1}(O_1)$, is one of, at most, eight candidate locations. The candidates are the four corners of $O_1$ plus the (up to four) intersection points between the border of $O_1$ and the diagonals' bisectors. Thus the algorithm to compute $minExistDNN_{S_1}(O_1)$ is to calculate $minMaxDist$ for each candidate location, and return the maximum.

We denote the intersections between $O_1$ and the two diagonal bisectors of $S_1$ as the *intersection points* of $O_1$, and the number of the intersection points of $O_1$ is at most 4. Also, an *intersection segment* of $O_1$ is a line segment on one of the diagonal bisectors inside rectangle $O_1$. To prove our claim above, we introduce two lemmas.

**Lemma 2** *For any point $p \in O_1$, there exists a point, $p'$, either on some border or on some intersection segment of $O_1$, such that $minMaxDist(p', S_1) \geq minMaxDist(p, S_1)$.*

**Lemma 3** *For any point $p'$ on the borders or the intersection segments of rectangle $O_1$, there exists a corner or intersection point $p''$, such that $minMaxDist(p'', S_1) \geq minMaxDist(p', S_1)$.*

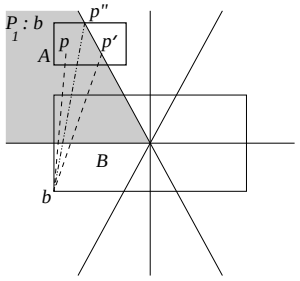


Figure 8: An example of Lemma 2 and 3, illustrating $minMaxDist(p', S_1) \geq minMaxDist(p, S_1)$ and $minMaxDist(p'', S_1) \geq minMaxDist(p', S_1)$.

An example of Lemma 2 and 3 is illustrated in Figure 8. For clarity, we only keep the information of partition 1. Point $p$ is an arbitrary point within $O_1$ in partition 1. An illustration of Lemma 2 is, by moving $p$ along the $x$-axis towards $p'$, $minMaxDist(p, S_1)$ increases. Therefore, $minMaxDist(p', S_1) > minMaxDist(p, S_1)$. An illustration of Lemma 3 is, by moving, now, $p'$ on the intersection segment towards $p''$, $minMaxDist(p', S_1)$ increases,too. Therefore, $minMaxDist(p'', S_1) > minMaxDist(p', S_1)$.

As a corollary of Lemma 2 and 3, we have:

**Theorem 3** *For any point $p \in O_1$, there exist a corner point or intersection point $p''$, such that $minMaxDist(p'', S_1) \geq minMaxDist(p, S_1)$.*

## 4 Finding Top-$t$ Most Influential Sites

This section designs an algorithm that computes top-$t$ most influential sites, by examining an R-tree $S$ of sites and an R-tree $O$ of objects. For each site $s$ in the query result, any correct algorithm needs to visit the R-tree node that stores $s$, as well as all of its ancestor nodes. We maintain a priority queue of entries (index entries or sites) from $S$, that are inside the query region $Q$. The queue is called $queue_{SIN}$. For clarity, we assume each MBR of any index entry in $S$ is either inside or outside $Q$. In Section 4.4 we address the case when an MBR partially intersects $Q$.

For each entry $S_j$ in $queue_{SIN}$, we store two values *minInfluence* and *maxInfluence*. If $S_j$ is a single site, these are a lower bound and an upper bound of its influence. If $S_j$ is an index entry, these are a lower bound

and an upper bound of the influence of the most influential site in the sub-tree of $S_j$. The algorithm stops when there are $t$ sites in $queue_{SIN}$, each of which has a *minInfluence* no less than the largest *maxInfluence* of all remaining entries. The algorithm aims at quickly making the estimates accurate, while focusing on expanding index entries which are highly likely to contain some of top-$t$ most influential sites. To expand an index entry $e$, it is replaced by the entries in the R-tree node referenced by $e$. Clearly, we cannot ignore the entries from $S$ whose MBRs are outside $Q$, or the entries from $O$. We maintain two queues, $queue_{SOUT}$ (reads Q-ESS-OUT) and $queue_O$, for these entries. Similar to the case of $queue_{SIN}$, at the beginning of the algorithm these two queues contain some entries in the root nodes of $S$ and $O$, which may be expanded as the algorithm runs on.

