Geoinformatica (2013) 17:63–95
DOI 10.1007/s10707-011-0149-0

Continuous aggregate nearest neighbor queries

Hicham G. Elmongui · Mohamed F. Mokbel · Walid G. Aref

Received: 3 February 2009 / Revised: 22 October 2011 /
Accepted: 14 November 2011 / Published online: 27 December 2011

Abstract This paper addresses the problem of continuous aggregate nearest-neighbor (CANN) queries for moving objects in spatio-temporal data stream management systems. A CANN query specifies a set of landmarks, an integer k, and an aggregate distance function f (e.g., min, max, or sum), where f computes the aggregate distance between a moving object and each of the landmarks. The answer to this continuous query is the set of k moving objects that have the smallest aggregate distance f. A CANN query may also be viewed as a combined set of nearest neighbor queries. We introduce several algorithms to continuously and incrementally answer CANN queries. Extensive experimentation shows that the proposed operators outperform the state-of-the-art algorithms by up to a factor of 3 and incur low memory overhead.

Keywords Continuous query · Spatio-temporal query · Aggregate nearest neighbor

This work is supported in part by the National Science Foundation under Grant Numbers IIS-0811954, III-1117766, IIS-0964639, IIS-0811935, CNS-0708604, IIS-0952977 (NSF CAREER) and by a Microsoft Research Gift.

H. G. Elmongui (✉)
Department of Computer and Systems Engineering, Faculty of Engineering,
Alexandria University, Alexandria 21544, Egypt
e-mail: elmongui@alexu.edu.eg

M. F. Mokbel
Department of Computer Science and Engineering, University of Minnesota - Twin Cities,
200 Union Street SE, Minneapolis, MN 55455, USA
e-mail: mokbel@cs.umn.edu

W. G. Aref
Department of Computer Science, Purdue University, 305 N. University St., West Lafayette,
IN 47907, USA
e-mail: aref@cs.purdue.edu

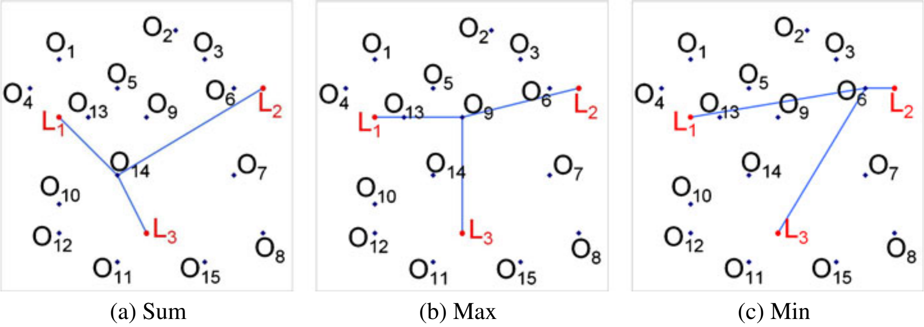
1 Introduction

The widespread of location-detection devices makes it possible to exploit new functionalities that result in new personalized services based on user locations. These new functionalities go way beyond the simple object finder services that are represented by either simple range or nearest-neighbor queries. In this paper, we focus on one of these new functionalities, in particular, the continuous aggregate k-nearest-neighbor queries. While traditional nearest-neighbor queries aim to find the k-nearest objects to only one certain point, the *aggregate* k-nearest-neighbor queries aim to find the k-nearest objects to a set of multiple points. In that sense, the query answer to the *aggregate* k-nearest-neighbor query needs to be defined through an *aggregate* to define how an object is considered close-by to the set of multiple points.

In general, the aggregate k-nearest-neighbor query problem can be formulated as follows: Given two sets of data points O and L, find the closest k points from O to all points in L based on a certain aggregate function f. Figure 1 illustrates the use of three different aggregate functions that may be used to define the query answer. In this figure, L_1, L_2, and, L_3 are the *set of multiple* points L that we need to compute their aggregate k-nearest objects among the set of points O that includes objects O_1 to O_{15}. Figure 1a–c represent the cases where the aggregate function f is *sum*, *max*, and *min*, respectively. For simplicity, we consider $k = 1$. In this case, Fig. 1a considers the *sum* nearest neighbor query where O_{14} is the object whose sum of distances to L_1, L_2, and, L_3 is minimal. On the other side, Fig. 1b considers the case of maximum nearest-neighbor query where O_9 is the object whose maximum distance to any of L_1, L_2, and, L_3 is minimal. Finally, Fig. 1c depicts the case of minimum nearest-neighbor query where O_6 is the object whose minimum distance to any of L_1, L_2, and L_3 is minimal.

The term "aggregate nearest-neighbor query" is coined in the literature in the context of spatial databases [29, 36] to refer to the case of several parties that look for a meeting point while minimizing a certain aggregate function with respect to all parties in the Euclidean space [29] and road networks [36]. Both proposed solutions in [29, 36] focus only on the case of snapshot queries, i.e., they do not maintain the result in case of location updates.

