Reverse Nearest Neighbor Heat Maps: A Tool for Influence Exploration

Yu Sun*, Rui Zhang§*, Andy Yuan Xue*, Jianzhong Qi*, Xiaoyong Du†
*Department of Computing and Information Systems, University of Melbourne
†Renmin University of China and Key Laboratory of Data Engineering and Knowledge Engineering, MOE, China
Email: *{sun.y, rui.zhang, andy.xue, jianzhong.qi}@unimelb.edu.au †duyong@ruc.edu.cn

***Abstract*—We study the problem of constructing a reverse nearest neighbor (RNN) heat map by finding the RNN set of every point in a two-dimensional space. Based on the RNN set of a point, we obtain a quantitative *influence* (i.e., *heat*) for the point. The heat map provides a global view on the influence distribution in the space, and hence supports exploratory analyses in many applications such as marketing and resource management. To construct such a heat map, we first reduce it to a problem called *Region Coloring* (RC), which divides the space into disjoint *regions* within which all the points have the same RNN set. We then propose a novel algorithm named CREST that efficiently solves the RC problem by labeling each region with the heat value of its containing points. In CREST, we propose innovative techniques to avoid processing expensive RNN queries and greatly reduce the number of region labeling operations. We perform detailed analyses on the complexity of CREST and lower bounds of the RC problem, and prove that CREST is asymptotically optimal in the worst case. Extensive experiments with both real and synthetic data sets demonstrate that CREST outperforms alternative algorithms by several orders of magnitude.**

I. Introduction

In market analysis, urban design, and facility placement, we often need to select a suitable location for new facilities such as a warehouse or a hospital. Emerging event-based social networks such as Meetup and Whova also need to select an appropriate location suitable for the event-participant arrangement. These problems are called the *location selection* problem, which is usually a multi-criteria decision making process involving various quantitative and qualitative factors. A quantitative factor usually considered is the *influence* of the location, which is commonly measured by the *reverse nearest neighbor* (RNN) set of the location [12], [22], [26]. Given two sets of points $\mathcal{O}$ and $\mathcal{F}$, the RNN set of a location p is a subset of $\mathcal{O}$ that are closest to p among all the points in $\mathcal{F}$. There are many ways to measure the influence of p by the RNN set. Straightforward measures consider only the size or total weight of the set [6], [26], [31]. Other measures consider various attributes of the data points in $\mathcal{O}$ and $\mathcal{F}$, such as the capacity constraint [16], [22], social relationship [19], [29], etc. While we can model the quantitative factors precisely by numbers, we can not do the same to many qualitative factors such as the area safety, demographic composition and convenience of public transportation. Some factors in decision-making are also vague and imprecise, which are subject to decision maker's judgments. To assist decision making based on quantitative measures while still allowing subjective judgments based on qualitative measures and other factors, we introduce the RNN heat map, which shows the influence (quantitative measures) of every point in the space. Comparing to existing studies [8], [10], [22], [26], [31] which give only the points or regions with the highest influence, the RNN heat map enables exploring the influence of the whole space while considering qualitative factors at any instant during the exploration. Consider a scenario where RNN heat maps are used to assist selecting locations of self-pickup and drop-off service points for courier companies. Let $\mathcal{O}$ be the potential clients and $\mathcal{F}$ be the existing service points. For simplicity, let the size of the RNN set measure the influence, i.e., *heat* (although any other functions related to the RNN set can be used). Fig. 1(a) shows such an RNN heat map for the New York City, whose satellite image is shown in Fig. 1(b). The darker regions indicate higher heat values. Such a heat map will allow the exploration of influential regions while considering qualitative factors as discussed above. Note that regions with high influence values do not necessarily correspond to regions of high client density because we need to consider the competition from existing facilities. For example in Fig. 2, the upper left corner has the highest client density, but the most influential and the 4^{th} influential regions are in the middle, denoted by the two gray rectangles (the 2^{nd} and the 3^{rd} most influential regions are also in the middle near these two but too small to be visible). Without the RNN heat map, it is very difficult or impossible to explore all these different choices and make well-informed decisions.

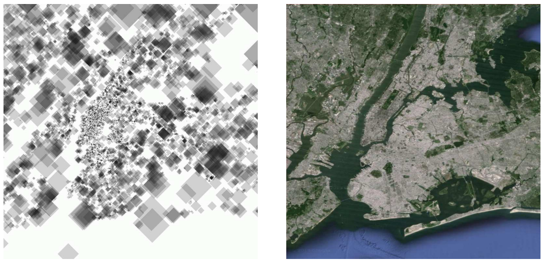

(a) RNN heat map (b) Satellite map

Fig. 1. RNN Heat Map of New York City

To construct such a heat map, we need to obtain the influence value of every point in the space. We call such a problem the *RNN heat map* (RNNHM) problem:

Definition 1 (RNN Heat Map Problem): Given two sets of points $\mathcal{O}$ and $\mathcal{F}$ and a distance metric in a two-dimensional space, the RNN set of a point q ($q \notin \mathcal{F}$) is a subset of $\mathcal{O}$ that have q as their nearest neighbor comparing with other points in $\mathcal{F}$. Given any influence measure, which is a real-valued function on the RNN set, associate each point in the space with its influence value, i.e., the heat value.

Since the number of points in the space is infinite, to solve the

[§] Corresponding author.

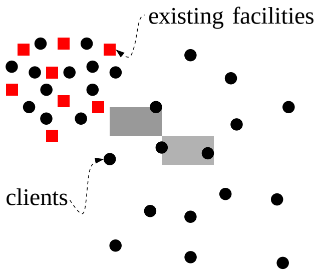
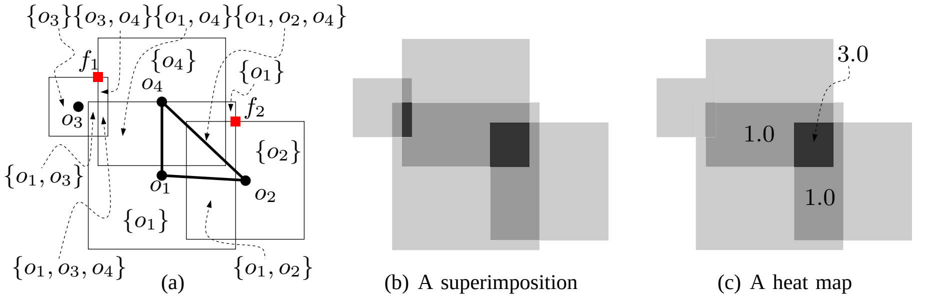


Fig. 2. Client density



(a) (b) A superimposition (c) A heat map

Fig. 3. An example of the RC problem

RNNHM problem, we first reduce it to a problem called *Region Coloring* (RC), which divides the space into disjoint *regions*, within which all the points have the same RNN set (detailed in Section III). We use Fig. 3(a) to illustrate the RC problem with L_∞. For simplicity and ease of presentation, we will first discuss how to obtain such regions and compute their influence values with the L_∞ metric, and then extend the techniques to L_1 and L_2 metrics. In Fig. 3(a), let $\mathcal{O} = \{o_1, o_2, o_3, o_4\}$, represented by the black dots, and $\mathcal{F} = \{f_1, f_2\}$, represented by the small red squares. For each point o in $\mathcal{O}$, we draw a "circle" called the *NN-circle* with o being the center and the distance to o's nearest neighbor (NN) being the radius, which is a square with the L_∞ metric. The NN-circles partition the space into separate regions. It can be proved that all the points in such a region have the same RNN set. The RC problem is to obtain the influence of each such region in the space.

Note that if we measure the influence simply by the size of the RNN set, we can build the heat map by a *superimposition* of the NN-circles, i.e., overlap/overlay of translucent NN-circles as shown in Fig. 3(b). A darker region suggests more NN-circles overlapping there and hence a higher influence. However, for a more generic influence measure than the size (or a weighted sum of the RNNs), the heat map can not be achieved by such a simple superimposition. For example, consider a taxi-sharing scenario [14] where the heat map assists taxi drivers to decide the next pick-up locations. In Fig. 3(a), let $\mathcal{O}$ be potential passengers, e.g., users of taxi booking apps, and $\mathcal{F}$ be taxis. Assume that taxi drivers make more profits when taking together multiple passengers whose destinations are close, say within one kilometer. Let the data points o_1, o_2, and o_4 connected by an edge denote such passengers. Under such setting, the influence of a location becomes the number of connected passengers in the RNN set. We build the heat map as shown in Fig. 3(c). We can see that there is only one darkest region, which has an influence value of 3.0 since its RNN set is $\{o_1, o_2, o_4\}$ and there are three edges connecting o_1, o_2 and o_4. In comparison, the superimposition as shown in Fig. 3(b) creates two darkest regions, both have an influence value of 3.0, one with the RNN set $\{o_1, o_3, o_4\}$ and the other $\{o_1, o_2, o_4\}$. Under the measure that favors connected data points, the RNN set $\{o_1, o_3, o_4\}$ only has an influence value of 1.0, which is not a good choice for picking up passengers. Another example is that in the previous courier company scenario, all the service points have a capacity limit (e.g., the storage space). Taking these attributes into account, the influence of a location will depend not only on the size of the RNN set but also on its serving capacity[1]. The superimposition will not be able to handle such influence measures.

Besides not being able to compute the RNN heat map for generic influence measures, a superimposition also cannot support interactive post-processing operations such as selectively showing regions with heat values above a threshold or regions having the top-k heat values, whereas these operations can be easily applied as post-processing of our proposed techniques, which aim to obtain the RNN set of every region in the space.

In this paper, we investigate algorithms to efficiently solve the RNN heat map problem. In some applications such as taxi-sharing, the heat map may change as clients move around and need to be recomputed frequently. Therefore, an efficient algorithm to the RNNHM problem is crucial. A straightforward approach such as employing a grid to divide the space and then using the cells to fit the regions has difficulties in finding the right granularity and suffers from low efficiency. When the influence measure involves a large amount of attributes such as the capacities of taxis and connections of clients, it can also be very expensive to compute [22]. To overcome these challenges, we propose an innovative algorithm named CREST (**C**onstructing **R**NN h**E**at map with the **S**weep line s**T**rategy) which efficiently solves the RNNHM problem. Through a detailed analysis, we prove that CREST is asymptotically optimal in the worst case. CREST is also generic in the sense that it applies to any influence measure computable from RNN sets and can easily support interactive post-processing operations as described above. The main contributions of this paper are summarized as follows.