There are some challenging issues. One issue is, how to set and update *minInfluence* and *maxInfluence* of entries in $queue_{SIN}$? A crucial sub-issue is how do we determine whether some $O_i$ in $queue_O$ affects some $S_j$ in $queue_{SIN}$ or $queue_{SOUT}$. (Recall that $O_i$ *affects* $S_j$, if and only if there does not exist another entry $S_k$ in $queue_{SIN}$ or $queue_{SOUT}$, such that $minExistDNN_{S_k}(O_i) < minDist(O_i, S_j)$.) Another issue is, how to choose an index entry to expand? As mentioned in the introduction, a good expanding strategy should consider the actual data and should satisfy certain goals that increase the efficiency of the algorithm.

Section 4.1 formally defines the three queues and discusses how to set *minInfluence* and *maxInfluence*. Section 4.2 presents our algorithm, which embodies the strategy of choosing index entries to expand. Section 4.3 explores in more detail one step in the algorithm, which is to determine whether it is highly likely that an entry in $queue_O$ can be pruned (because it does not affect any entry in $queue_{SIN}$). With the introduction of yet another new metric called *minMinExistDNN*, we can mathematically measure the probability that some $O_i$ in $queue_O$ can be pruned by some $S_j$ in $queue_{SOUT}$. Finally, Section 4.4 extends the solution to the general case when the MBR of an index entry in $S$ may partially intersect $Q$.

### 4.1 Data Structures

To clearly define what we store in each of the three queues, let's introduce two concepts. An index entry $e$, from either $S$ or $O$, is said to be an *expanded* entry (by our algorithm), if the content of the tree node referenced by $e$ has been examined. That is, if the node has been retrieved from disk. All entries in an examined tree node are said to be *visited*. The three queues we use can only contain visited but not expanded entries. A non-visited entry is in some node not yet retrieved from disk. An expanded entry has been removed from the queues. We have:

- $queue_{SIN} = \{S_j | S_j$ is a visited but not expanded entry in $S$, whose MBR is inside $Q$ and whose *maxInfluence*$>0\}$.
- $queue_O = \{O_i | O_i$ is a visited but not expanded entry in $O$, which affects some entry in $queue_{SIN}\}$.
- $queue_{SOUT} = \{S_j | S_j$ is a visited but not expanded entry in $S$, whose MBR is outside $Q$ and which is affected by some entry in $queue_O\}$.

Here the **maxInfluence** for an entry $S_j$ in $queue_{SIN}$ is the total weight of visited but not expanded entries in $O$ that affect $S_j$. If $S_j$ has *maxInfluence*=0, all sites in $S_j$ have influence=0, and there is no need to keep $S_j$ in $queue_{SIN}$.

The only reason we keep $queue_O$ and $queue_{SOUT}$ is to compute the *minInfluence* and *maxInfluence* of entries in $queue_{SIN}$. So $queue_O$ only consists of entries from $O$ that affect at least one entry in $queue_{SIN}$, and $queue_{SOUT}$ only consists of entries from $S$ (but outside $Q$) that are affected by at least one entry in $queue_O$.

Here the **minInfluence** for an entry $S_j$ in $queue_{SIN}$ is a lower bound of the maximum influence for a site in $S_j$. It is computed in the following way. Consider some $O_i$ in $queue_O$ which *only* affects $S_j$, i.e. $O_i$ does not affect any other entry in $queue_{SIN}$ or $queue_{SOUT}$. Every object in $O_i$ considers some site in $S_j$ as the nearest site. There must exist a site $s_1$ in $S_j$ whose influence is at least $|O_i|/|S_j|$. Here $|O_i|$ is the total weight of objects in the sub-tree of $O_i$, while $|S_j|$ is the number of sites in the sub-tree of $S_j$. If there is another $O_i'$ which only affects $S_j$, there must exist a site $s_1'$ in $S_j$ whose influence is at least $(|O_i| + |O_i'|)/|S_j|$. In general, the *minInfluence* of an index entry $S_j$ in $queue_{SIN}$ is the $\sum(|O_i|)/|S_j|$ for every $O_i$ in $queue_O$ that only affects $S_j$. This is also true in case $S_j$ is a site ($|S_j| = 1$).

To maintain the content of the three queues, we need to know which $O_i$ affects which $S_j$. If an algorithm naively considers every MBR in $O$ to affect all MBRs in $S$, it either fails to identify the most influential sites, or is very inefficient. We should be able to find the minimum number of entries in $queue_{SIN}$ and $queue_{SOUT}$ affected by each entry in $queue_O$. We have addressed this issue in Section 3, where we pointed out that $O_1$ does not affect $S_2$, if there exists $S_1$ such that $minExistDNN_{S_1}(O_1) > minDist(O_1, S_2)$.

