MaxBRkNN Queries for Streaming Geo-Data

Hui Luo[1], Farhana M. Choudhury[1], Zhifeng Bao[1], J. Shane Culpepper[1] and Bang Zhang[2]

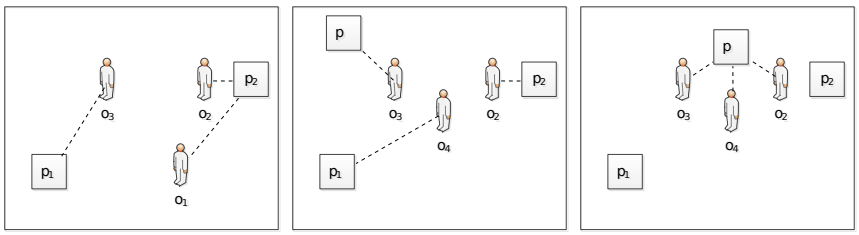
[1] School of Science, RMIT University
{hui.luo, farhana.choudhury, zhifeng.bao, shane.culpepper@rmit.edu.au}@rmit.edu.au
[2] CSIRO, Mattbang.Zhang@data61.csiro.au

Abstract. The problem of maximizing bichromatic reverse k nearest neighbor queries (MaxBRkNN) has been extensively studied in spatial databases, where given a set of facilities and a set of customers, a MaxBRkNN query returns a region to establish a new facility p such that p is a kNN of the maximum number of customers. In the literature, current solutions for MaxBRkNN queries are predominantly static. However, there are numerous applications for dynamic variations of these queries, including advertisements and resource reallocation based on streaming customer locations via social media check-ins, or GPS location updates from mobile devices. In this paper, we address the problem of continuous MaxBRkNN queries for streaming objects (customers). As customer data can arrive at a very high rate, we adopt two different models for recency information (sliding windows and micro-batching). We propose an efficient solution where results are incrementally updated by reusing computations from the previous result. We present a *safe interval* to reduce the number of computations for the new objects, and prune the objects that cannot affect the result. We perform extensive experiments on datasets integrated from four different real-life data sources, and demonstrate the efficiency of our solution by rigorously comparing how different properties of the datasets can affect the performance.

1 Introduction

Given two distinct types of objects, a set P of facilities and a set O of customers, if a facility p ($p \in P$) is a kNN of a customer o ($o \in O$), then o is one of the *Bichromatic Reverse k Nearest Neighbor* (BRkNN) of p. Given a set of facilities and a set of customers, a *Maximizing Bichromatic Reverse k Nearest Neighbor* (MaxBRkNN) query returns a region to establish a new facility p such that p is a kNN of the maximum number of customers [6, 12, 19]. In this study, we explore the problem of MaxBRkNN queries over streaming geo-data in spatial databases. This problem is critical in many real-time resource supply scenarios. For example, when a disaster happens, how can supplies be allocated dynamically to different rescue stations? The optimal location p with the greatest need for supplies should be updated based on patient arrivals in near real-time.

However, existing MaxBRkNN studies neglect the fact that the cardinality of the objects in the current spatial region can change continuously as new objects



(a) (b) (c)

Fig. 1: Motivation Example

are arriving and expiring. Below is an example of MaxBRkNN on streaming objects. For ease of illustration, we will consider $k = 1$ in the following example, which can be easily extended to the case k>1.

Example 1. Figure 1a shows the location of two facilities p_1 and p_2, and the location of three customers o_1, o_2, o_3 for the time instance t_0. As the set of customers changes, due to the arrival of new customers and the departure of other customers, Figure 1b and Figure 1c present two alternatives for new facility p placements at time t_1. The reverse kNN of each facility is shown with a connecting dotted line in the figures.

In Figure 1a, $RNN(p_1, P)$=$\{o_3\}$, $RNN(p_2, P)$=$\{o_1, o_2\}$, where $RNN(p_i, P)$ denotes the set of RNN customers for p_i in P. After the arrival of a new customer o_4 and the departure of a customer o_1 at time t_1, Figure 1b and Figure 1c show two alternative location choices for placing a new p that can serve the maximum number of customers. The location shown in Figure 1b is not a suitable choice as $RNN(p, P \cup p) = \{o_3\}$, where the location shown in Figure 1c is the optimal choice as $RNN(p, P \cup p) = \{o_2, o_3, o_4\}$. As the set of customers changes, the optimal location needs to be continuously updated.

In this paper, we address the problem of continuous updates in MaxBRkNN queries for streaming objects, where, given a set P of static facility locations, a stream of locations of customers O, a positive integer k, the problem is to update the optimal region in space to place a new facility p such that p has the maximum number of BRkNNs in O. To maintain recency information and minimize memory costs, a sliding window model is imposed on the stream, and a customer is valid only while it remains in the window.

We consider two commonly used sliding window models: *count-based* and *time-based* windows [8]. Both are represented by a window size $|W|$ and a slide size Δw, which are either a fixed number of objects for *count-based* windows, or time intervals for *time-based* windows. We also consider two variants of count-based windows: (i) real-time updates where Δw is '1', i.e., the result needs to be updated each time a new customer arrives; and (ii) micro-batching of customers for Δw> 1, i.e., the Δw customer changes are processed together to update the result. We propose solutions robust to both windows, however, the choice of an appropriate sliding window setting will depend on the intended application.