In this paper, we overcome the limitations and overhead of re-evaluating previous snapshot-based approaches upon location updates. Indeed, we consider the problem



(a) Sum (b) Max (c) Min

Fig. 1 Examples of aggregate nearest neighbor queries

of *continuous* k-aggregate nearest-neighbor queries (CANN) where we aim to *continuously* find the closest *k moving objects* from the set O to a set of stationary points L—the landmarks. The set of moving objects O are continually changing their locations. We aim to provide an incremental algorithm in which we maintain an initial answer with slight overhead rather than reevaluating the query with each movement update. Our problem setting has several real life applications. For example, consider a franchise with L locations that wants to send k e-coupons every few minutes to a set of close-by customers among all customers O who are driving their vehicles. It is of interest to the franchise to carefully select these k customers as the *current* close-by customers to the L locations. The close-by customers can be represented as either sum, minimum, or maximum distance. As an illustration, consider that L_1, L_2, and L_3 in Fig. 1 are the locations of a certain franchise that aims to send one e-coupon. The coupon can be sent to that customer with minimum *sum*, *max*, or, *min* distance to the three franchise locations. It is up to the franchise to choose any of the three aggregate functions as any of them will make sense in terms of choosing the best k customers. Thus, we aim to incorporate the three aggregate functions in our approach.

Our proposed algorithms for the continuous k-aggregate nearest-neighbor queries rely on combining the aggregate computation with the sorting and top-k selection in a way that limits the expensive aggregate computations and sorting to only few objects that have the potential to be part of the answer, i.e., top-k. In this context, we propose two algorithms, namely, H-CANN and P-CANN. The first algorithm is a holistic algorithm; it takes into account all landmarks when it decides whether or not to prune an object from being part of the answer. On the other hand, P-CANN is a best-effort progressive algorithm that eagerly prunes moving objects, even during the computation of the aggregate distance of the moving objects to the landmarks. P-CANN exploits some interesting geometric properties of the aggregate distance functions to assign a threshold to each landmark. These assignments impose an optimization problem, which is to have an interesting landmark order. We give several heuristics to retrieve the best order, and show through experiments the best policy to use for each aggregate function. P-CANN is a "best-effort" algorithm, it might only produce the k' aggregate nearest neighbors, where $k' < k$. In the experimental section, we show that, on the average, $k' \approx 0.95k$.

None of the proposed algorithms need to recompute the query answer to maintain the query answer continuously. In fact, all the proposed algorithms update the query answer on the fly whenever a location update is made available from the underlying infrastructure.

The contributions of this paper can be summarized as follows:

- We propose two algorithms (H-CANN and P-CANN) to retrieve the aggregate-nearest neighbors of a set of landmarks in a moving object database.
- We introduce several policies (first-fit, best-fit, and worst-fit) to determine the order of landmarks upon assigning pruning thresholds to each landmark in P-CANN.
- We perform an extensive performance evaluation of the proposed algorithms, and state the cases in which each algorithm is performing best. Experimental results show a factor of up to 3 improvement in performance over existing state-of-the-art algorithms.

The rest of the paper is organized as follows. Section 2 highlights related work. Section 3 gives some preliminaries, including the system environment, the formal problem definition, and a classification of the aggregate distance functions. We propose H-CANN and P-CANN in Sections 4 and 5, respectively. In Section 6, we evaluate the proposed techniques through an extensive experimental study. We conclude the paper by a summary and final remarks in Section 7.

2 Related work

Over the last decade, there has been several works that address the issue of continuous queries over moving objects (e.g., [8, 9, 12, 14, 16, 17, 20, 22, 25, 26, 30–32, 35, 37]). However, most of these works focus on the case of range queries [8, 12, 14, 25] and nearest-neighbor queries [9, 16, 26, 27, 30, 32, 35, 37]. Recently, there has been increasing interest in supporting other kinds of continuous queries, e.g., density queries [13, 18], reverse-nearest-neighbor queries [19, 34], closest pair queries [33], medoid queries [33], and skyline queries [15]. Up to the authors' knowledge, the problem of continuous k-aggregate nearest-neighbors has not been addressed before.

Most related to our work are the group-nearest-neighbor query (GNN) and the aggregate-nearest-neighbor query (ANN). The GNN query retrieves the object whose sum of distances to group members is minimal (e.g., [21, 28]). The ANN query (e.g., [29, 36]) is a generalization of the GNN query: besides the sum function, other aggregate functions are considered such as the maximum and the minimum of the distances to group members. This problem has been investigated for both spatial data that uses Euclidean distances [29] and for road networks [36] where the distance between two points is the sum of the road segments that constitute the shortest path between these two points. The algorithms in [29] and [36] are *progressive* where one uses an iterator interface on top of them and call "getNext()" to retrieve the next aggregate nearest neighbor. However, they are *not* incremental. They work on snapshot queries. If the objects are to change their location, the query has to be recomputed from scratch. The proposed techniques are focused on continuous queries.