An implementation detail is that in $queue_O$ each $O_i$ maintains a list of pointers, one to each affected $S_j$ in $queue_{SIN}$ and $queue_{SOUT}$. Likewise, each $S_j$ keeps a list of pointers, one to each $O_i$ that affects it.

### 4.2 The Algorithm

Our algorithm which computes top-$t$ most influential sites is shown in Figure 9.

A crucial operation throughout the algorithm is **expand**. To expand an index entry from one of the three queues, we remove it as well as the links to it, while trying to insert entries in the referenced node into the same queue. For instance, suppose we want to expand an index entry $S_j$ in $queue_{SIN}$ (expanding an index entry in $queue_O$ or $queue_{SOUT}$ is similar). There must have some entries in $queue_O$ which affect $S_j$. In our data structure, $S_j$ keeps a list of pointers, referencing its affecting entries in $queue_O$. And each $O_i$ in $queue_O$ also points to $S_j$. These links should be removed. When examining the node referenced by $S_j$, we try to insert each child entry into the same queue as $S_j$. Here to identify the entries in $queue_O$ that may affect a child entry, we only need to examine an $O_i$ if it used to affect $S_j$. More formally, we have:

**Theorem 4** *If $O_i$ does not affect $S_j$, none of its children will affect $S_j$. If $S_j$ is not affected by $O_i$, none of its children will be affected by $O_i$.*

Due to entry expansions, we may need to re-evaluate the associations of some queue entries. Two simple pruning cases are (1) if a queue entry $S_j$ is not affected by any $O_i$'s, remove $S_j$ from the queue; (2) if $O_i$ does not affect any entry in the $queue_{SIN}$, remove $O_i$ from the queue. There is another more subtle case as follows. Suppose $S_j$ is expanded, and let $S_k$ be an entry in the referenced node. $S_k$ is checked for associations against $O_i$ that affects $S_j$. If $S_k$ is affected by $O_i$, we need also to examine if any entry originally affected by $O_i$ could be pruned from $O_i$ using $minExistDNN_{s_j}(O_i)$.

Step 1 of the algorithm examines the root nodes of $O$ and $S$ and builds up the three queues. This step works in the following way. First, the entries in the root node of $S$, whose MBRs are inside $Q$, are pushed into $queue_{SIN}$. Then, we select the entries from the root node of $O$ that affect at least one entry in $queue_{SIN}$, and push them into $queue_O$. Third, we push into $queue_{SOUT}$ the root entries of $S$ which are affected by some entries in $queue_O$. Finally, the entries in $queue_{SIN}$ not affected by any entry in $queue_O$ are removed.

Step 2 is a stopping condition. The algorithm stops when $t$ most influential sites are identified. Note that our algorithm can return a site without computing its actual influence. As long as the *minInfluence* of a site is big enough, we can report it. Of course, if we want to find the actual influence, we can keep expanding nodes until its *minInfluence* and *maxInfluence* become equal.

Step 3 makes sure that $queue_{SIN}$ has at least certain number of entries. The intuition is that if there are only a few sites in $Q$, the algorithm should focus on expanding most or all of them before expanding entries in $queue_O$ or $queue_{SOUT}$.

Step 4 picks the set (called $top\beta SIN$) of $\beta$ entries from $queue_{SIN}$ with *maxInfluence*. The rationale is that we want to focus on expanding the index entries whose corresponding sub-trees may contain some of the top-$t$ most influential sites. The index entries among them are expanded once. Then we need to give the related entries from the other two queues a chance. So Step 5 picks the set (called $C_O$) of entries from $queue_O$ that affect some entries in $top\beta SIN$.

At Step 6, we handle the case when some entry in $C_O$ has high probability to be pruned by expanding some entry $S_j$ in $queue_{SOUT}$. In this case, we should expand $S_j$ to prune that entry in $C_O$. This will result in more accurate influence estimation for entries in the picked $top\beta SIN$. This topic, including a mathematical definition of the probability, is discussed in detail below in Section 4.3.

At this step, we want to work on making the *minInfluence* and *maxInfluence* estimations of entries in $top\beta SIN$ as accurate as possible. That is to say, we need to focus on an entry in $C_O$ which, if expanded,

---

**Algorithm *TopInfluentialSites***($O$, $S$, $t$, $Q$).
Input: An R-tree $O$ of objects, an R-tree $S$ of sites, an integer $t$, and a spatial region $Q$
Action: Return the top-$t$ most influential sites in $Q$.