To the best of our knowledge, there is no prior work on supporting MaxBRkNN queries over streaming objects. Existing approaches present static-only solutions [6, 12, 19]. In large datasets, static MaxBRkNN solutions are

incapable of efficiently supporting sliding window models since the number of redundant computations are incurred as a function of the window size. Therefore, we propose new approaches to incrementally update the current result set based on the previous computations.

In our proposed approach, we reuse the computations for the customers shared between two consecutive windows to update the optimal result in order to avoid redundant computations. Specifically, every time the sliding window shifts a distance of Δw, the results of the previous window and the overlapping objects are used to update the information of the new objects locally. We present the notion of a "*safe interval*" to reduce the number of computations for updated objects. Then only the objects that can influence the optimal region will be evaluated to update the result.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 formalises the MaxBRkNN problem. Section 4 proposes our baseline algorithm and optimization method. Section 5 evaluates the proposed methods through extensive experiments on real dataset, and Section 6 concludes the paper.

2 Related Work

Several studies have investigated the problem of finding a location or a region in space to establish a new facility such that the facility can serve the maximum number of customers based on different optimization criteria. The related body of work includes query processing on (i) static objects and (ii) dynamic objects.

2.1 Facility location selection queries on static objects

Based on the optimization criteria, the facility location selection problem can be categorized mainly as Maximizing Bichromatic Reverse k Nearest Neighbor (MaxBRkNN) queries and distance aggregation queries.

MaxBRkNN queries. Wong et al. [12] introduced the MaxOverlap algorithm to solve the MaxBRkNN problem. Here, a circle is first defined for each object $o \in O$, denoted by the Nearest Location Circle (NLC), where the center of the circle is o and the radius is the distance between o and its kNN. They reduced the problem of finding a region in space to the problem of finding the intersection point of the NLCs that is covered by the largest number of NLCs. The optimal region is obtained from the overlap of such NLCs. The MaxOverlap algorithm was extended to support the L_p-norm and three-dimensional space in a later work by the same authors [13].

Zhou et al. [19] presented the MaxFirst algorithm. In contrast to the other studies, they consider the probability of a customer o visiting each of the i^{th} nearest facilities while finding the result. They construct a Quadtree by iteratively partitioning the space into quadrants until each quadrant is fully covered by an NLC. For each quadrant, an *upper* and a *lower bound* of a number of NLCs that intersect with that quadrant are computed. In each iteration, a quadrant with the highest *upper bound* (which is more likely to become a part of the optimal region) is further partitioned into four quadrants. The process continues until the upper and the lower bound of the quadrants converge.

Liu et al. [7] presented an approach called MaxSegment to reduce the search space by transforming the optimal region search problem to the optimal interval search problem in a one-dimensional space. The authors use a plane sweep-like method to find the optimal interval. Finally, the optimal interval is transformed back to the optimal region in the original two-dimensional space and returned as the result.

Lin et al. [6] presented the OptRegion algorithm to solve the MaxBRkNN problem. The key idea is to index the set of facilities with a kd-tree to find the k nearest facilities of each object $o \in O$, and obtain an NLR (region enclosed by NLC) for each o. A *sweepline* algorithm is employed to generate the intersection lists for every NLR by traversing a "line" along the y-coordinates. The optimal region is obtained from the NLRs containing the maximal intersection point.

The existing solutions rely on the fact that the set of objects is static, and most of the solutions construct an index over the objects (or the NLCs). Therefore, these solutions are not easily extendable to the streaming object scenario.

Distance aggregation queries. Qi et al. [11] have explored the optimal location selection query, which finds a location for a new facility that minimizes the average distance from each customer to its closest facility. An *influence set* to manage a potential location p that includes customers for whom the nearest facility distance is reduced if a new facility is established at p. A similar problem was explored in other work [4, 10, 14, 15, 18] which try to find a location for a new facility such that the maximum distance between the facility and any customer is minimized. Papadias et al. [9] found a location that minimizes the sum of the distances of a facility placed in that location from the customers. These queries focus on an aggregation (such as average or summation) over the distances of the optimal location from the objects. All of these approaches only consider static objects and do not directly address the streaming MaxBRkNN problem explored in this paper.

2.2 Facility location selection queries on dynamic objects

Ghaemi et al. [3] studied the MaxBRkNN problem for moving objects and facilities in road networks, but the solution can only work when $k = 1$ (the nearest facility). Their approach relies heavily on pre-computation, and uses multiple lookup tables to answer the queries online. Specifically, for each o, they store all edges (or parts of edges) with a distance less than or equal to its nearest facility. The set of these edges is denoted as the "local network" of o. The information of whether an edge or a part of an edge belongs to the local network of multiple objects is also stored. When the location of an object is updated, the most promising edges that could be the optimal location for a new facility are obtained using the pre-computed information. Additionally, three atomic operations to support complex movement operations are proposed. An assumption for these pre-computed local networks is that, an object can move only to a neighboring location. In contrast, as a streaming object can arrive anywhere in the space (or along any edge), the local network cannot be pre-computed for that object. Thus it is not easy to extend this method to solve our problem.

Table 1: Related Work on MaxBRkNN

Study	Input		Distance
	O	P	
[6, 7, 12, 13, 19]	Static	Static	Euclidean
[3]	Moving	Moving	Network
Our work	**Streaming**	**Static**	**Euclidean**

Table 1 summarizes the existing related work and their problem settings. Our work can also be used for other distance functions. Although the contributions of the existing work are important, there exists a research gap between these approaches and some real-life applications. The MaxBRkNN problem has not been explored previously in the streaming object setting, and the existing methods are not easily extensible for our problem.