- We propose the RNN heat map problem, which computes a heat map showing the distribution of RNN-based scores to support effective exploratory analyses.
- We propose an innovative algorithm named CREST which efficiently solves the RNN heat map problem. The algorithm utilizes two novel techniques to respectively avoid processing any RNN queries and greatly reduce the times of influence computation.
- We carefully analyze the complexity of CREST and lower bounds of the RC problem, and prove that CREST is asymptotically optimal in the worst case.
- We also conduct extensive experiments with both real and synthetic data sets. The results confirm the superiority of CREST by showing that CREST outperforms alternative algorithms by several orders of magnitude.

The remainder of this paper is organized as follows. Section II reviews related work. Section III formalizes the problem. Section IV discusses a baseline algorithm. Section V describes the CREST algorithm. Section VI analyzes the complexity. Section VII extends CREST to other settings. Section VIII shows the experiments and Section IX concludes the paper.

[1] The influence of a location p is computed by $\sum_{f \in \mathcal{F} \cup \{p\}} \min\{c(f), |\mathcal{R}(f)|\}$, where $c(f)$ is the capacity and $\mathcal{R}(f)$ the RNN set of f [22].

II. Related Work

RNN Query. The RNN query is introduced by Korn et al. [12]. Yang et al. [28] proposed the Rdnn-tree (a variant of R-tree) to process the RNN query. Maheshwari et al. [15] present a data structure for answering the monochromatic RNN query by utilizing a persistent search tree [18]. The structure first obtains the NN-circles enclosing a query point in the x dimension and then among these retrieved NN-circles locates the face (region) enclosing the point in the y dimension. These algorithms focus on computing the RNN set of a single query point. None of them directly applies to the RNNHM problem. In the RNNHM problem, the aim is to compute the RNN set for every point in the space all at the same time, and the challenge is to avoid the expensive RNN computation. The All Nearest Neighbor (ANN) [7] operation takes as input two discrete and finite sets of points and computes for each point in the first set the NN in the second set. For the RNNHM problem, however, we need to obtain RNN sets for essentially infinite points in a continuous space. Therefore, the techniques for ANN do not apply.

Influence Measures based on RNN Sets. Various influence measures based on the RNN set have been studied. Korn et al. [12] propose to use the size (or sum of weights) of RNN sets as the influence value. To find the optimal points whose RNN sets are of the maximum size (influence), Cabello et al. [6] propose the maximization problem MaxCov and they solve the problem by finding the depth of an arrangement of disks. Wong et al. [26] solve MaxCov by the devised MaxOverlap algorithm. Huang et al. [11] and Xia et al. [27] investigate finding such points in a given set. Sun et al. [21], [22], [23] additionally consider the capacity constraints of such points and study how to achieve a global influence maximization instead of a local maximization. Qi et al. [17] define the influence based on the average distance between a point and its RNNs. As RNNHM applies to a general measure, the RNNHM problem can be viewed as a generalized version of the above problems and therefore the solution of RNNHM can be adapted to solve these problems. However, their solutions do not apply to RNNHM, since the special properties exploited in these problems do not present in RNNHM.

RNN Variations. RNNHM is a variant of the RNN query. There are also many other studies on variations of the RNN query. For instance, Lu et al. [13] investigate reverse spatial and textual nearest neighbor queries, in which both location and textual descriptions are considered in the distance metric. Similarly, Sun et al. [24] consider temporal aggregates of location-based social network check-ins in the distance metric. Zhang et al. [30] design indexes utilizing modern memory hierarchies to speed up such query processing. Ali et al. [1] study approaches to continuous retrieval of the query objects. She et al. [19], [20] devise algorithms to arrange social events to proper users using RNN sets. These problems are quite different from RNNHM and the proposed algorithms cannot be adapted to solve the RNNHM problem.

Sweep Line Strategy. The sweep line strategy is a quite generic approach to handling geometric objects. The Bentley–Ottmann (BO) algorithm uses this strategy to compute intersections of line segments [4] or rectangles [5]. The BO algorithm and the proposed CREST algorithm compute very different problems and are different in many aspects. i) BO computes only pairwise intersections of line segments or rectangles, while CREST computes the overlaps and relative complements of multiple circles, squares, and axis-aligned line segments, which are much more challenging. ii) In order to efficiently compute the RNN sets, besides the line status, CREST need to memorize the RNN sets of previous events. This requires a delicate design to minimize the overhead and achieve optimal performance. BO does not have such optimization.

III. Problem Formulation

We first introduce basic concepts in Section III-A and then reduce RNNHM to the Region Coloring (RC) problem in Section III-B. Frequently used symbols are listed in Table I.

TABLE I
Frequently Used Symbols

Symbol	**Meaning**
$\mathcal{O}$	the set of clients
$\mathcal{F}$	the set of facilities
n	the number of data points in $\mathcal{O}$
$\mathcal{C}(o_i)$	the NN-circle of $o_i \in \mathcal{O}$
$\underline{x_i}$ (resp. $\underline{y_i}$)	the left (resp. lower) side of $\mathcal{C}(o_i)$
$\overline{x_i}$ (resp. $\overline{y_i}$)	the right (resp. upper) side of $\mathcal{C}(o_i)$
e_l	the l-th event
x_l	the x-coordinate of e_l
$I(l)$	the line status between e_{l-1} and e_l
y_t	the t-th element in a line status
$\langle y_{t-1}, y_t \rangle$	two consecutive elements in a line status
r_l^t	the rectangle $[x_{l-1}, x_l, y_{t-1}, y_t]$
$\mathcal{R}(\cdot)$	the RNN set of an object

A. Preliminaries

We consider two types of RNN queries: the bichromatic and monochromatic RNN queries. In the former type, the data points and their NNs belong to two different sets $\mathcal{O}$ and $\mathcal{F}$. In the latter type, they are from the same set, i.e., $\mathcal{O} = \mathcal{F}$. Let $d(p, q)$ be the distance between two points p and q. We consider three different distance metrics: L_∞, L_1, and L_2. We start with solving the bichromatic RNNs with L_∞ metric because the bichromatic type is generic and L_∞ is simpler.

RNN Query. In bichromatic RNNs, we are given two sets $\mathcal{O}$ and $\mathcal{F}$. The set $\mathcal{O}$ can be considered as (the locations of) clients while $\mathcal{F}$ as (the locations of) facilities. The clients find their NNs from the facility set. The RNN set of a point f in $\mathcal{F}$, denoted by $\mathcal{R}(f)$, consists of the points in $\mathcal{O}$ that have f as their NN, i.e.,

$$\mathcal{R}(f) = \{o \in \mathcal{O} \mid \forall f' \in \mathcal{F} : d(o, f) \leq d(o, f')\}.$$

For a point q not in $\mathcal{F}$, we obtain its RNN set $\mathcal{R}(q)$ by adding q into the facility set $\mathcal{F}$ and computing $\mathcal{R}(q)$ as above.

Nearest Neighbor Circle (NN-circle). An *NN-circle* of a point o, denoted by $\mathcal{C}(o)$, is a circle with o being the center and the distance from o to its NN being the radius. With the L_∞ distance metric, the distance between two points is the maximum difference between their coordinates among all dimensions, i.e., $d(p, q) = \max\{|p_x - q_x|, |p_y - p_y|\}$ in a two-dimensional space, where the subscripts denote the coordinates

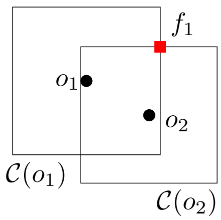


Fig. 4. NN-circles

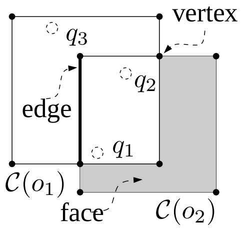


Fig. 5. Arrangement

in the x and y dimensions, respectively. Hence, NN-circles are of a square shape. (With L_1 and L_2, the NN-circles are of diamond and circular shapes, respectively.) For example, in Fig. 4, set $\mathcal{O}$ consists of two points o_1 and o_2. Set $\mathcal{F}$ consists of one point f_1. The NNs of o_1 and o_2 are both f_1. The NN-circles $\mathcal{C}(o_1)$ and $\mathcal{C}(o_2)$ are the two squares.

B. Problem Reduction

Our goal is to draw the heat map of a given space based on the RNN sets of the points. In a continuous space, the number of points is infinite, which makes finding the RNN set of every point infeasible. To overcome this, we reduce RNNHM to an equivalent problem Region Coloring (RC), which divides the space into regions and "colors" (i.e., associates) every *region* with a "heat" (i.e., influence value). We divide the space using the NN-circles as follows. The *arrangement* (i.e., layout) of the NN-circles, as illustrated in Fig. 5, forms a planar graph, which also induces a *subdivision* of the space. We use the notions in planar graphs such as *vertices*, *edges* and *faces* (as illustrated in Fig. 5) in the arrangement directly. In the arrangement, each face represents a unique *region*, which is a maximal connected subset of the space that does not contain a vertex or an edge (e.g., the gray region in Fig. 5). In each region, all the points have the same RNN set. If two points of a region have different RNN sets, there must exist at least one NN-circle that one point lies inside but the other does not; this means one side of the NN-circle must *cut* the region, making it no longer a region by definition. The RNN set of each point in the region consists of the centers of the NN-circles that *enclose* the region. For example, in Fig. 5, points q_1 and q_2 lie in the region enclosed by NN-circles $\mathcal{C}(o_1)$ and $\mathcal{C}(o_2)$, and they have the same RNN set $\{o_1, o_2\}$. For point q_3, its RNN set is $\{o_1\}$, which is different from that of q_1 or q_2. Therefore, q_3 must lie in a different region. Note that the opposite does not hold, i.e., different regions may have the same RNN set. We formalize the above facts with the following proposition.

Proposition 1: The points in the same region of the subdivision formed by the arrangement of the NN-circles have the same RNN set.

For RNNHM, each region can be used to represent all the points it contains. To associate each point with a heat, it suffices to color each region with the heat of the points it contains. Since the influence is computed straightforwardly based on the RNN set, in the following discussion, we do not distinguish the process of outputting the RNN set of a region and the process of computing and outputting the influence value. We will simply use the term "labeling a region" to denote the two processes. Assuming that the NN-circles are already precomputed (there are efficient algorithms to compute and maintain the NN-circles [12]), we define the above region coloring problem as follows.