1. Push the root entries of $O$ and $S$ into $queue_{SIN}$, $queue_O$ and $queue_{SOUT}$.
2. If the $t$ entries from $queue_{SIN}$ with maximum *maxInfluence* are all sites, with *minInfluence* no less than the *maxInfluence* of all remaining entries in $queue_{SIN}$, **return** these $t$ sites as query result.
3. While $|queue_{SIN}| < \beta$ (for a given constant $\beta \geq 1$), expand all index entries in $queue_{SIN}$.
4. Pick and expand $\beta$ most influential entries in $queue_{SIN}$. Let $\boldsymbol{top\beta SIN}$ be the $\beta$ entries in $queue_{SIN}$ with maximum influence.
5. Let $\boldsymbol{C_O}$ be the set of every $O_i$ in $queue_O$ that affects at least one entry in $top\beta SIN$.
6. **for** every $O_i$ in $C_O$
   - 6.1 Compute the largest probability that $O_i$ can be pruned by expanding some $S_j$ in $queue_{SOUT}$.
   - 6.2 If this probability is above $\gamma$ (for a given constant $\gamma \in [0, 1]$), expand the corresponding $S_j$.
7. Let $impO_i$ be the most important $O_i \in C_O$.
8. While there exists an $S_j$ in $queue_{SIN}$ or $queue_{SOUT}$ whose MBR contains the MBR of $impO_i$, expand that $S_j$.
9. If there exists an $S_j$ in $queue_{SIN}$ or $queue_{SOUT}$ more important than $impO_i$, expand that $S_j$.
10. If $impO_i$ is an index entry which affects multiple in-queue entries from $S$, expand $O_i$.
11. **goto** Step 2.

---

Figure 9: Computation of top-$t$ most influential sites.

will maximally increase the accuracy of the most number of estimations of entries in $top\beta SIN$.

Step 7 picks such an entry $O_i$, which is *the most important*. Here the *importance* of $O_i$ is defined as:

$$|O_i| * (\#\text{ affected entries in } top\beta SIN) * area(O_i)$$

Intuitively,

- If the total weight of objects in $O_i$ is large, $O_i$ is important. To know more precisely how this large weight is actually distributed is beneficial.
- If $O_i$ affects many entries in $top\beta SIN$, $O_i$ is important, for its total weight contributes to *max-Influence* of all those entries.
- If $O_i$ has a large area, it is important. Suppose some $O_i'$ has a tiny area. Even though it may have a large total weight of objects and it may affect many entries in $top\beta SIN$, expanding it may not help for every child entry may affect all the entries that $O_i'$ currently affects.

Step 8 shows the case when there exists an entry $S_j$ that spatially contains the most important $O_i$. In this case, however deep we expand $O_i$, all of its descendant entries will affect $S_j$. So we make sure such an $S_j$ does not exist.

Now we want to give a chance to entries in $queue_{SOUT}$ or $queue_{SIN}$, which are affected by some entry in $C_O$. Step 9 says we should expand such an $S_j$ if it is *more important than* $O_i$. Intuitively, if $S_j$ is affected by many entries in $C_O$, and if $S_j$ has a large area, it is more important. Our policy is, $S_j$ is more important than $O_i$, if

$$area(S_j) * (\#\text{ entries in } C_O \text{ that affect } S_j) >$$

$$\alpha * area(O_i) * (\#\text{ entries in } top\beta SIN \text{ affected by } O_i)$$

Here we use some chosen constant $\alpha$ to control how likely to expand some entry $S_j$, which is affected by some entries in $C_O$. A larger $\alpha$ means $S_j$ is less likely to be expanded.

### 4.3 The Probability of Pruning An Entry in $queue_O$ by Expanding An Entry in $queue_{SOUT}$

In Step 6 of Algorithm TopInfluentialSites, we expand an entry $S_j$ such that by expanding it, the probability of pruning the previously picked $O_i$ is above a threshold $\gamma$. Here we consider the probability that there exists a child entry of $S_j$, say *child*, that satisfies: $minExistDNN_{child}(O_i) < minDist(O_i, S_k)$, where $S_k$ is the entry in $queue_{SIN}$ with the minimum $minDist$ from $O_i$.