3 Problem Formulation

Let P be a set of static facilities, where each $p \in P$ is defined as a pair $(p.lat, p.lng)$, representing its geo-spatial location. Let O be a stream of tuples $\langle o.lat, o.lng, o.t\rangle$ in the order of their arrival time $o.t$, where each item represents a customer $o \in O$ and $(o.lat, o.lng)$ represents its geo-spatial location.

We adopt the sliding window model where an object (customer) is valid while it belongs to the current sliding window W. The window size can be specified by time, count, and the update size as the number of insertions and deletions between two consecutive windows [5]. A time-based window contains the objects whose arrival time is within $|W|$ most recent time-slots, where Δw can be different. In a count-based window, $|W|$ and the update size (slide size) Δw are constant. The window contains the $|W|$ most recent data objects where the window updates for each new Δw ($1 \leq \Delta w \leq |W|$) object arrivals. A small value of Δw represents real-time updates, where a larger Δw depicts micro-batching of objects. When the context is clear, we use the terms 'customer' and 'object' interchangeably. Let O_n be the set of customers inserted, and O_o be the set of customers expired from the current window W. Before defining our problem, we first present the necessary preliminaries.

k Nearest Neighbor Circle (c). Let kNN (o) be the k nearest neighbor facility of a customer o. The k nearest neighbor circle c_o is a circle with the location of o as the center and the distance between o and kNN (o) as the radius. Let C be the set of kNN circles of all of the customers in the current window.

Intersection Circles Set (IS). Given a kNN circle c_o of a customer o, the intersection circle set IS_o is the set of circles that contain or intersect with c_o.

Maximal Intersection Point ($s^{\uparrow}$). As there is at least one intersection point when any two circles overlap, let s_o be the intersection point in circle c_o with the largest number of overlapping circles from C, and η_s be the number of the circles overlapping with s_o. Let $s^{\uparrow}$ be the intersection point with the largest η_s, and $\eta^{\uparrow}$ is its corresponding number of overlapping circles.

Definition 1. (Maximal Intersection Region, R) Given C, we define the maximal intersection region R such that (i) For $\forall\ r \in R$, $|\mathrm{R}k\mathrm{NN}(r, P)|$ is maximal, where r is a point location; (2) For $\forall r$, $r' \in \mathrm{R}$, $\mathrm{R}k\mathrm{NN}(r, P) = \mathrm{R}k\mathrm{NN}(r', P)$.

Problem Statement. Given a set P of static facility locations, a stream of customers O, a positive integer k, and a sliding window model on O, the continuous MaxBRkNN problem on a stream is to continuously update the Maximal Intersection Region, R for the customers valid in the updated window. There may exist multiple Maximal Intersection Regions.

4 Algorithm

In this section we propose the following different solutions to address the MaxBRkNN problem on streaming objects: (i) As there are multiple studies that address the MaxBRkNN query for static objects, first we apply one of the approaches directly to solve the problem on streaming objects (customers) as our baseline. (ii) The baseline is originally designed for static objects and does not reuse any computation for the streaming objects. Instead, we propose an optimized solution where computations are shared among the consecutive windows as much as possible. (iii) We further propose two more optimizations: (a) *safe interval*, and (b) pruning of objects that cannot update the result from the previous window, on top of our proposed solution to improve overall efficiency.

4.1 Baseline Algorithm

We adopt the solution, OptRegion, proposed by Lin et al. [6] for our baseline, as they have shown that their solution consistently outperforms two other state-of-the-art solutions (MaxOverlap [12] and MaxFirst [19]). The OptRegion solution applies the principle *region-to-point transformation* [12] to find the intersection point overlapping with the maximum number of circles. If such a maximal intersection point $s^{\uparrow}$ is found, then the maximal intersection region R (the result of the MaxBRkNN query) can be easily obtained from the circles overlapping with $s^{\uparrow}$.

As the OptRegion algorithm is proposed for static objects, every time the sliding window updates, we invoke the algorithm in our baseline. Algorithm 1 shows the pseudocode of the baseline. Here, the set of the facilities P is indexed using a kd-tree. The index is initially built before processing any queries. The input of the algorithm is the set of all objects in the initial window W and the kd-tree over P. Whenever the window updates, the steps of the OptRegion algorithm are executed. There are three main steps in the algorithm:

1. **Find the kNN circle c_o:** For each object o in the window, the kd-tree over P is used to find the k nearest neighbors of o. Then the c_o for each o (Lines 2-3) is constructed, where the center is the location of o, and the radius is the distance from o to its kNN.
2. **Find the set of intersecting circles:** For each circle c_o, the sweepline algorithm outlined in Algorithm 1 of Lin et al. [6] is used to determine the set IS_o of the circles intersecting with c_o. This sweepline algorithm scans

Algorithm 1 Baseline (W, kd-tree over P)

```
1: η↑ ← 1
2: for o in W do
3:     compute c_o of o using the kd-tree
4: for o in W do
5:     IS_o ← Set of circles overlapping with c_o by a sweepline algorithm
6:     for c_i in IS_o do
7:         update IS_i
8: Sort |IS| in descending order of |IS_o|
9: for IS_o in IS do
10:    if |IS_o| >η↑ then
11:        compute the exact η of c_o
12:        if η > η↑ then
13:            update η↑ and s↑
14:    else Break
15: find the intersection of all circles containing s↑ and update R
16: return R
```

along the y-coordinate of each circle that is valid in the current window from top to bottom to find IS_o. Let the highest and the lowest y-coordinate of a c_o be $y_o^{\uparrow}$ and $y_o^{\downarrow}$, respectively. When the sweepline reaches the $y_o^{\uparrow}$ of c_o, c_o is inserted in a status tree (AVL tree). However, when the bottom point $y_o^{\downarrow}$ is encountered, c_o is deleted from the status tree. This status tree, which is updated dynamically, saves the candidate IS_o of the current circle.