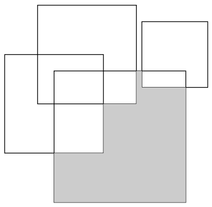


Fig. 6. Bounding edge

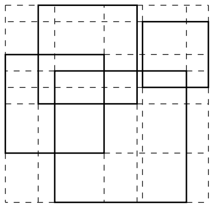


Fig. 7. Side extension

Definition 2 (Region Coloring): Given a set of NN-circles, Region Coloring is to label each region in the arrangement of the NN-circles based on the RNN set of any point contained in the region.

IV. A Baseline Algorithm

A simple approach to the RC problem is to pick a point p inside each region, use a *point enclosure query* to obtain the NN-circles that enclose p, obtain the RNN set and label the region. However, picking a point in each region is an expensive operation. This is because it requires computing an exact representation of each region in the arrangement, which means every edge bounding a region needs to be computed (cf. Fig. 6) and hence has a very high complexity ($\mathcal{O}(n^2 \log n)$ [9]). To avoid such complicated computations, we extend the sides of each NN-circle to let them span across the whole arrangement, as shown in Fig. 7. By doing so we form a grid over the arrangement, where each grid cell can be easily located. We scan the grid cells and compute the RNN set for the centroid of each cell, which solves the RC problem. An alternative way is to use a regular grid where each cell has the same size. However, it is difficult to determine a proper cell size to guarantee that each cell falls in exactly one region unless each point is treated as a cell, which again is impossible to compute. To efficiently compute the RNN set of a point, instead of checking each NN-circle to test whether it encloses a certain point, we build an index that supports point enclosure queries for the NN-circles. We use the *S-tree* [25] for ease of analysis, although other spatial indexes such as the R-tree may be used.

Algorithm Complexity. Let $n = |\mathcal{O}|$ denote the number of NN-circles, and m denote the number of grid cells. There are at most $2n$ extended sides vertically or horizontally, thus $m = O((2n)^2) = O(n^2)$. To obtain the grid cells, it takes $O(2 \times 2n \log 2n) = O(n \log n)$ time to sort the sides. It then takes $O(n \log^2 n)$ time to build an S-tree index and $O(\log n + \alpha)$ time to process a point enclosure query [25], where α is the number of NN-circles returned. Let λ be the maximum size of the RNN sets in the arrangement. The time complexity of the baseline algorithm is $O(n \log^2 n + m \log n + m\lambda)$. Since we consider a general influence measure, which can be any function with any computational cost, in the analysis we only count the number of times of influence computation, i.e., m in the above complexity. We further derive a bound for m as follows. Let r be the number of regions formed by n NN-circles. It can be proved by *Euler characteristic* that r is between $\Theta(n)$ and $\Theta(n^2)$. In particular, when the n NN-circles do not intersect with each other, $r = n + 1 = \Theta(n)$; when the n NN-circles are placed as shown in Fig. 8, where they all have the same side length n and the i^{th} NN-circle is centered at point (i, i), $r = n^2 - n + 2 = \Theta(n^2)$. Since $r \leq m$ and $m = O(n^2)$, we obtain $\Theta(r) \leq m \leq \Theta(n^2)$.

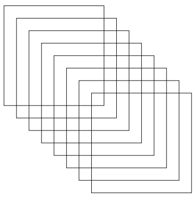


Fig. 8. Worst case

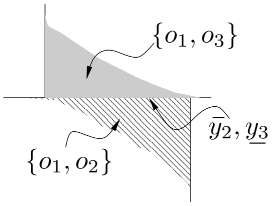


Fig. 9. Set changing

The S-tree index for point enclosure queries occupies the most space, which is $O(n \log^2 n)$ [25]. Thus, the space complexity is $O(n \log^2 n)$. We summarize the above analysis with the following theorem.

Theorem 1: The baseline algorithm for the RC problem stops in $O(n \log^2 n + m \log n + m\lambda)$ time and uses $O(n \log^2 n)$ space, where m is the number of grid cells and λ is the maximum size of the RNN sets.

Limitations of the algorithm. One drawback of the baseline algorithm is that it needs to process point enclosure queries, which is reflected in the $n \log^2 n$ and $m \log n$ terms in the time complexity. Another drawback is that it further divides the regions into multiple grid cells, which means a region in the original arrangement will be labeled multiple times. The number of these grid cells m may increase quadratically with the increase of n (closer to $\Theta(n^2)$). A large m means we need to process a large number of point enclosure queries and label a large number of grid cells, which significantly deteriorates the efficiency. We aim to reduce m (the number of times of region labeling) to the number of regions in the arrangement, which is optimal in the RC problem. Therefore, we have two directions for improvements: (i) to avoid point enclosure queries, and (ii) to reduce the number of times region labeling. We present our CREST algorithm which achieves these two goals in the following section.

V. The CREST Algorithm

We employ the classic *sweep line* strategy [4], [9] (cf. Section II) to avoid forming a large number of cells to be labeled as done by the grid dividing strategy. We let a line sweep from the left to the right of the space, and store information about the NN-circles that are currently *cut* by the sweep line. We call such information the *line status*, and say that an *event* is triggered when the line status changes. As illustrated in Fig. 10, we use the *distinct* vertical sides of the NN-circles as event points (i.e., $x_1, x_2, \ldots, x_9$), and the *sorted* horizontal sides of the NN-circles as the line status. Every pair of adjacent vertical sides and horizontal sides forms a *subregion* to be labeled. We notice that some of these subregions come from the same original region formed by the NN-circles, and hence do not require the RNN set and influence computations repetitively. We use the *change intervals* to avoid labeling such regions multiple times. We avoid the RNN computation with point enclosure queries by utilizing the fact that the RNN set of a region can be obtained efficiently by modifying the RNN sets of the adjacent regions. For example, in Fig. 9, if the RNN set of the lower region is $\{o_1, o_2\}$ and the boundary between the two regions is formed by the upper and lower sides of $\mathcal{C}(o_2)$ and $\mathcal{C}(o_3)$, denoted by $\bar{y}_2$ and $\underline{y_3}$, respectively, then we can immediately obtain the RNN set $\{o_1, o_3\}$ of the upper region by removing o_2 from $\{o_1, o_2\}$ and then adding o_3 to $\{o_1\}$. We call the already-computed RNN sets of adjacent regions the *base sets* and cache them for obtaining the RNN sets of newly swept regions. We also devise techniques to constrain the number of cached base sets. Powered by these techniques, we achieve a highly efficient algorithm to the RNNHM/RC problem, which is proved to be asymptotically optimal in many cases (cf. Section VI). Next we detailed these techniques.

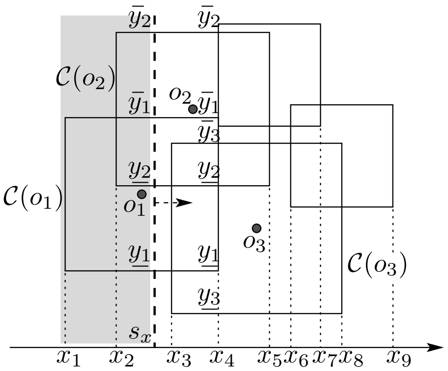


Fig. 10. Example of events and line status

A. Concepts and Notation

Events. Let $\underline{x_i}$ (resp. $\underline{y_i}$) and $\bar{x}_i$ (resp. $\bar{y}_i$) be the x- (resp. y-) coordinates of the left and right (resp. lower and upper) sides of NN-circle $\mathcal{C}(o_i)$, respectively. The *distinct x-coordinates* of the vertical sides (of all the NN-circles) are stored in ascending order in a queue, which is called the *event queue* and denoted by $\mathcal{Q}_e$. The elements in $\mathcal{Q}_e$ are called the *events* or *event points*. For convenience, we refer to the coordinates of the sides simply as sides when the context is clear. We denote the l^{th} event (i.e., the l^{th} ejected element from $\mathcal{Q}_e$) by e_l and the x-coordinate of e_l by x_l. Note the difference between an event coordinate x_i and the side coordinates of $\mathcal{C}(o_i)$ (i.e., $\bar{x}_i$ or $\underline{x_i}$). They may have an equal value, but different semantics.

Line Statuses. Let s_x be the x-coordinate of the sweep line. We say that the line *cuts* NN-circle $\mathcal{C}(o_i)$ if and only if $s_x \in (\underline{x_i}, \overline{x_i}]$ (i.e., it is in the horizontal range of $\mathcal{C}(o_i)$). By definition, the NN-circles that are cut by the line remain the same between two consecutive events (including their positions). Let $\mathcal{C}(o_{l_1}), \mathcal{C}(o_{l_2}), \ldots, \mathcal{C}(o_{l_{n(l)}})$ be the $n(l)$ NN-circles cut by the line when it sweeps from e_{l-1} to e_l. We sort the horizontal sides (not only coordinates) $\underline{y_{l_1}}, \overline{y_{l_1}}, \underline{y_{l_2}}, \overline{y_{l_2}}, \ldots, \underline{y_{l_{n(l)}}}, \overline{y_{l_{n(l)}}}$ of these NN-circles in ascending order (ties are broken arbitrarily), and use the sorted list as the line status between events e_{l-1} and e_l, which is denoted by $\mathcal{I}(l) = \|\underline{y_{l_a}}, \ldots, \overline{y_{l_b}}, \ldots \underline{y_{l_c}}, \ldots, \overline{y_{l_d}}\|, \quad a, b, c, d \in \{1, 2, \ldots, n(l)\}$. For example, in Fig. 10, the current line status is $\mathcal{I}(3) = \|\underline{y_1}, \underline{y_2}, \bar{y}_1, \bar{y}_2\|$. For convenience, we denote by y_i the i^{th} element in the line status, and hence the line status between e_{l-1} and e_l is

$$\mathcal{I}(l) = \|y_1, y_2, \ldots, y_{2n(l)}\|, \;\; y_1 \leq y_2 \leq \ldots \leq y_{2n(l)}.$$

Pair and Subregion. Any two consecutive elements in the line status is termed as a *pair*, which is denoted by $\langle y_{t-1}, y_t \rangle$. We denote by $\langle y_{t-1}, y_t \rangle \in \mathcal{I}(l)$ that the pair $\langle y_{t-1}, y_t \rangle$ comes from the line status $\mathcal{I}(l)$. We denote by $[x, x', y, y']$ a rectangle whose diagonally opposite corners are (x, y) and (x', y') with $x < x'$ and $y \leq y'$. When $y = y'$, $[x, x', y, y']$ is in fact a horizontal line segment. For ease of presentation, we treat it as a special rectangle. We denote by $(x_0, y_0) \in [x, y, x', y']$ that point (x_0, y_0) is in rectangle $[x, y, x', y']$. Here the rectangle is *open* (i.e., $(x_0, y_0) \in [x, y, x', y']$ iff $x_0 \in (x, x')$ and $y_0 \in (y, y')$) and no point is in the special rectangle. When the line sweeps from e_{l-1} to e_l, the x-coordinate x_{l-1} of event e_{l-1} is strictly less than that of e_l. This forms a rectangle $[x_{l-1}, x_l, y_1, y_{2n(l)}]$ between the two events. In this rectangle, each pair $\langle y_{t-1}, y_t\rangle \in \mathcal{I}(l)$ forms a *small* rectangle $[x_{l-1}, x_l, y_{t-1}, y_t]$. The small rectangle has no vertex or edge in it, which makes it a connected subset of a region. We call each small rectangle a *subregion*, and denote by r_l^t the one formed by pair $\langle y_{t-1}, y_t\rangle \in \mathcal{I}(l)$. We denote by $\mathcal{R}(r_l^t)$ the RNN set of the points in subregion r_l^t, or simply by $\mathcal{R}(\langle y_{t-1}, y_t\rangle)$ when the line status is clear.