In Figure 10(a), even though $S_j$ cannot be used to prune $O_i$, it is very likely that some child entry in $S_j$ has a much smaller $minExistDNN$ with $O_i$, and

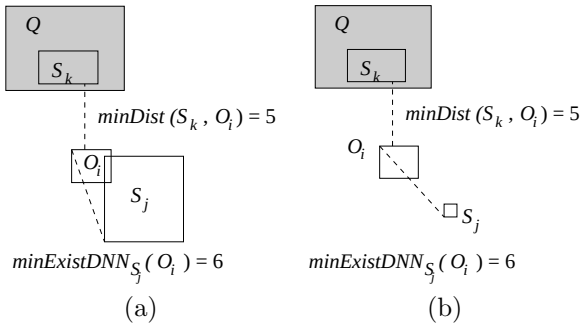


Figure 10: The probability of pruning $O_i$ by expanding $S_j$ is bigger in (a) than (b).

therefore the probability is high. On the other hand, Figure 10(b) shows a case with a small probability. Intuitively, in the latter case $S_j$ is small, and thus expanding it to its child entries may not reduce the $minExistDNN$ very much.

Our goal in this section is to mathematically define this probability. To do so, we first introduce a new metric called $minMinExistDNN$.

**Definition 3** *Given MBRs $O_i$ and $S_j$, the smallest possible minExistDNN of $O_i$ with regards to any MBR contained in $S_j$, is* $\mathbf{minMinExistDNN_{S_j}(O_i)} =$

$$\min_{rect \subseteq S_j} \{minExistDNN_{rect}(O_i)\}$$

We already know that $minDist(O_i, S_k) \leq minExistDNN_{S_j}(O_i)$, for otherwise $O_i$ should have already been pruned by $S_j$. Definition 3 implies that if $minDist(O_i, S_k) \leq minMinExistDNN_{S_j}(O_i)$, the probability of having $O_i$ pruned by expanding $S_j$ is 0. So let's assume that $minDist(O_i, S_k)$ is between $minMinExistDNN_{S_j}(O_i)$ and $minExistDNN_{S_j}(O_i)$. The probability of having $O_i$ pruned by expanding $S_j$ is formally defined as

$$\frac{minDist(O_i, S_k) - minMinExistDNN_{S_j}(O_i)}{minExistDNN_{S_j}(O_i) - minMinExistDNN_{S_j}(O_i)}$$

To see that this is a reasonable probability, we point out that if $minDist(O_i, S_k)$ changes from $minMinExistDNN_{S_j}(O_i)$ to $minExistDNN_{S_j}(O_i)$, the probability increases from 0 to 1, which matches our intuition.

The algorithm to compute $minMinExistDNN_{S_j}(O_i)$ is: pick the location $l$ in $S_j$ which has minimum distance to the center of $O_i$, and return the maximum distance from $l$ to a corner of $O_i$. Due to space limitations the proof is omitted. An illustration of this new metric appears in Figure 11.

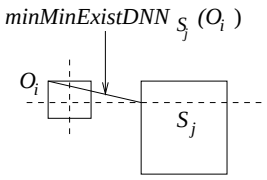


Figure 11: Illustration of $minMinExistDNN$.

### 4.4 The General Case when An MBR in $S$ May Intersect with $Q$

So far we have assumed that no index entry's MBR, in the R-tree $S$ of sites, intersects with the query region $Q$. This unrealistic assumption is only used to simplify the presentation of our algorithm. We can easily extend our algorithm to handle the general case without this assumption. The changes that need to be made are as follows.

First, $queue_{SIN}$ also contains index entries from $S$ whose MBRs intersect with $Q$. Second, to compute the $minDist$ between some $O_i$ to an entry $S_j$ in $queue_{SIN}$, use the intersection part between $S_j$'s MBR and $Q$. Finally, suppose an edge of some $S_j$, whose MBR intersects with $Q$, is completely outside $Q$. This edge should be stored as an entry in $queue_{SOUT}$ so as to increase the chance of pruning entries in $queue_O$.

## 5 Performance

### 5.1 Experimental Setup

We compare our algorithm *TopInfluentialSites* with the Voronoi based method extended from [SRAE01]. In particular, we implement the optimized version of the Voronoi based method, such that if a Voronoi cell fully contains a subtree, we do not expand that subtree. Both algorithms store sites and objects in R-trees. The fan-out of each node in an R-tree is 40% of the capacity. The node capacity varies from 1KB to 4KB in different experiments. We also utilize an LRU buffer with capacity varying from 64 disk pages to 512 disk pages, for each R-tree. All the data structures and algorithms are coded using Java, and ran on a PC with 2.66-GHz Pentium 4 processor.