3. **Find the maximal intersection point $s^{\uparrow}$:** Here, the maximum number of circles overlapping with an intersecting point s of a c_o can be at most IS_o (the number of circles intersecting with c_o). The set IS_o of the objects are sorted and considered in descending order of cardinality. We maintain the maximum value $\eta^{\uparrow}$ of intersecting circles for any intersection point found so far. If the IS_o of any c_o is greater than $\eta^{\uparrow}$, the actual number of intersections for c_o is computed by sweeping around the perimeter of c_o and finding the intersection point s with IS_o. The value of $s^{\uparrow}$ and $\eta^{\uparrow}$ are updated if necessary.
4. Finally, the optimal region R is constructed as the intersecting region of all of the circles overlapping with the point $s^{\uparrow}$, and R is returned as the result. The process is repeated when the window updates again.

Example 2. Figure 2 illustrates an example of the initial process of the baseline. For ease of presentation, we omit the set P from the figure. Let A and B be the top point and bottom point of c_1, respectively. The five states of the sweepline of c_1 are l_1, l_2, l_3, l_4 and l_5. In the l_1 state, the AVL tree has two nodes: c_1 and c_4. In l_2, c_4 is removed from the AVL tree. For l_3, the top point of c_2 is encountered, and c_2 is inserted into the AVL tree. In l_4, c_3 is inserted in the tree. And finally for l_5, the sweepline reaches the bottom point B, and c_1 is deleted. We then stop updating IS_1. We choose the union of circles (except itself) under these states as the upper bound of IS_1, which is $\{c_2, c_3, c_4\}$. Correspondingly, we also need to update the IS of each circle c_i in IS_1, for example, c_1 should be added to IS_4. Note that $IS_1 = \{c_2, c_3, c_4\}$, c_4 does not intersect with c_1 since they do not have any common intersection point. We sort $|IS_o|$ in descending order to

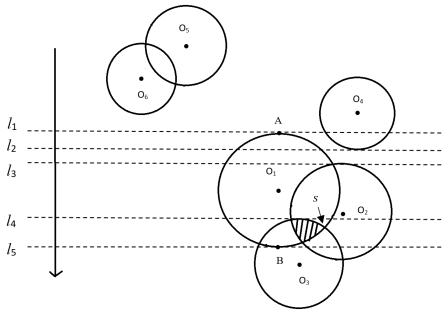


Fig. 2: An example of the initial process.

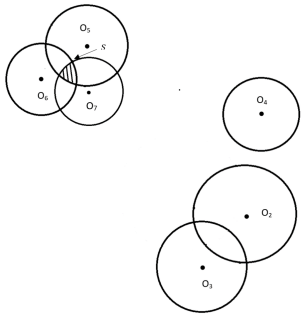


Fig. 3: An example of the first slide.

get: $|IS_1| = 3$, $|IS_2| = 2$, $|IS_3| = 2$, $|IS_4| = 1$, $|IS_5| = 1$, $|IS_6| = 1$. *We traverse* c_1 *with the largest* $|IS_o|$*, and* s *(shown in the figure) is returned because it has the largest* η*, which equals to* 3*. Then we update* $\eta^{\uparrow}$ *to* 3 *and* $s^{\uparrow}$ *to* s*. The next iteration results in early termination as* $|IS_2| < 3$*. Finally,* R *is found (shown as the shaded area) as the area overlapped by* c_1*,* c_2 *and* c_3*, which overlaps with* $s^{\uparrow}$*.*

An example of the update phase for the first slide is illustrated in Figure 3 for $\Delta w = 1$*. From the example in Figure 2,* o_1 *is expired and* o_7 *arrives.We repeat a new computation from scratch to find the new* R *(shown as a shaded area in the example).*

Drawbacks of baseline algorithm. When the sliding window updates, the baseline algorithm repeats the process for all of the objects in W. However, for a count-based window, only Δw objects are inserted, where $W - \Delta w$ objects are common in the two consecutive windows. For a time-based window, $W - O_o$ objects are common in the two consecutive windows. Thus, the baseline algorithm requires a substantial number of repeated computations. Therefore, we propose the following refinements to the algorithm that only consider the inserted and expired objects (that are not common between two consecutive windows) when updating the optimal region.

4.2 Algorithmic Improvements

The key idea of our proposed algorithm is to share the computations between two consecutive windows whenever possible. After initializing the sliding window, the optimal region R is first obtained by any of the existing MaxBRkNN algorithms on static objects for the objects in the initial W. Then each time the window updates, our proposed optimization algorithm outlined in Algorithm 2 is called to update the result R. Here, the input of the algorithm is the previous window W', the current window W, and the kd-tree over P. Algorithm 2 consists of the following steps:

- The set of the newly inserted objects O_n and the expired objects O_o are obtained from the previous and the current window.
- **Updating only for the required objects:** In contrast to the baseline where the c_o of each object in the current window is computed, we compute the kNN circle c_o of only the objects in O_n using a sweepline algorithm,

Algorithm 2 Optimized Algorithm (W', W, kd-tree over P)

```
1: O_n ← W − W'
2: O_o ← W' − W
3: η ← 1
4: for each o in O_n do
5:     compute c_o of o using the kd-tree
6: for each o in O_n do
7:     IS_o ← Set of circles overlapping with c_o by a sweepline algorithm
8:     for c_i in IS_o do
9:         update IS_i
10: for each o in O_o do
11:     for c_i in IS_o do
12:         update IS_i
13: C ← Set of c_o for all o ∈ W
14: Lines 9 - 13 of Algorithm 1
15: Find the intersection of all circles containing s↑ and update R
16: return R
```

and compute the set IS_o of circles intersecting with each $o \in O_n$. The set of intersecting circles IS_o is also updated for each expired object $o \in O_o$ (Lines 10 - 12). Thus, instead of the steps in Lines 2 - 7 of Algorithm 2 where the computations are done for each object in W, the computations are now done only for the objects in O_n and O_o that are not common between W and W'.