B. Avoiding Point Enclosure Queries

We obtain the RNN set of each subregion by finding the NN-circles enclosing it. When the line sweeps from e_{l-1} to e_l, the subregions between e_{l-1} and e_l are enclosed by the NN-circles in the x dimension if and only if these NN-circles are cut by the line. Therefore, we only need to check whether these NN-circles enclose the subregions in the y dimension, which can be easily achieved by checking the line status. We use the following lemma to show the RNN set of a pair in the line status. Due to space limitation, we omit the proofs of the lemmas in this section.

Lemma 1: $\forall \langle y_{t-1}, y_t\rangle \in \mathcal{I}(l)$, the RNN set $\mathcal{R}(r_l^t)$ of subregion $r_l^t = [x_{l-1}, x_l, y_{t-1}, y_t]$ is an empty set if $y_{t-1} = y_t$ or a set consists of the centers of the NN-circles that are cut by the line and enclose r_l^t in the y dimension, i.e., $\mathcal{R}(r_l^t)$ is

$$\begin{cases} \varnothing & \text{if } \ y_{t-1} = y_t, \\ \{o_i \mid \underline{x_i} < x_l \le \bar{x}_i \text{ and } \underline{y_i} \le y_{t-1} < y_t \le \bar{y}_i\} & \text{if } \ y_{t-1} \neq y_t. \end{cases}$$

By Lemma 1, we can obtain the RNN set $\mathcal{R}(\langle y_{t-1}, y_t\rangle)$ of a pair as follows. When $y_{t-1} = y_t$, the RNN set is empty. For convenience, we call such pairs *invalid* pairs and the others (with $y_{t-1} < y_t$) *valid* pairs. For a valid pair, we check the elements in the line status in the range of $(-\infty, y_{t-1}]$. Since the elements are sorted in ascending order, y_{t-1} (resp. y_t) of a valid pair must be the last (resp. first) element among elements of the same value. Thus, we only need to check elements from the beginning of the line status to the first element (inclusive) of the pair. Starting with an empty set, which is called the *base set* and denoted by $\mathcal{R}$, if an element is a lower side, we add the center of the corresponding NN-circle to $\mathcal{R}$, otherwise we remove the center from $\mathcal{R}$. When reaching the second element (exclusive) of the pair, we stop and $\mathcal{R}$ is the RNN set of the pair. For example, in Fig. 10, the line status is $\mathcal{I}(3) = \|\underline{y_1}, \underline{y_2}, \bar{y}_1, \bar{y}_2\|$. For pair $\langle y_1, y_2\rangle$, $\underline{y_1}$ is the only element we encountered in the checking range and hence $\mathcal{R}(\langle y_1, y_2\rangle) = \{o_1\}$. We formally describe the above approach with the following corollary.

Corollary 1: $\forall \in \langle y_{t-1}, y_t\rangle \in \mathcal{I}(l)$ with $y_{t-1} \neq y_t$, the RNN set $\mathcal{R}(r_l^t)$ of a subregion $r_l^t = [x_{l-1}, x_l, y_{t-1}, y_t]$ can be obtained by checking elements y_i for $i = 1$ to $t-1$ and maintaining the set $\mathcal{R}(r_l^t)$ as follows

$$\begin{cases} o_k \text{ is removed from } \mathcal{R}(r_l^t) & \text{if } y_i \text{ is } \bar{y}_k, \\ o_k \text{ is added into } \mathcal{R}(r_l^t) & \text{if } y_i \text{ is } \underline{y_k}. \end{cases}$$

It is easy to observe that if we have obtained the RNN set $\mathcal{R}(\langle y_{t-1}, y_t\rangle)$ of a valid pair, we can start from y_t (which is the first element among elements of the same value) and use $\mathcal{R}(\langle y_{t-1}, y_t\rangle)$ as the base set for the valid pair $\langle y_{t'-1}, y_{t'}\rangle$ immediately next to it. In this way, we can obtain the RNN set of *every* valid pair (in one line status) with a single traversal of the line status. Continuing with the above example in Fig. 10, for pair $\langle y_2, y_3\rangle$, we use $\mathcal{R}(\langle y_1, y_2\rangle) = \{o_1\}$ as base set, encounter $\underline{y_2}$, add o_2, and stop with $\mathcal{R}(\langle y_2, y_3\rangle) = \{o_1, o_2\}$. For pair $\langle y_3, y_4\rangle$, we remove o_1 from $\{o_1, o_2\}$ and stop with $\mathcal{R}(\langle y_3, y_4\rangle) = \{o_2\}$.

C. Reducing the Number of Times of Region Labeling

1) Locating the Change Interval: With the above approach, we obtain the RNN sets and label the corresponding regions between two events e_{l-1} and e_l. We then move the sweep line forward across e_l and label regions between e_l and e_{l+1}. Crossing e_l, we obtain a new line status $\mathcal{I}(l+1)$. We notice that some of the pairs in $\mathcal{I}(l)$ and $\mathcal{I}(l+1)$ represent the same regions (not subregions) even though they are formed by different NN-circles. For example, in Fig. 10, between e_2 and e_3, $\mathcal{I}(3) = \|\underline{y_1}, \underline{y_2}, \bar{y}_1, \bar{y}_2\|$, while between e_3 and e_4, $\mathcal{I}(4) = \|\underline{y_3}, \underline{y_1}, \underline{y_2}, \bar{y}_3, \bar{y}_1, \bar{y}_2\|$. The pair $\langle \underline{y_2}, \bar{y}_1\rangle \in \mathcal{I}(3)$ and new pair $\langle \bar{y}_3, \bar{y}_1\rangle \in \mathcal{I}(4)$ represent the same region. Besides new pairs, a pair also represents the same region if it exists in both $\mathcal{I}(l)$ and $\mathcal{I}(l+1)$ and the RNN sets of the pair in the two line statuses are the same (e.g., $\langle \bar{y}_1, \bar{y}_2\rangle$ in the above example). The reason is that, by Lemma 1, the RNN set of a pair is changed if and only if the pair is entirely enclosed by an NN-circle that is inserted into (i.e., newly cut) or removed from (i.e., no longer cut by) the line. When the RNN set of a pair does not change, the two subregions formed by the pair must be connected (and hence represent the same region), since no side of NN-circles separates them. To reduce the number of times of region labeling, we should avoid processing pairs representing the same regions, i.e., only *some* of the newly formed pairs and the pairs that exist in both line status whose RNN sets are changed should be processed. We use the following lemma to precisely locate the pairs that need to be processed when only one NN-circle is changed in (i.e., inserted into or removed from) the line.

Lemma 2: When a line status $\mathcal{I}(l)$ is changed into a new line status $\mathcal{I}(l')$ because an NN-circle $\mathcal{C}(o_c) = [\underline{x_c}, \bar{x}_c, \underline{y_c}, \bar{y}_c]$ is newly or no longer cut by the sweep line, i.e., $\underline{y_c}$ and $\bar{y}_c$ are inserted into or removed from $\mathcal{I}(l)$, we only need to process the pairs in the following set

$$\{\langle y_{t-1}, y_t\rangle \in I(l') \mid \underline{y_c} \le y_{t-1} < y_t \le \bar{y}_c\}.$$

By Lemma 2, the pairs that need to be processed are located within a *range*. We call such a range a *changed interval* and denote it by $[y_{c_i}, y_{c_j}]$. Note that y_{c_i} and y_{c_j} are coordinate values, not line elements. When the line triggers (i.e., crosses) an event, multiple NN-circles are inserted into or removed from the line, and hence several (initial) changed intervals are created. We cannot process such changed intervals one by one, since they may intersect and affect each other. We need to merge the intersected changed intervals. When merging two changed intervals, we need to be careful about the line elements that are of the same value so that no regions are labeled repeatedly. Specifically, any two changed intervals

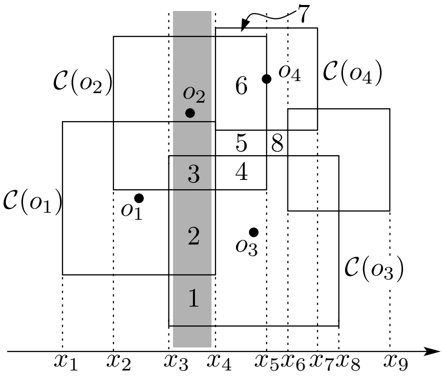


Fig. 11. Example of changed intervals

$[y_{c_i}, y_{c_j}]$ and $[y_{c_{i'}}, y_{c_{j'}}]$ with $y_{c_i} \leq y_{c_{i'}}$ are merged into a new one $[y_{c_i}, \max\{y_{c_j}, y_{c_{j'}}\}]$ if $y_{c_j} \geq y_{c_{i'}}$. After merging, we only need to handle separated changed intervals, which can be processed individually. For example, in Fig. 11, when the line crosses x_3, the grey area is processed, in which $\mathcal{C}(o_3)$ is inserted into the line. The pairs (i.e., subregions), for convenience denoted by 1, 2, and 3, need to be processed, which are in changed interval $[\underline{y_3}, \bar{y}_3]$. When the line crosses x_4, $\mathcal{C}(o_1)$ is removed and $\mathcal{C}(o_4)$ is inserted. Only pairs 4, 5, 6 and 7 in $[\underline{y_1}, \bar{y}_4]$ (merged from $[\underline{y_1}, \bar{y}_1]$ and $[\underline{y_4}, \bar{y}_4]$) are processed. When the line crosses x_5, $\mathcal{C}(o_2)$ is removed and only pair 8, the only one in $[\underline{y_2}, \bar{y}_2]$, is processed. The validity of the sweep line strategy still holds with the above approach, since a simple induction will show that all pairs in all line statuses are properly processed.