In the experiments, We use real data of the Digital Chart of the World taken from the R-tree-Portal [The03]. One dataset consists of 24,493 populated places in North America, and another dataset contains 9,203 cultural landmarks in North America. These two data sets form two combinations of sites and objects. One takes the cultural landmarks (9,203 points) as the sites data and the populated places (24,493 points) as the objects data. The ratio between # sites and # objects is 1 : 2.5. Another combination is the reverse case, where the ratio is 2.5 : 1. Our experiments are performed on both combinations of datasets. The weight of an object is 1, i.e., the influence of a site is the number of its RNNs.

### 5.2 Selection of Parameters

In our experiments, we choose the value of three parameters $\alpha$, $\beta$ and $\gamma$ experimentally. Due to space limitations, the experiment charts are omitted, and the results are reported as follows.

We choose $t = 4$ and $\beta = t$. In fact, our preliminary results show that with the increase of $t$, our algorithm is not affected much.

Recall that $\gamma$ determines whether an entry in $queue_O$ should be expanded (Section 4.3). Intuitively, if $\gamma$ is small, we may expand many site MBRs, while if $\gamma$ is large, we may not expand any of the site MBRs. Our experimental results show that when $\gamma$ is between 0.5 to 0.7, the performances differ a little. In the following experiments, we choose $\gamma = 0.5$.

Finally, $\alpha$ determines whether $S_j$ is more important than $O_i$ in step 9 of the algorithm *TopInfluentialSites* (Figure 9). Our preliminary results show that in order to make $S_j$ and $O_i$ comparable, the value of $\alpha$ depends on the ratio between # sites and # objects. In our algorithm, we choose $\alpha$ to be the ratio.

### 5.3 Experimental Results

In our experiments, we compare our method and the Voronoi based method extensively by varying the query size, the page size and the buffer size. In this section, We denote the algorithm *TopInfluentialSites* as **TIS** and the Voronoi based method as **Voronoi**. Except otherwise stated, the page size of an R-tree is 1KB and each R-tree uses an LRU buffer with capacity of 128 pages. Our results are the average of ten runs. Notice that all figures comparing Voronoi and TIS used *logarithmic* scales.

We first compare the number of disk page accesses by varying the query size, when $\#sites : \#objects = 1 : 2.5$. Figure 12(a) shows the comparison of the total disk I/Os of both site R-tree and object R-tree, and in Figure 12(b), we compare them separately. Notations of **Voronoi (obj)**, **Voronoi (sites)**, **TIS (obj)** and **TIS (site)** represent the disk I/Os on the object R-tree and site R-tree using Voronoi or TIS, respectively.

TIS greatly outperforms Voronoi on all query sizes. As shown in Figure 12(a), when the query size is small (0.001% and 0.01%), Voronoi is barely comparable to TIS. With the increase of the query size, the disk I/Os of Voronoi increases almost exponentially. This is because, for every site in the query range, Voronoi based method performs nearest neighbor and range queries on the site R-tree to compute its Voronoi cell, and then performs range query (using the Voronoi cell) on the object R-tree. When the query size increases, the number of sites inside the query increases dramatically, therefore, the number of range queries performed on both R-trees increases as well. Figure 12(b) reveals how the total disk I/Os are divided among the site R-tree and the object R-tree. In both methods, the

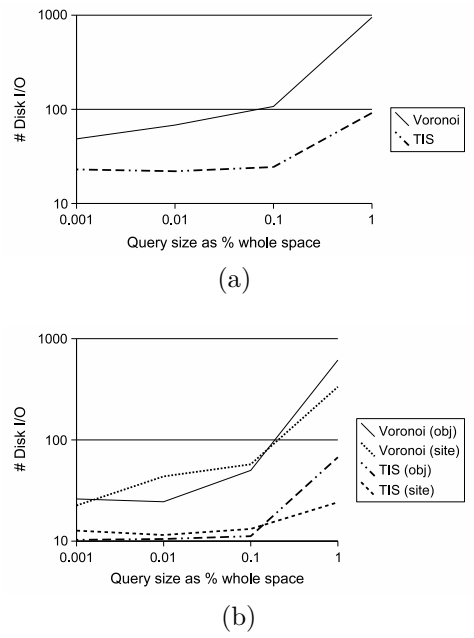


Figure 12: Comparing the number of disk I/Os when $\#sites : \#objects = 1 : 2.5$.