- **Finding the maximal intersecting point, $s^{\uparrow}$:** We execute Lines 9 - 13 of Algorithm 1 to find $s^{\uparrow}$. In contrast to the baseline, we do not sort IS_o based on their cardinality again, as only a subset of them are likely to change.
- Finally, similar to the baseline, we compute the optimal region R from the point $s^{\uparrow}$ and return R.

Example 3. From the example in Figure 2 and Figure 3, let $\Delta w = 1$, o_1 expires, and o_7 arrives as a new object. First, we construct c_7 and compute $IS_7 = \{c_5, c_6\}$ for the new object using the sweepline algorithm. Then we update $IS_2 = \{c_3\}$ and $IS_3 = \{c_2\}$ because c_1 is expired, and then update $IS_5 = \{c_6, c_7\}$ and $IS_6 = \{c_5, c_7\}$ for the arriving object c_7. As $|IS_5|$ has the maximal value, then we sweep around c_5 to find the maximal intersection point s (shown in Figure 3). Finally, the previous $s^{\uparrow}$ is replaced by the new s, and the result region R (shown as the shaded area) is also updated. It is worth noting that we only need to compute the new circles (c_7 in this example) in our algorithm, but all of the valid circles require recomputations in the baseline algorithm.

Drawbacks of the optimization algorithm. Although Algorithm 2 improves the performance by reusing computations, it has the following drawbacks:

1. In Line 7, the sweepline algorithm outlined in Algorithm 1 by Lin et al. [6] is employed to determine the set IS_o of the intersecting circles for each object $o \in O_n$. This sweepline algorithm scans along the y-coordinate of each valid circle in the current window from top to bottom to find IS_o. However, when

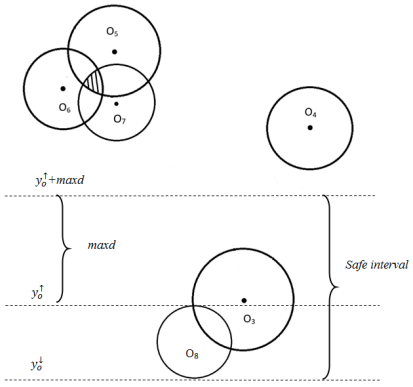


Fig. 4: Illustration of pruning rule 1.

only a subset of the circles change (inserted or expired), sweeping along the y-coordinate of each circle results in many unnecessary circle scans.

2. The value $\eta^{\uparrow}$ denotes the maximum number of circles overlapping with the intersecting point found so far. Each time $\eta^{\uparrow}$ is initialized as '1'. However, the optimal region R of the previous window, and thus the $\eta^{\uparrow}$ for the previous window can be decreased by at most Δw due to the expired circles.

In order to overcome these drawbacks, we propose two different pruning rules.

Pruning Rule 1: safe interval. The sweepline algorithm presented in Algorithm 1 by Lin et al. [6] scans the circles from top to bottom along the y-coordinate. In contrast, we want to determine a safe interval around the circle c_o such that any circle outside the safe interval cannot overlap with c_o, thus avoid scanning the unnecessary circles for c_o in the sweepline algorithm.

Let, the maximum diameter among the circles valid in the current window be $maxd$, and the highest and the lowest value in the y-coordinate of a c_o are $y_o^{\uparrow}$ and $y_o^{\downarrow}$, respectively. As the sweep is performed in a top-down manner, only the circles whose highest y-coordinate are within $y_o^{\uparrow} + maxd$ and $y_o^{\downarrow}$ can possibly overlap with the circle c_o. Thus we only need to scan the safe interval $y_o^{\uparrow} + maxd$ to $y_o^{\downarrow}$ in the sweepline algorithm for a circle c_o.

Example 4. In Fig. 4, c_8 is generated after the second slide. The circles c_4, c_5, c_6, c_7 are pruned as their top points are not within safe interval, while c_3 is added into IS_8.

Pruning Rule 2: using the result of the previous window. As shown in Line 10 of Algorithm 1 (which is also executed in Line 14 of Algorithm 2), we only need to compute the exact η of an object o if the corresponding $|IS_o|$ is greater than $\eta^{\uparrow}$. As only O_o circles can expire, the $\eta^{\uparrow}$ of the previous window (the maximum number of overlapping circles) can decrease by at most O_o. Thus, we can update the algorithm where additional input is the $\eta^{\uparrow}$ of the previous window, and initialize $\eta^{\uparrow}$ of the current window as $\eta^{\uparrow} - |O_o|$. If $\eta^{\uparrow} - |O_o|$ is less than '1', we initialize $\eta^{\uparrow}$ as '1'. For a count-based window, $|O_o| = \Delta w$.