2) Caching and Retrieving Base Sets: To obtain the RNN set within a changed interval, an efficient way is to use the RNN set of the pair that is immediately preceding of the changed interval as the base set. Such a pair must be a valid pair, since the changed interval includes the line elements whose values are equal to the interval boundary. Therefore, we cache (only) the RNN set of each valid pair in the line status. We index the RNN set of a pair with its first element. Specifically, if the pair's first element is the lower (resp. upper) side of $\mathcal{C}(o_i)$, we assign the RNN set a *key* $2i - 1$ (resp. $2i$). When the RNN set of a pair is changed, the record in the index is also updated accordingly (for elements of the same value, the record is always maintained only at the last one for efficient access and space saving). In this way, the base set for a changed interval is the record of the element that is one position ahead of the changed interval. (In case that the change interval is at the end of the line status, we also keep an empty set for the last element of a line status.) When we process several separated changed intervals *in ascending order*, it is guaranteed that such a record is always available and up-to-date. Specifically, let y_t be the element whose record we need. If no such element y_t exists, the base set is an empty set, since the changed interval must be at the beginning of the line status. If y_t is the boundary of a preceding changed interval, then such a record is already updated and ready to use, otherwise y_t must exist in the last line status and the record is also available.

We use an example to illustrate the above approach. In Fig. 10, $\mathcal{I}(1)$ is empty, $\mathcal{I}(2) = \|\underline{y_1}, \bar{y}_1\|$, and $[\underline{y_1}, \bar{y}_1]$ is the changed interval which is at the beginning of $\mathcal{I}(2)$. We thus use an empty set as the base set, and keep the records $(2 \times 1 - 1, \{o_1\})$ for $\langle \underline{y_1}, \bar{y}_1 \rangle$, and $(2 \times 1, \varnothing)$ for $\bar{y}_1$ (the last element), respectively. In $\mathcal{I}(3)$, $\mathcal{C}(o_2)$ is inserted and the changed interval is $[\underline{y_2}, \bar{y}_2]$. The element immediately preceding the changed interval is $\underline{y_1}$. We obtain $\{o_1\}$ as the base set with key $2 \times 1 - 1 = 1$, and keep records $(2 \times 2 - 1 = 3, \{o_1, o_2\})$, $(2 \times 1 = 2, \{o_2\})$ and $(2 \times 2 = 4, \varnothing)$ for $\langle \underline{y_2}, \bar{y}_1 \rangle$, $\langle \bar{y}_1, \bar{y}_2 \rangle$ and $\bar{y}_2$, respectively. We now have $(1, \{o_1\})$, $(2, \{o_2\})$, $(3, \{o_1, o_2\})$ and $(4, \varnothing)$ cached for future use.

D. The Algorithm

We now present the detailed steps of CREST, as summarized in Algorithm 1. We first obtain the event queue $\mathcal{Q}_x$ by storing the vertical sides of the NN-circles in $\mathcal{Q}_x$ in ascending order (line 4). The sides are stored in a way such that for each side, we can directly obtain the NN-circle to which it belongs and whether it is the left or right side. We then process the elements in $\mathcal{Q}_x$ one by one (line 5). If an element is a left (resp. right) side, we insert (resp. remove) the two horizontal sizes of the NN-circle corresponding to the element into (resp. from) a balanced search tree $\mathcal{T}$ in which the data are stored in the doubly linked leaf nodes (e.g., a B$^+$-tree) (lines 6-14). When the next element in $\mathcal{Q}_x$ is greater than the current one, we process the event. The structure $\mathcal{T}$ now stores the information of the current line status. We obtain separated changed intervals by merging the y-coordinates of the inserted and removed sides in this event (line 15). For each changed interval (line 17), we locate the starting and ending elements in $\mathcal{T}$ (line 18), and retrieve the base set from the RNN set records (line 19), which are stored in a random access data structure

Algorithm 1: The CREST algorithm

```
   Input: An arrangement of $n$ NN-circles
   Output: A subdivision with each region labeled
 1 $\mathcal{T} \leftarrow \varnothing$        ◇ the index structure for the horizontal sides
 2 $\mathcal{U} \leftarrow \varnothing$        ◇ the changed NN-circles between events
 3 $\mathcal{P} \leftarrow \varnothing$        ◇ the cached RNN sets
 4 $\mathcal{Q}_x \leftarrow$ the vertical sides of the NN-circles in ascending order
 5 for each element $v$ in $\mathcal{Q}_x$ do
 6     $\mathcal{C}(o_i) \leftarrow$ the NN-circle to which $v$ belongs
 7     Add $\mathcal{C}(o_i)$ into $\mathcal{U}$
 8     if $v$ is a left side then
 9         Insert $\underline{y_i}$, $\bar{y}_i$ into structure $\mathcal{T}$
10     else
11         Delete $\underline{y_i}$, $\bar{y}_i$ from structure $\mathcal{T}$
12         Remove the corresponding records from $\mathcal{P}$
13     if the next element $v'$ of $\mathcal{Q}_x$ equals $v$ then
14         Continue the outer for-loop
15     Merge the changed intervals of the NN-circles in $\mathcal{U}$
16     Delete all elements in $\mathcal{U}$
17     for each separated changed interval do
18         Find the starting element $st$ and ending element $ed$
19         $\mathcal{R} \leftarrow$ Retrieve the base set from $\mathcal{P}$
20         for each element $y$ between $st$ and $ed$ do
21             $\mathcal{C}(o_j) \leftarrow$ the NN-circle to which $y$ belongs
22             if $y$ is the lower side then
23                 $\mathcal{R} \leftarrow$ add $o_j$ into the base set
24                 $p \leftarrow 2j - 1$
25             else
26                 $\mathcal{R} \leftarrow$ remove $o_j$ from the base set
27                 $p \leftarrow 2j$
28             if $y$ is greater than the next element $y'$ then
29                 Label the region represented by pair $\langle y, y' \rangle$
                   with set $\mathcal{R}$
30                 $\mathcal{P}[p] \leftarrow \mathcal{R}$
```

such as an array. We then sequentially check the elements in each changed interval (line 20). For each element, we either add or remove the corresponding data point (i.e., o_i) from the base set to obtain RNN sets of the valid pairs and label the regions (lines 21-29). To facilitate efficient insert, delete and copy operations on the base set, we keep the data points in a linked list and store pointers to the nodes in the linked list with an additional random access data structure indexed by the data points. The RNN sets we obtained are also dynamically recorded to support the future base set retrieval (line 30). After all changed intervals are processed, we eject the next element in $\mathcal{Q}_x$ and repeat the above steps until $\mathcal{Q}_x$ is empty.

VI. Complexity Analysis

A. Complexity of CREST

We analyze the time complexity of CREST following the steps in Algorithm 1. We sort the $2n$ vertical sides of the n NN-circles in $O(n \log n)$ time. When we process the events, each horizontal side is inserted into and then deleted from the structure $\mathcal{T}$ once. Therefore, there are at most $2n$ elements in $\mathcal{T}$, and the $2 \times 2n$ insertions and deletions can be done in $O(n \log n)$ time. To merge the changed intervals at an event, we can first sort them in lexicographical order and then obtain the merged result with a linear scan. This requires $O(\beta \log \beta + \beta) = O(\beta \log \beta)$ time, where β is the number of changed intervals at the event. Since each NN-circle can only be a changed interval twice, the total number of changed intervals in all events is $O(n)$. Thus, the overall time required for merging the changed intervals is bounded by $O(\sum \beta \log \beta) = O(\log n \sum \beta) = O(n \log n)$. For each merged changed interval, we obtain its starting element in $\mathcal{T}$ in $O(n \log n + \lambda)$ time, where λ is the maximum size of the RNN sets in the arrangement. This is because we first search in $\mathcal{T}$ in $O(\log n)$ time to obtain an element y_i whose value is equal to the lower endpoint of the interval. Starting from y_i, we obtain the starting element by checking backward (to the beginning of $\mathcal{T}$) until the elements are less than y_i. This procedure takes $O(2\lambda) = O(\lambda)$ time, since λ is the maximum size of the RNN sets and there are at most λ upper sides and λ lower sides that are of the same y-coordinate. Symmetric analysis applies to obtaining the ending element. We have only $O(n)$ changed intervals, and thus obtaining starting and ending elements can be done in $O(n \log n + n\lambda)$ time. We then process the elements between them. We first retrieve a base set, which takes at most $O(\lambda)$ copying time. Thus, it takes $O(n\lambda)$ time to obtain base sets for $O(n)$ changed intervals. For each element between the starting and ending elements, we either add into or remove from the base set its corresponding data point (i.e., o_i). It takes at most $O(\lambda)$ time for the adding or removing operations to obtain an RNN set for a valid pair. This is because to get an RNN set of size α_t by changing an RNN set of size α_s, at most α_s data points are removed and α_t data points are added, which takes $O(\alpha_s + \alpha_t) = O(\lambda)$ time. We denote by k the number of valid pairs, and hence the time for obtaining the RNN sets for k valid pairs is bounded by $O(k\lambda)$. For each valid pair, we record its RNN set and label its corresponding region, and this takes $O(k\lambda)$ time.

Putting all things together, we have that CREST stops in $O(n \log n + n\lambda + k\lambda)$ time. Since k denotes the number of times of region labeling in CREST, k must be greater than or equal to the number of regions in the arrangement which is in turn greater than or equal to the number of NN-circles n. Therefore, the time complexity of CREST is $O(n \log n + k\lambda)$. Shortly (in Section VI-B) we will prove that $k = \Theta(r)$, where r is the number of regions in the arrangement.