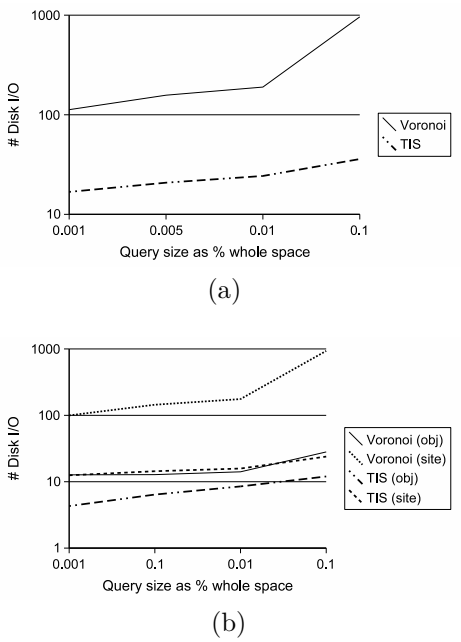


Figure 13: Comparing the number of disk I/Os when $\#sites : \#objects = 2.5 : 1$.

number of disk I/Os on the site R-tree is comparable to that on the object R-tree.

We also perform range queries by varying the query size, when $\#sites : \#data = 2.5 : 1$, as shown in Figure 13. Similar to the previous experiment charts, Figure 13(a) shows the comparison of total disk I/Os and Figure 13(b) shows the I/Os on the site R-tree and the object R-tree separately.

Again TIS outperforms Voronoi in all query sizes. In this case, the number of sites is much larger than the number of objects, and queries with small sizes contain many sites. Therefore, compared to the case of site-object ratio being 1 : 2.5, the number of disk I/Os of small queries are much larger. Especially, as shown in Figure 13(b), the disk I/Os on the site R-tree dominates the total disk I/Os in the Voronoi method.

In both cases of Figure 12 and Figure 13, our method (TIS) shows stability to the data sets and the query size, because (1) we examine both trees in only one pass, (2) we visit a node from either trees only if necessary.

In the remaining experiments, we use $\#sites : \#objects = 1 : 2.5$.

In Figure 14, the page size varies from 1KB to 4KB, and the buffer size (128 pages) is unchanged. The size of range query is 1% of the whole space. When the page size is 1KB, the number of entries in each node is around 45. With the increase of page size, the number of tree nodes decreases. Therefore, the number of disk I/Os in both methods drops dramatically. Still, TIS is better than Voronoi by an order of magnitude in all cases.

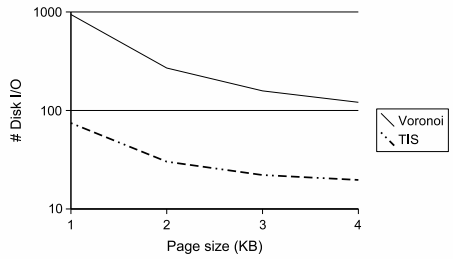


Figure 14: Varying page size.

In Figure 15, the performances are compared under various sizes of the LRU buffer. The buffer size is changed from 64 pages to 512 pages. With the page size being 1KB, the total number of nodes of the object R-tree and that of the site R-tree are 1010 and 380, respectively. It is intuitive that when the size of the LRU buffer increases, the number of disk I/Os decreases. The number of disk I/Os drop dramatically when the buffer size increases to 512 pages, since most pages could be accommodated in the buffer. Again, TIS outperforms Voronoi in all cases, while the difference narrows with the increase of the buffer size.

Finally, with regard to the algorithm *TopInfluentialSites*, we also compare our strategy of expanding the entries in the queues with the naive round robin strategy. In Figure 16, we still denote our strategy as **TIS** and the round robin strategy as **TIS RR**.

In Figure 16, we perform queries with different sizes using both strategy, and our carefully designed strategy outperforms the naive one in all cases. With the

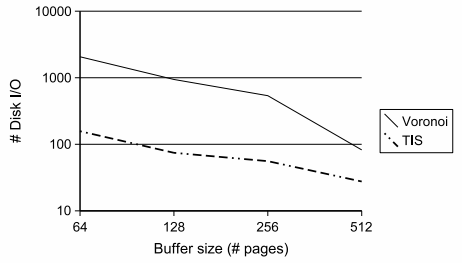


Figure 15: Varying buffer size.