5 Experimental Evaluation

In this section, we present the experimental evaluation for our solutions when continuously updating results for MaxBRkNN on sliding windows. In particular, we compare our proposed optimization solution (OP) with the baseline (BA), and further evaluate the benefit of applying each of the proposed pruning rules – pruning rule 1 (P1) and pruning rule 2 (P2) as proposed in Section 4.2.

5.1 Experimental Settings

All of our experiments were conducted on a Intel(R) Core(TM) i5-7200U CPU@2.50GHz processor and 8GB memory, running a Ubuntu 17.10 operating system. All algorithms were implemented in C++. The GCC version was 7.2.0.

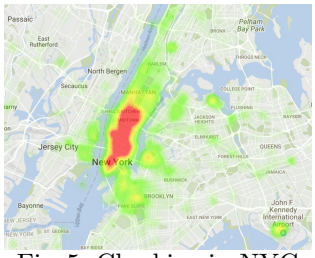


Fig. 5: Checkins in NYC

Data Source	Check-ins	Proportion
Foursq_NYC [16]	227,428	15.29%
Foursq_Global [17]	289,727	19.48%
LSSD [1]	710,827	47.79%
Gowalla [2]	139,171	9.36%
BrightKite [2]	120,359	8.18%

Table 2: Dataset: Check-in information.

Datasets. We conduct all experiments on an integrated real dataset, which is a combination of five different real check-in datasets. Table 2 shows a summary of all the datasets before integration. Each dataset contains the location of the points of interest (POIs) and the check-in locations of the users. The Foursq_NYC dataset collected from Foursquare contains POIs and check-in information of only New York City. The rest of the datasets consist of check-ins and POIs, where the locations are distributed all around the world. These datasets are collected from Foursquare [16, 17], the Location Sharing Services Dataset (LSSD) [1], Gowalla [2], and BrightKite [2].

We combine these datasets and take a subset, denoted as 'NYC', where the locations of the POIs and check-ins are all within New York City. We get 1,471,074 check-ins from mid April 2008 to mid September 2013 after duplicate removal, and 299,698 unique POIs in NYC. A graphical view of the check-in locations of this dataset is shown in Figure 5 as a heatmap.

To conduct the experiments, an area where the check-ins are located is chosen as a pre-defined percentage of the total area of the dataset. Note that, area size is a parameter of our experiments where the default area is 1%. We issued a range query with the area size in a random location. If the number of check-ins in that region is greater than 10% of the total check-ins, we denote that as a high density area. Similarly, we find mid and low density areas with check-in numbers – around 5% and 1% of the total check-ins. We report the average performance for 50 shifts of the sliding window for each setting.

Evaluation and parameterization. We study the efficiency of the baseline (BA), our proposed solution (OP), pruning rule 1 on top of OP (OPF), and both pruning rule 1 and pruning rule 2 together on top of OP (OPFS) as a function

Table 3: Parameters

Parameter	Symbol	Range
Window size	$\|W\|$	3000, 3500, **4000**, 4500, 5000
Slide size	Δw	1, 100, 200, **400**, 600, 800
No. of k nearest neighbors	k	2, 5, **10**, 20, 50
Density of check-ins	d	1%, 5%, **10%**
Area of check-ins	a	0.5%, **1%**, 2%, 4%

Table 4: Runtime comparison of BA and OPFS (ms).

Model	BA	OPFS
Micro-batch	1,429	189
Real-time	1,348	54

of varying parameters. The parameters and their ranges are listed in Table 3, where the default values are in bold. For all experiments, a single parameter is varied while the rest are fixed as their default.

Among the parameters, Δw denotes the number of objects updated in the window. Since Δw of a general count-based window is a fixed integer between 1 to $|W|$, in contrast to a time-based window where Δw varies, we present our experimental results for count-based window to better demonstrate the effect of varying a single parameter. However, all of our proposed approaches are applicable for both types of windows.

For each Δw-sized update of the window, we study the impact of each parameter on: (i) the total runtime, and (ii) the percentage of iterations pruned by P1 and P2 with respect to the OP. As a small value of Δw represents real-time updates, and a larger Δw depicts micro-batching of objects, we present our experimental results in two parts:

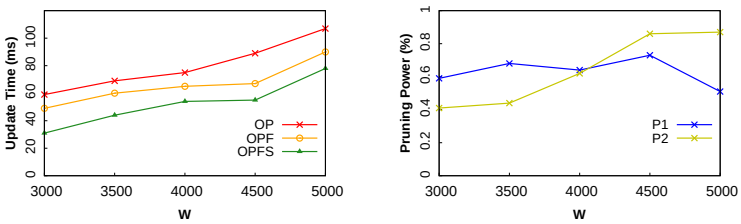
- Section 5.2.1 presents the real-time update experiments, where Δw is 1, and a single parameter other than Δw is varied while keeping the rest as default.
- Section 5.2.2 presents the experiment for micro-batching, where Δw is varied in a set of experiments from 1 to 800, and the default Δw is set as 400.

5.2 Performance Evaluation

We first compare the baseline (BA) and our proposed OPFS (which applies both pruning rules on top of our OP method) for the default parameter settings and show the runtime of both methods in Table 4. As BA applies an existing algorithm originally designed for static objects, the runtime of BA is about one to two orders of magnitude higher than OPFS for the streaming query. Due to this huge difference in performance, we exclude BA from the rest of the experimental evaluation as this is representative of the best case for the baseline algorithm, and compare the performance among OP, OPF, and OPFS only.