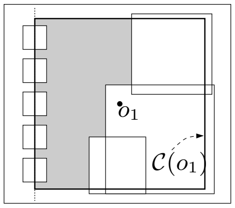


Fig. 12. Multilabelling

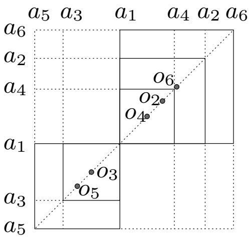


Fig. 13. Reduction

The space required by the queue $\mathcal{Q}_x$ and structure $\mathcal{T}$ is $O(n)$. The space required by caching the RNN sets is $O(n\lambda)$, and the storage of the base set requires $O(n+\lambda)$ space. Overall, the space complexity of CREST is $O(n\lambda)$.

B. Bounding the Times of Region Labeling in CREST

From the above analysis, we can see that the number of times of region labeling k largely decides the performance of CREST. In CREST, we successfully avoid labeling the same region multiple times in different line statuses. Although rarely happens, multi-labeling still exists within the same line status. For example, in Fig. 12, at the event of the left side of $\mathcal{C}(o_1)$, the gray region is labeled six times. Despite the multi-labeling, we show with the following lemma that the number of times of region labeling in CREST and the number of regions in the arrangement, up to a constant factor, are asymptotically the same (as a function of n).

Lemma 3: $k = \Theta(r)$, where r is the number of regions in the arrangement.

Proof: Let v, e, and c be the number of vertices, edges and connected components in the arrangement, respectively. We call the number of edges bounding a region the degree of the region, and the number of edges incident to a vertex the degree of the vertex. In an arrangement of squares, there are only 2-, 3-, and 4-degree vertices, which are denoted by v_2, v_3, and v_4, respectively. In CREST, the number of times a region is labeled cannot be greater than the degree of the region, since each time the region is labeled we need a distinct valid pair which requires at least one of the edges bounding the region. Therefore, k is less than or equal to the sum of degrees of all regions which equals $2e$, i.e., $k \leq 2e$. In the arrangement, we also have $v = v_2 + v_3 + v_4$, $2e = 4v_4 + 3v_3 + 2v_2$ and $v - e + r - c = 1$ (Euler characteristic). Combining these three equations, we obtain $r = v_4 + v_3/2 + c + 1$. We then have that

$$k \leq 2e = 4v_4 + 3v_3 + 2v_2 \leq 6(v_4 + v_3/2 + c + 1) + 2v_2 = 6r + 2v_2.$$

The number of 2-degree vertices is less than or equal to $4n$ and hence less than or equal to $4r$, since each square makes at most four 2-degree vertices and $n \leq r$. Therefore, it follows that

$$k \leq 6r + 2v_2 \leq 6r + 8n \leq 14r.$$

Obviously, $r \leq k$, and hence $r \leq k \leq 14r$, which completes the proof. ■

We conclude the above analysis with the following theorem.

Theorem 2: The CREST algorithm solves the (bichromatic) RC problem in $O(n \log n + r\lambda)$ time with $O(n\,\lambda)$ space, where r and λ are the number of regions and the maximum size of the RNN sets in the arrangement, respectively.

C. A Lower Bound of the RC Problem

We show that $\Omega(n \log n + r\lambda^*)$ is a lower bound of the RC problem (in the algebraic computation tree model) [3], where λ^* is the *average* size of RNN sets in the arrangement. When $r\lambda^*$ is the dominating term, at least the RNN sets of all regions need to be output, the above bound is a trivial lower bound. Thus, we only need to show that it requires $\Omega(n \log n)$ operations even without considering the output cost. This bound is proved by the reduction from the *point distinctness problem* to a special case of the RC problem.

Definition 3 (Element Distinctness): Given real numbers $a_1, \ldots, a_n \in \mathbb{R}$, determine whether or not there is a pair i, j with $i \neq j$ and $a_i = a_j$.

We show that the element distinctness problem can be reduced to the RC problem in linear time. For each real number a_i, we create a point (a_i, a_i), $i = 1, 2, \ldots, n$ in the plane. We then build a square $\mathcal{C}(o_i)$ with point (a_i, a_i), $i = 2, \ldots, n$ and point (a_1, a_1) being the diagonally opposite corners and o_i being the center. An example of such reduction is shown in Fig. 13. These squares form an arrangement of NN-circles in a two-dimensional space. We use this arrangement as input to any algorithm that solves the RC problem. A correct algorithm outputs exactly n RNN sets (including the empty set) if and only if the elements are distinct. The reason is that each RNN set corresponds to only one region, and there are n regions (including the exterior face) in the arrangement if and only if the elements are distinct. It has been proved that the element distinctness problem has a lower bound $\Omega(n \log n)$ [3] (in the algebraic computation tree model), which implies that RC has a lower bound $\Omega(n \log n)$ without the output cost. Therefore, $\Omega(n \log n + r\lambda^*)$ is a lower bound of the RC problem.

D. Optimality of CREST

From the time complexity of CREST and lower bound of the RC problem, CREST is asymptotically optimal in terms of the number of times of region labeling (i.e., influence computation) in all cases, since $k = \Theta(r)$.

In the following cases, we show that the upper bound $O(n \log n + r\lambda)$ of CREST is also tight, which indicates that CREST is overall asymptotically optimal. For the bound to be tight, it is sufficient to show that $\lambda = \Theta(\lambda^*)$.

Case (i). When the clients and facilities are relatively uniformly distributed such that λ is bounded by a sufficiently large constant C (which depends on $\frac{|\mathcal{O}|}{|\mathcal{F}|}$), since $\lambda^* \leq \lambda$, λ^* is also bounded by C. Thus, $\lambda = \Theta(\lambda^*) = O(1)$. An example is that none of the n squares intersects any other ones (or only a few of them overlap).

Case (ii). When λ is unbounded, we show with the worst case illustrated in Fig. 8 that $\lambda = \Theta(\lambda^*)$ also holds when every square intersects all the other ones. In this arrangement, $\lambda = n$, and we have that $r = n^2 - n + 2$ and $r \cdot \lambda^* = \frac{n^3+2n}{3}$. Therefore, it follows that

$$\lambda^* = \frac{n^3 + 2n}{3(n^2 - n + 2)} = \frac{n}{3} \cdot \frac{n^3 + 2n}{n^3 - n^2 + 2n} \geq \frac{n}{3} = \frac{\lambda}{3}.$$

Since $\lambda^* \leq \lambda$, it follows that $\lambda = \Theta(\lambda^*)$, which indicates CREST is overall asymptotically optimal.

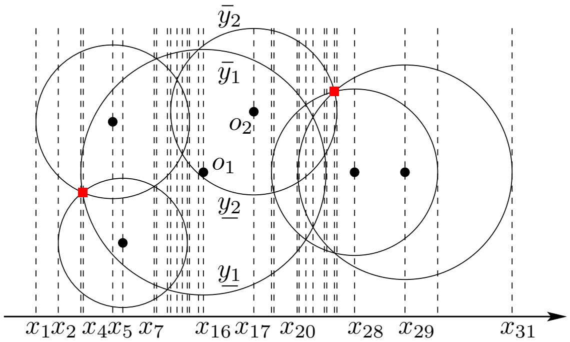

$\bar{y}_2$
$\bar{y}_1$
o_2
o_1
$\underline{y_2}$
$\underline{y_1}$
x_1 x_2 x_4 x_5 x_7 x_{16} x_{17} x_{20} x_{28} x_{29} x_{31}

Fig. 14. CREST with L_2 distance metric

VII. RNNHM in Other Settings

We show how CREST solves the RNNHM problem with the monochromatic RNNs, L_1, and L_2 distance metrics.

A. Monochromatic RNNs

CREST directly applies to the monochromatic RNNs, since $\mathcal{O}$ and $\mathcal{F}$ being the same set does not affect the computation of the NN-circle and these NN-circles still form a planar subdivision with axis-aligned edges as in the bichromatic RNNs. By Korn et al. [12], an RNN set contains at most six points for monochromatic RNN queries, which means $\lambda = O(1)$. Therefore, by Theorem 2, the time complexity of CREST for the monochromatic RNNs is $O(n \log n + r)$ and the space complexity is $O(n)$.

B. RNNHM with L1 Distance

In two-dimensional spaces, the L_1 distance can be viewed as equivalent to the L_∞ distance by rotation and scaling. Specifically, with the L_1 distance, NN-circles are of diamond shape. If we rotate (around the origin) the coordinate system counter-clockwise by $\pi/4$, diamonds become squares. Each point (x, y) in the original system has a corresponding point (x', y') in the rotated system with $x' = x \cos\theta - y \sin\theta$, $y' = x \sin\theta + y \cos\theta$ and $\theta = \pi/4$. In the rotated system, CREST directly applies. The transformation takes $O(n)$ time and the overall time and space complexities stay unchanged.

C. RNNHM with L2 Distance

With the L_2 distance, the NN-circles are of circular shape. They form a planar subdivision with *curved* edges, as shown in Fig. 14. CREST still applies in such a subdivision but requires modifications as follows. We use the x-extreme points of circles (instead of vertical sides of squares) as event points. In the line status, we use the arc segments of circles between two consecutive events as line elements (instead of horizontal sides of squares). For each line element y_i (i.e., an arc segment), we assign two values y_i^s and y_i^l, which are the smallest and largest y-coordinates of y_i between two consecutive events e_{l-1} and e_l, respectively. Line element y_i is less than y_j iff (i) $y_i^s < y_j^s$ or (ii) $y_i^s = y_j^s$ and $y_i^l < y_j^l$ or (iii) $y_i^s = y_j^s, y_i^l = y_j^l$ and

$y_i^m < y_j^m$, where y_i^m and y_j^m are the y-coordinates of y_i and y_j at $\frac{x_{l-1}+x_l}{2}$, respectively. We include intersection points as event points (e.g, x_4 and x_7 in Fig. 14). This is because the arc segments of NN-circles switch positions at intersection points. We also use the center of each NN-circle as event points (e.g., x_5 and x_{29} in Fig. 14) to guarantee that each line element is y-monotone (i.e., strictly increasing or decreasing in the y-dimension). Before processing an event, we update values y_i^s and y_i^l for each line element y_i regardless of whether it is related to the event. This update is required in order to maintain a proper order in the line status because the arcs go up or down between events and their upper and lower values change even if they are irrelevant to the events. Note that such an update does not change the relative order of the line elements irrelevant to the events, and thus can be completed in linear time.