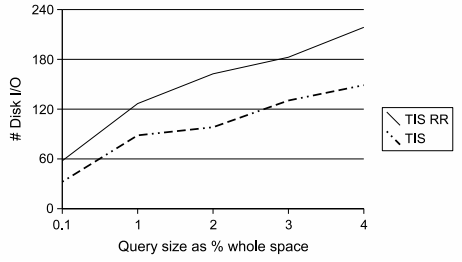


Figure 16: Comparison with the round-robin expansion strategy.

increase of the query size, the number of entries in three queues increases, and choices of picking an entry to expand become more important as we do not want to expand all entries. Therefore, the difference of two strategies increases.

## 6 Conclusions

This paper addressed the new problem of finding top-$t$ most influential spatial sites. An existing work on finding RNNs [SRAE01] can be extended to solve this problem. However, the approach is inefficient as it needs to browse both the site R-tree and the object R-tree many times. We proposed an algorithm called TopInfluentialSites, which solved the problem by browsing both R-trees once. The algorithm possesses pruning ability based on a novel metric called *minExistDNN*. Experimental results on real datasets have revealed more than an order of magnitude improvement on the query performance.

## References

[BJKv02] R. Benetis, C. S. Jensen, G. Karčiauskas, and S. Šaltenis. Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects. In *Proc. of Int. Database Engineering & Applications Symposium (IDEAS)*, pages 44–53, 2002.

[BKOS97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Sprinter Verlag, 1997.

[FSAE01] H. Ferhatosmanoglu, I. Stanoia, D. Agrawal, and A El Abbadi. Constrained Nearest Neighbor Queries. In *Proc. of Symposium on Spatial and Temporal Databases (SSTD)*, pages 257–278, 2001.

[KM00] F. Korn and S. Muthukrishnan. Influence Sets Based on Reverse Nearest Neighbor Queries. In *ACM SIGMOD*, pages 201–212, 2000.

[KMS02] F. Korn, S. Muthukrishnan, and D. Srivastava. Reverse Nearest Neighbor Aggregates Over Data Streams. In *VLDB*, pages 814–825, 2002.

[LNY03] K.-I. Lin, M. Nolen, and C. Yang. Applying Bulk Insertion Techniques for Dynamic Reverse Nearest Neighbor Problems. In *Proc. of Int. Database Engineering & Applications Symposium (IDEAS)*, pages 290–297, 2003.

[MVZ02] A. Maheshwari, J. Vahrenhold, and N. Zeh. On Reverse Nearest Neighbor Queries. In *Proc. of Canadian Conf. on Computational Geometry (CCCG)*, pages 128–132, 2002.

[PKZT01] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP Operations in Spatial Data Warehouses. In *Proc. of Symposium on Spatial and Temporal Databases (SSTD)*, pages 443–459, 2001.

[RKV95] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest Neighbor Queries. In *ACM SIGMOD*, pages 71–79, 1995.

[SAE00] I. Stanoi, D. Agrawal, and A. El Abbadi. Reverse Nearest Neighbor Queries for Dynamic Databases. In *ACM/SIGMOD Int. Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, pages 44–53, 2000.

[SFT03] A. Singh, H. Ferhatosmanoglu, and A. S. Tosun. High dimensional reverse nearest neighbor queries. In *Proc. of Int. Conf. on Information and Knowledge Management (CIKM)*, pages 91–98, 2003.

[Smi97] M. Smid. Closest Point Problems in Computational Geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook on Computational Geometry*. Elsevier Science Publishing, 1997.

[SRAE01] I. Stanoi, M. Riedewald, D. Agrawal, and A. El Abbadi. Discovery of Influence Sets in Frequently Updated Databases. In *VLDB*, pages 99–108, 2001.

[The03] Yannis Theodoridis. The R-tree-portal. *http://www.rtreeportal. org*, 2003.

[TPL04] Y. Tao, D. Papadias, and X. Lian. Reverse kNN Search in Arbitrary Dimensionality. In *VLDB*, pages 744–755, 2004.

[YL01] C. Yang and K.-I. Lin. An index structure for efficient reverse nearest neighbor queries. In *ICDE*, pages 485–492, 2001.

[YPMT05] M. L. Yiu, D. Papadias, N. Mamoulis, and Y. Tao. Reverse Nearest Neighbors in Large Graphs. In *ICDE*, pages 186–187, 2005.