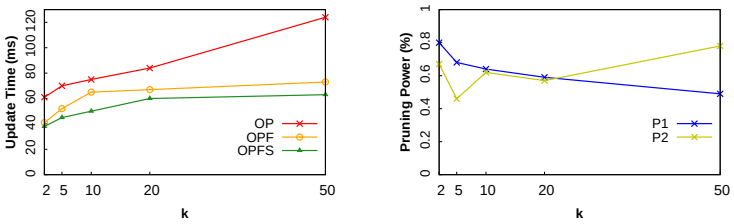
5.2.1 Real-time Processing ($\Delta w = 1$)

Varying $|W|$. Fig. 6a shows the effect of varying the size $|W|$ of a window. As more kNN circles are likely to intersect with each other as $|W|$ increases, the



(a) Runtime for update (b) Percentage of pruning

Fig. 6: Effect of varying $|W|$ with real-time processing.

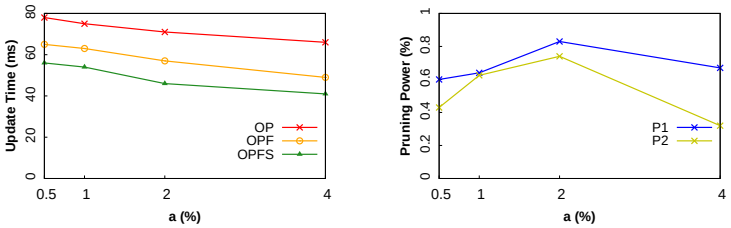


(a) Runtime for update (b) Percentage of pruning

Fig. 7: Effect of varying k with real-time processing.

runtime increases for all of the three methods as $|W|$ increases. As $P1$ reduces the number of circles to be checked while finding the intersection of the newly arrived objects, the percentage of pruning by $P1$ decreases for a higher $|W|$ (where there are more circles), as shown in Fig. 6b. In contrast, the percentage of pruning from $P2$ increases with $|W|$. The reason is that, although more circles may intersect with each other, there are not many intersection points that can become a candidate better than the previous window's result, thus more intersection points can be pruned from consideration.

Varying k. The radius of each kNN circle increases w.r.t. k. As the radius increases, more circles intersect, and thus the runtime increases with k in all of the approaches (Fig. 7a). The runtime of the OP approach increases rapidly for $k > 20$, where the other two methods do not vary much. As more circles intersect, the percentage of pruning by $P1$ decreases with the increase of k, as shown in Fig. 7b.



(a) Runtime for update (b) Percentage of pruning

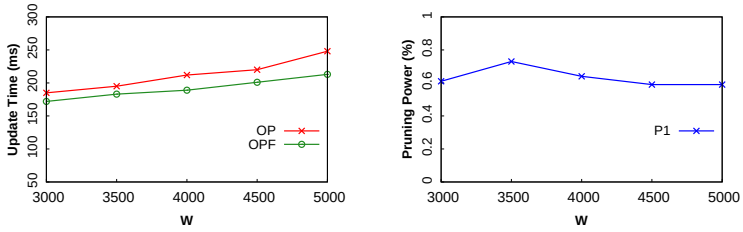
Fig. 8: Effect of varying a with real-time processing.

Table 5: Varying d with real-time processing.

Density	Update Time (ms)				Pruning Power	
	BA	OP	OPF	OPFS	$P1$	$P2$
1%	904	61	52	45	0.60	0.29
5%	1,225	74	61	54	0.67	0.58
10%	1,348	76	65	57	0.64	0.63

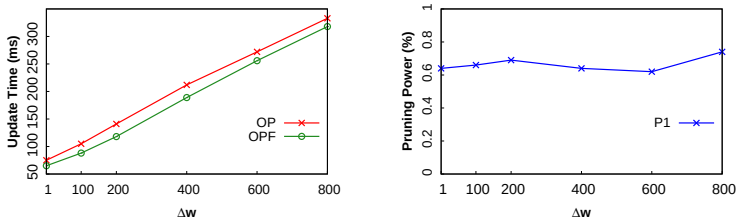
Varying a. We vary the size of the area where the check-ins are located as a percentage of the total area of the dataset. A higher percentage of the area (a) denotes that locations of the objects in a window are sparser. As the chances of circles intersecting with each other decrease as their density becomes sparser, the runtime decreases (Fig. 8a).

Varying d. Table 5 shows the performance of the four approaches for varying d. As more circles are likely to intersect with each other for a higher percentage of density, the runtime increases for each method. The runtime of BA is around one to two orders of magnitude higher than the other methods. As the result region is likely to have a higher number of intersecting circles for a higher density, $P2$ shows a greater benefit by pruning the intersecting points that cannot be better than the previous window's result.



(a) Runtime for update (b) Percentage of pruning

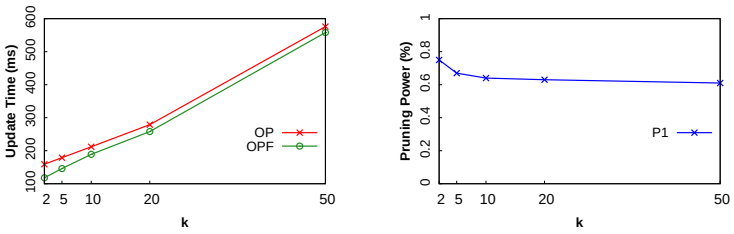
Fig. 9: Effect of varying $|W|$ with micro-batching.