Apart from the above modifications, CREST remains the same. Specifically, if an event point at e_{l-1} is the left boundary point of $\mathcal{C}(o_c)$, we insert $\underline{y_c}$ and $\bar{y}_c$ into the line status with $\underline{y_c}^l = \bar{y}_c{}^s = y_{o_c}$ and $\underline{y_c}^s$ and $\bar{y}_c{}^l$ being the lower and upper y-coordinates where e_l intersects $\mathcal{C}(o_c)$, respectively. We also create a change interval with $\underline{y_c}$ and $\bar{y}_c$. If an event point is an intersection point, we obtain the relevant line elements (arcs incident to the intersection point) in the line status, switch their positions and create a change interval with these elements. If an event point is the right boundary point or center of an NN-circle, we remove or update the two line elements corresponding to the NN-circle. We do not create change intervals for either of these two types of event points, since no pair is between the removed elements and updating the line elements by the centers is only to keep them y-monotone. We then merge and handle change intervals as before.

Complexities. In the worst case (as shown in Fig. 8), CREST runs in $O(n^3)$ time with the L_2 distance, since there can be as many as $O(n^2)$ events and for each event we need to update $O(n)$ line elements. However, the worst case complexity is much lower than that of an existing algorithm [22], which suffers from an exponential running time in the worst case. The algorithm was proposed to obtain regions with the maximum influence value, but it could be adapted to solve the RC problem if we remove its pruning techniques. The algorithm [22] follows the filter and refine paradigm by enumerating all possible regions and then checking their existence. For example, when $\mathcal{C}(o_1)$ intersects $\mathcal{C}(o_2)$ and $\mathcal{C}(o_3)$, it enumerates the regions $\hat{o_1}\hat{o_2}\hat{o_3}$, $\hat{o_1}\hat{o_2}\bar{o_3}$, $\hat{o_1}\bar{o_2}\hat{o_3}$, $\hat{o_1}\bar{o_2}\bar{o_3}$, where $\hat{o_i}$ means inside $\mathcal{C}(o_i)$ and $\bar{o_i}$ means outside $\mathcal{C}(o_i)$, and then checks whether such regions really exist. In our experiments (in Section VIII), CREST constantly outruns the algorithm on data sets of various settings.

VIII. Experiments

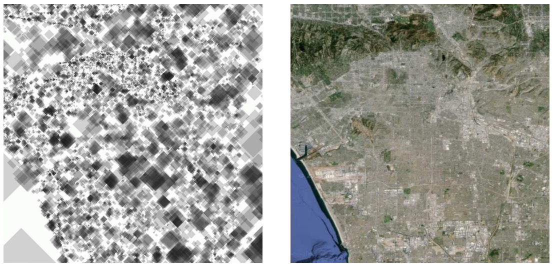
In this section, we experimentally evaluate the performance of CREST. We use both real and synthetic data sets. Two real data sets, NYC and LA, contain points-of-interest in New York City and Los Angeles, respectively (we obtain the data sets from the authors of [2]). Table II lists the details of the real data sets. We also generate two synthetic data sets, Uniform and Zipfian, which contain points of uniform and Zipfian distributions, respectively. The skew coefficient in Zipfian distribution is set to 0.2. In the experiments, we use the bichromatic RNNs since the monochromatic type is just a special case of the bichromatic RNNs. We use L_1 and L_2 distance metrics since they are used more often than L_∞ in real-world scenarios, and L_1 and L_∞ are equivalent in two-dimensional spaces. We uniformly sample from the data sets to obtain the client set $\mathcal{O}$ and the facility set $\mathcal{F}$. All algorithms are implemented using C++ and the experiments are conducted on a desktop computer with a 3.4GHz Intel i7-2600 CPU and 8GB main memory.

TABLE II
Real Data Sets

Name	Size	Description
NYC	128,547	points-of-interest in New York City
LA	116,596	points-of-interest in Los Angeles

A. Showcasing Real-World Heat Maps

In the first set of experiments, we show the RNN heat maps for two cities: New York City and Los Angeles. For each data set NYC and LA, we uniformly sample $20,000$ points as the clients and $6,000$ points as the facilities, since in real world scenarios the number of clients is usually larger than the number of facilities. For simplicity, we measure the influence by the size of RNN sets, although any other function on the RNN sets may be used. Fig. 1(a) and Fig. 1(b) (in Section I) show the RNN heat map and the satellite map of New York City (within latitude and longitude ranges $[40.50,\ 40.95] \times [-74.15,\ -73.70]$), while Fig. 15(a) and Fig. 15(b) show the heat and satellite maps of Los Angeles (within $[33.82, 34.17] \times [-118.47,\ -118.12]$), respectively. Comparing the heat and satellite maps, we can see that they are closely geographically correlated as expected. For instance, the mountain and sea area have few clients or facilities, and hence have very low heat. We can easily explore regions of various influences to help various decision making applications such as those described in the motivating examples. If the decision maker is interested in any specific area, she can zoom in to see more details.

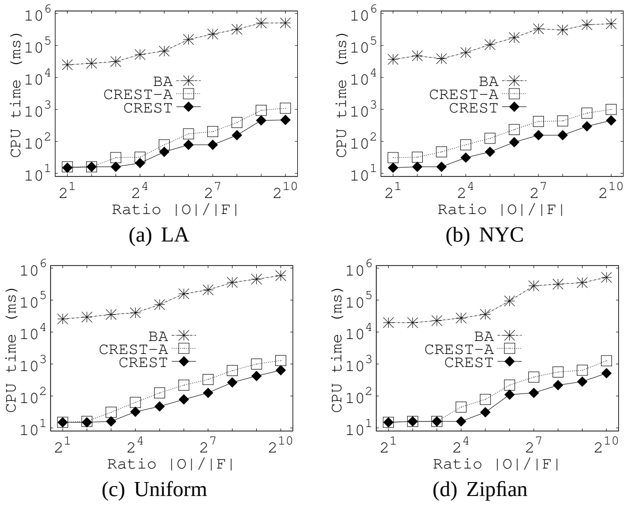


(a) Heat map for LA (b) Satellite map for LA

Fig. 15. Real-world heat map

B. Performance of CREST with L1 Distance

In this set of experiments, we compare the running time of three algorithms: the baseline algorithm (**BA**), the CREST algorithm with only the RNN computation optimization, denoted by **CREST-A**, and the **CREST** algorithm with both RNN computation optimization and repetitive region labeling optimization. We cannot evaluate the effect of the latter optimization alone, since it is built upon the former optimization. We compute the influence by (i) the size of RNN sets and (ii) the function considering the capacity constraints of facilities [22] (described in the Introduction), respectively. The results of the latter function are consistent with those using the size and hence are omitted due to space limitation.



(a) LA

(b) NYC

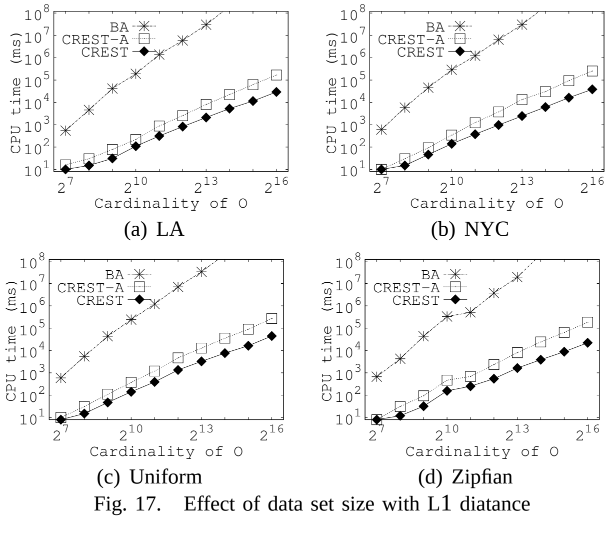
(c) Uniform

(d) Zipfian

Fig. 16. Effect of $\frac{|\mathcal{O}|}{|\mathcal{F}|}$ with L1 diatance

Effect of $\frac{|\mathcal{O}|}{|\mathcal{F}|}$. We first vary the ratio of $\frac{|\mathcal{O}|}{|\mathcal{F}|}$ from 2^1 to 2^{10}. Since the baseline algorithm does not terminate within 24 hours on large data sets, we only show results on relatively small data sets and fix $n = |\mathcal{O}|$ at 2^{10}. We plot the results in Fig. 16 (note the log scale in the axes). We can see that in all data sets, CREST outperforms the baseline by at least three orders of magnitude, and outperforms CREST-A by several times. With the increase of $\frac{|\mathcal{O}|}{|\mathcal{F}|}$, the running time of CREST also moderately increases. This is because both the number of regions and the maximum size of the RNN sets increase with $\frac{|\mathcal{O}|}{|\mathcal{F}|}$. The growth rates of CREST-A and CREST are similar, which indicates that the ratio of $\frac{|\mathcal{O}|}{|\mathcal{F}|}$ mainly affects the number of regions in the arrangement, but not the number of times a same region is repeatedly labeled. We can also see from the slope of the lines that with the increase of $\frac{|\mathcal{O}|}{|\mathcal{F}|}$, the number of regions increases only polynomially (rather than exponentially). This indicates that the performance of CREST will stay stable even if $\frac{|\mathcal{O}|}{|\mathcal{F}|}$ becomes very large.

Effect of Data Set Size. We then fix $\frac{|\mathcal{O}|}{|\mathcal{F}|}$ at 2^7 and vary the size of the client set $\mathcal{O}$ from 2^7 to 2^{16}. The results are plotted in Fig. 17. When the size of $\mathcal{O}$ is greater than 2^{13}, the baseline runs for more than 24 hours and is early terminated, hence the results are not presented.



(a) LA

(b) NYC

(c) Uniform

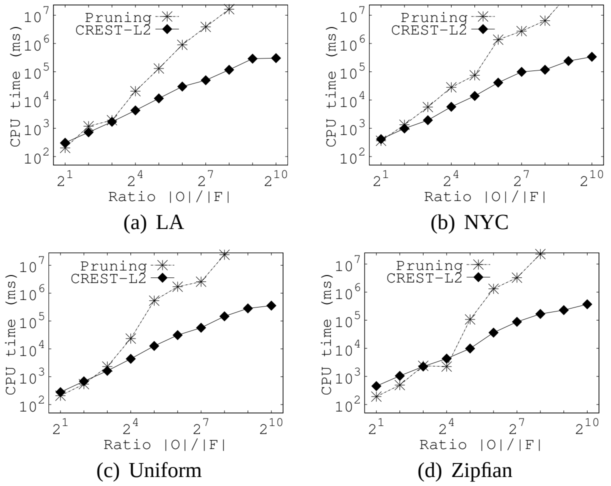
(d) Zipfian

Fig. 17. Effect of data set size with L1 diatance

Again we can see that CREST outruns the baseline by at least three orders of magnitude and outruns CREST-A by up to an order of magnitude. The running time of the baseline increases much faster than that of the other two algorithms, which indicates the number of point enclosure queries computed in the baseline increases dramatically when $n = |\mathcal{O}|$ becomes larger. The growth rate of CREST-A is also higher than that of CREST. This implies that the number of times of repeated labeling becomes larger with the increase of data size. The lowest growth rate of CREST also demonstrates its scalability for processing much larger data sets.

C. Performance of CREST with L2 Distance

We repeat the above experiments with the L_2 distance metric, where the CREST algorithm for L_2 (**CREST-L2**) is compared with the pruning algorithm (**Pruning**) described in Section VII-C. We compute the influence with the function in [22] and use the two algorithms to find the regions with the *maximum* influence, since in such settings the pruning algorithm performs the best. This also shows the flexibility of CREST since the adaptation to various influence functions and supporting post-processing operations is very easy.



(a) LA

(b) NYC

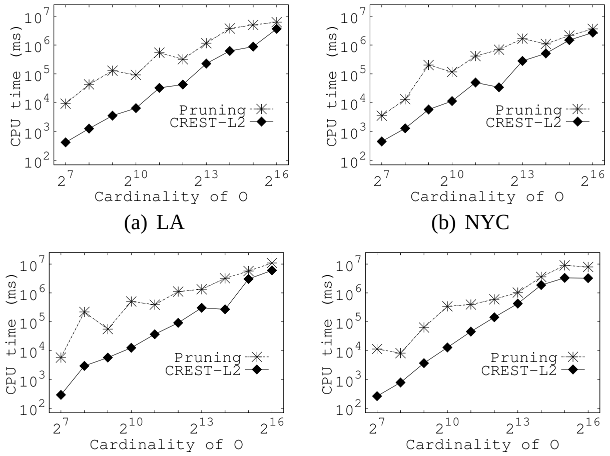
(c) Uniform

(d) Zipfian

Fig. 18. Effect of $\frac{|\mathcal{O}|}{|\mathcal{F}|}$ with L2 distance

Effect of $\frac{|\mathcal{O}|}{|\mathcal{F}|}$. We first vary the ratio from 2^1 to 2^{10} and plot the results in Fig. 18. In all data sets, when the ratio is greater than 2^3, we can see that CREST consistently outperforms the Pruning algorithm by several orders of magnitude. With the increase of $\frac{|\mathcal{O}|}{|\mathcal{F}|}$, the performance of the Pruning algorithm deteriorates rapidly, since the number of regions enumerated grows exponentially with the increase of $\frac{|\mathcal{O}|}{|\mathcal{F}|}$. Comparing with the Pruning algorithm, CREST has a much lower growth rate. When $\frac{|\mathcal{O}|}{|\mathcal{F}|}$ is less than or equal to 2^2, in Fig. 18(d) the Pruning algorithm runs slightly faster than CREST. This is because the number of regions enumerated in the Pruning algorithm is small, while the number of events in CREST is large when the data distribution is very skewed. Overall, CREST still outruns the Pruning algorithm by up to three orders of magnitude.

Effect of Data Set Size. Next, we fix $\frac{|\mathcal{O}|}{|\mathcal{F}|}$ at 2^5 and vary the size of $|\mathcal{O}|$ from 2^7 to 2^{16}. The results are presented in Fig. 19. In all the data sets, again CREST consistently outperforms the Pruning algorithm. With the increase of $|\mathcal{O}|$, CREST and the Pruning algorithm have a similar growth rate. The running time of the Pruning algorithm gets closer to that of CREST when $|\mathcal{O}|$ is very large. This is because although the number of regions increases with the increase of $|\mathcal{O}|$, most of them are pruned without being searched in the Pruning algorithm, which does not happen in CREST. It is notable that even we solve the maximization problem with CREST which is quite general, it still outruns the specialized Pruning algorithm designed for the problem, which demonstrates the efficiency of CREST.



(a) LA (b) NYC

(c) Uniform (d) Zipfian

Fig. 19. Effect of data set size with L2 distance

IX. Conclusions

In this paper, we proposed the RNN heat map problem, which computes the influence of every point in the space. Comparing to existing studies which give only the points or regions with the highest influence, the RNN heat map enables exploring the influence of the whole space while considering qualitative factors at any instant during the exploration. We solved the problem by first reducing it to the Region Coloring (RC) problem, and then computing the influence on regions instead of points with a novel algorithm called CREST. We proposed two techniques in CREST, one to avoid point enclosure queries in the influence computation and the other to reduce the total number of times of the influence computation. Through a detailed analysis, we showed that the number of influence computation in CREST is asymptotically optimal. We also showed that the worst-case time complexity of CREST is much lower than that of the baseline algorithm and in many cases meets the lower bound of RC. We conducted extensive experiments on both real and synthetic data sets. The results showed that CREST outperforms alternative algorithms by up to three orders of magnitude.

Acknowledgments. This work is partially supported by the National Natural Science Foundation of China (Nos. 61402155 and 61432006). Rui Zhang is supported by ARC Future Fellow project FT120100832. Jianzhong Qi is supported by Melbourne School of Engineering Early Career Researcher Grant (No. 4180-E55) and University of Melbourne Early Career Researcher Grant (No. 603049).

References

[1] M. E. Ali, R. Zhang, E. Tanin, and L. Kulik. A motion-aware approach to continuous retrieval of 3d objects. In *ICDE*, pages 843–852, 2008.

[2] J. Bao, Y. Zheng, and M. F. Mokbel. Location-based and preference-aware recommendation using sparse geo-social networking data. In *GIS*, pages 199–208, 2012.

[3] M. Ben-Or. Lower bounds for algebraic computation trees. In *STOC*, pages 80–86, 1983.

[4] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. on Computers*, 100(9):643–647, 1979.

[5] J. L. Bentley and D. Wood. An optimal worst case algorithm for reporting intersections of rectangles. *IEEE Trans. on Computers*, 1980.

[6] S. Cabello, J. M. Díaz-Báñez, S. Langerman, C. Seara, and I. Ventura. Facility location problems in the plane based on reverse nearest neighbor queries. *Eur. J. of Oper. Res.*, 202(1):99–106, 2010.

[7] Y. Chen and J. M. Patel. Efficient evaluation of all-nearest-neighbor queries. In *ICDE*, pages 1056–1065, 2007.

[8] Z. Chen, Y. Liu, R. C.-W. Wong, J. Xiong, G. Mai, and C. Long. Efficient algorithms for optimal location queries in road networks. In *SIGMOD*, pages 123–134, 2014.

[9] M. De Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry*. 2008.

[10] Y. Gao, S. Qi, L. Chen, B. Zheng, and X. Li. On efficient k-optimal-location-selection query processing in metric spaces. *Information Sciences*, 298:98–117, 2015.

[11] J. Huang, Z. Wen, J. Qi, R. Zhang, J. Chen, and Z. He. Top-k most influential locations selection. In *CIKM*, pages 2377–2380, 2011.

[12] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *SIGMOD*, pages 201–212, 2000.

[13] J. Lu, Y. Lu, and G. Cong. Reverse spatial and textual k nearest neighbor search. In *SIGMOD*, pages 349–360, 2011.

[14] S. Ma, Y. Zheng, and O. Wolfson. Real-time city-scale taxi ridesharing. *TKDE*, 2014.

[15] A. Maheshwari, J. Vahrenhold, and N. Zeh. On reverse nearest neighbor queries. In *CCCG*, pages 128–132, 2002.

[16] S. Melkote and M. S. Daskin. Capacitated facility location/network design problems. *Eur. J. of Oper. Res.*, 129(3):481–495, 2001.

[17] J. Qi, R. Zhang, L. Kulik, D. Lin, and Y. Xue. The min-dist location selection query. In *ICDE*, pages 366–377, 2012.

[18] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29(7):669–679, 1986.

[19] J. She, Y. Tong, and L. Chen. Utility-aware social event-participant planning. In *SIGMOD*, pages 1629–1643, 2015.

[20] J. She, Y. Tong, L. Chen, and C. C. Cao. Conflict-aware event-participant arrangement. In *ICDE*, pages 735–746, 2015.

[21] Y. Sun, J. Huang, Y. Chen, X. Du, and R. Zhang. Top-k most incremental location selection with capacity constraint. In *WAIM*, pages 165–171, 2012.

[22] Y. Sun, J. Huang, Y. Chen, R. Zhang, and X. Du. Location selection for utility maximization with capacity constraints. In *CIKM*, pages 2154–2158, 2012.

[23] Y. Sun, J. Qi, R. Zhang, Y. Chen, and X. Du. Mapreduce based location selection algorithm for utility maximization with capacity constraints. *Computing*, 97(4):403–423, 2015.

[24] Y. Sun, J. Qi, Y. Zheng, and R. Zhang. K-nearest neighbor temporal aggregate queries. In *EDBT*, pages 493–504, 2015.

[25] V. K. Vaishnavi. Computing point enclosures. *IEEE Trans. on Computers*, 100(1):22–29, 1982.

[26] R. C.-W. Wong, M. T. Özsu, A. W.-C. Fu, P. S. Yu, L. Liu, and Y. Liu. Maximizing bichromatic reverse nearest neighbor for l p -norm in two- and three-dimensional spaces. *VLDBJ*, 2011.

[27] T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On computing top-t most influential spatial sites. In *VLDB*, pages 946–957, 2005.

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

[29] D.-N. Yang, C.-Y. Shen, W.-C. Lee, and M.-S. Chen. On socio-spatial group query for location-based social networks. In *KDD*, pages 949–957, 2012.

[30] R. Zhang and M. Stradling. The hv-tree: a memory hierarchy aware version index. *PVLDB*, 3(1-2):397–408, 2010.

[31] Z. Zhou, W. Wu, X. Li, M. L. Lee, and W. Hsu. Maxfirst for maxbrknn. In *ICDE*, pages 828–839, 2011.