(a) Runtime for update (b) Percentage of pruning

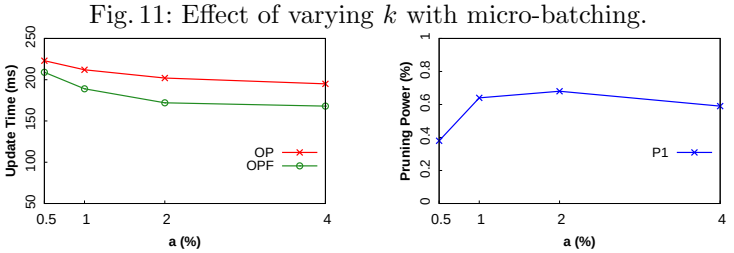
Fig. 10: Effect of varying Δw with micro-batching.

5.2.2 Micro-batching ($\Delta w \geq 1$) Here, we show the performance when varying different parameters in a more general sliding window setting, where the Δw can be greater than or equal to 1. The runtime and the percentage of pruning by varying each parameter are shown in Fig. 9a - 12b. The trends are mostly similar to the previous set of experimental evaluations, except that $P2$ is more effective for real-time processing. This is because $P2$ relies on the difference that can happen to the result of the previous window, i.e., $\eta^{\uparrow} - \Delta w$, but this



(a) Runtime for update (b) Percentage of pruning

Fig. 11: Effect of varying k with micro-batching.



(a) Runtime for update (b) Percentage of pruning

Fig. 12: Effect of varying a with micro-batching.

value becomes smaller for a larger value of Δw, thus the pruning effect is reduced in larger windows. As the rest performance results are similar, we focus on the effect of varying Δw in this section.

Varying Δw. A larger Δw denotes that a smaller number of objects are shared between two consecutive windows. As shown in Fig. 10a, the runtime is proportional to the increase of Δw, as more objects that are not common among the windows (updated objects) need to be processed.

Table 6: Varying d with micro-batching.

Density	Update Time (ms)				Pruning Power
	BA	OP	OPF	OPFS	$P1$
1%	1,408	186	157	157	0.64
5%	1,429	198	164	164	0.69
10%	1,489	212	189	189	0.72

6 Conclusion

In this paper, the problem of maximizing bichromatic reverse k nearest neighbor queries on streaming geo-data is introduced for the first time. We proposed an efficient solution where results are incrementally updated by reusing computations from the previous result. Our solution can work on both the count-based and the time-based sliding window models, thereby supporting both real-time processing and micro-batch processing. Extensive experiments based on real datasets have been conducted to verify the efficiency of our solution.

Acknowledgements. This work was partially supported by ARC DP170102726, DP180102050, and NSFC 61728204, 91646204. Zhifeng Bao is supported by a Google Faculty Award.

References

[1] Z. Cheng, J. Caverlee, K. Lee, and D. Z. Sui. Exploring millions of footprints in location sharing services. *ICWSM*, 2011:81–88, 2011.

[2] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *SIGKDD*, pages 1082–1090, 2011.

[3] P. Ghaemi, K. Shahabi, J. P. Wilson, and F. Banaei-Kashani. Continuous maximal reverse nearest neighbor query on spatial networks. In *GIS*, pages 61–70, 2012.

[4] J. J. Cardinal and S. Langerman. Min-max-min geometric facility location problems. In *EWCG*, pages 149–152, 2006.

[5] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker. No pane, no gain: Efficient evaluation of sliding-window aggregates over data streams. *SIGMOD Rec.*, 34(1):39–44, 2005.

[6] H. Lin, F. Chen, Y. Gao, and D. Lu. Optregion: Finding optimal region for bichromatic reverse nearest neighbors. In *DASFAA*, pages 146–160, 2013.

[7] Y. Liu, R.-W. Wong, K. Wang, Z. Li, C. Chen, and Z. Chen. A new approach for maximizing bichromatic reverse nearest neighbor search. *Know. and Inf. Sys.*, 36 (1):23–58, 2013.

[8] K. Mouratidis, S. Bakiras, and D. Papadias. Continuous monitoring of top-k queries over sliding windows. In *SIGMOD*, pages 635–646, 2006.

[9] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group nearest neighbor queries. In *ICDE*, pages 301–312, 2004.

[10] J. Qi, Z. Xu, Y. Xue, and Z. Wen. A branch and bound method for min-dist location selection queries. In *ADC*, pages 51–60, 2012.

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

[12] R. C.-W. Wong, M. T. Özsu, P. S. Yu, A. W.-C. Fu, and L. Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. *PVLDB*, 2(1):1126–1137, 2009.

[13] 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 lp-norm in two and three-dimensional spaces. *PVLDB*, 20(6):893–919, 2011.

[14] D. Yan, Z. Zhao, and W. Ng. Efficient algorithms for finding optimal meeting point on road networks. *PVLDB*, 4(11):968–979, 2011.

[15] D. Yan, Z. Zhao, and W. Ng. Efficient processing of optimal meeting point queries in euclidean space and road networks. *Knowl. Inf. Syst.*, 42(2):319–351, 2015.

[16] D. Yang, D. Zhang, V. W. Zheng, and Z. Yu. Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. *Transactions on SMC*, 45(1):129–142, 2015.

[17] D. Yang, D. Zhang, and B. Qu. Participatory cultural mapping based on collective behavior data in location-based social networks. *TIST*, 7(3):30:1–30:23, 2016.

[18] P. Zhang, H. Lin, Y. Gao, and D. Lu. Aggregate keyword nearest neighbor queries on road networks. *GeoInformatica*, pages 1–32, 2017.

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