There has been not much work on the continuous version of the aggregate nearest neighbor problem. Conceptual partitioning algorithm (CPM), which is originally designed for the continuous monitoring of nearest neighbor queries, has been extended to the aggregate nearest neighbor queries [26]. Since we are comparing against CPM in the experimental section, we summarize CPM in the following section.

2.1 Conceptual partitioning algorithm

The conceptual partitioning algorithm (CPM) is an efficient method for monitoring continuous nearest neighbor queries. It assumes that the moving objects are indexed by a main-memory grid. The size of each grid cell is the same, say $\delta \times \delta$. Each grid cell is associated with the moving objects located therein.

The main module of CPM is the nearest-neighbor computation algorithm. For each submitted query, this module is invoked to compute the initial results of the query. CPM organizes the grid cells based on their proximity to the query region. This organization provides a natural order of the grid cells to be processed so as to minimize the probed cells during the search. The probed cells are referred to as the *influence region*.

With the arrival of the location updates from the moving objects, only these updates affecting the influence region are considered for the maintenance of the query results.

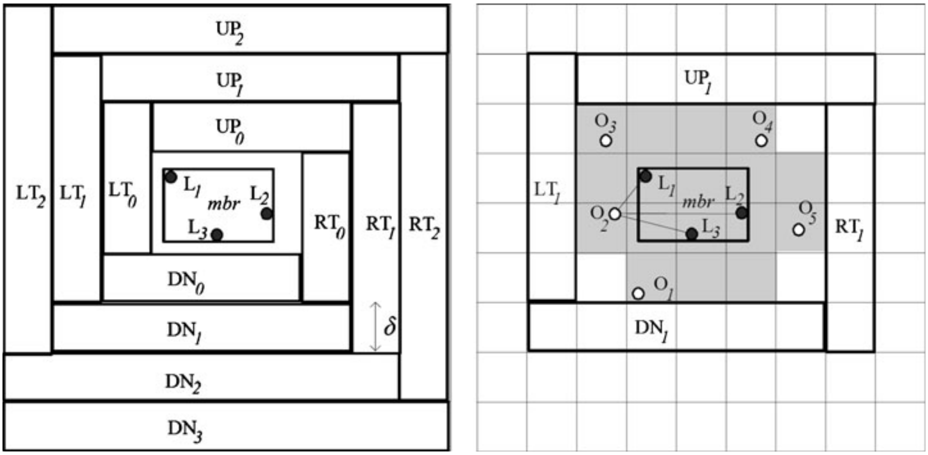
The CPM algorithm is extended to answer the aggregate nearest neighbor queries as shown in Fig. 2. The minimum bounding rectangle (*mbr*) of the query points $\{L_1, L_2, L_3\}$ is calculated, and the space is partitioned around it as rectangles as shown in Fig. 2a. After the initial aggregate nearest neighbor computation, the influence region is determined. Figure 2b shows the influence region for this query when the aggregate function $f = sum$. For details, the reader is referred to [26].

Both the CPM algorithm and our proposition use the Euclidean distance as the distance metric when computing the aggregate distance function.

2.2 The spatio-temporal histogram (ST-histogram)

Introduced in [11], the ST-histogram provides for the selectivity estimation of moving objects. Rather than examining and/or sampling all incoming location updates, ST-histograms are built and maintained by monitoring the actual selectivities of the outstanding continuous queries. The ST-histogram returns the selectivity of the objects located inside any polygon-shape.

The ST-histogram is a grid-based selectivity estimator. Each grid cell contains the selectivity of the objects residing within. Using a feedback from the query processor about the actual selectivity of a query, the grid cells are updated accordingly to reflect this feedback on the estimator.



(a) Partitioning into Rectangles (b) Influence Region

Fig. 2 Conceptual partitioning (CPM) algorithm

3 Preliminaries

3.1 System environment

For the rest of this paper, we assume an environment where each of the moving objects is equipped with a location detection device, e.g., a GPS. The moving objects will report their location periodically every time unit.

For the continuous query evaluation of spatio-temporal queries, there are two models of handling the location updates. (1) The updates are pushed into the query processor as soon as they are available (e.g., as in [24]). (2) The query processor pulls the current location of the objects for execution in order to update the query answer (e.g., as in [25]). In this paper, we assume the first model, where the query answer gets updated whenever location updates are made available to the algorithm.

3.2 Formal problem definition

The continuous aggregate nearest neighbor query is defined as follows. The query is defined with the following parameters: (1) A set of N landmarks $L = \{L_1, L_2, \ldots, L_N\}$, (2) a set of M moving objects $O = \{O_1, O_2, \ldots, O_M\}$, whose location updates arrive as a stream, and (3) an aggregate distance function f. The query continuously retrieves the object O_c whose aggregate distance according to L is minimum. The aggregate distance of any object O_i is denoted by D_i and is computed from the Euclidean distance between the moving object and all the landmarks. Therefore,

$$D_i = f_{k=1}^{N}(||O_i - L_k||)$$

3.3 Classification of aggregate functions

We classify the aggregate functions into two classes, A and B, based on their evaluators (defined below).

Definition 1 The evaluator E with order m of an aggregate function f (that takes a vector $\underline{X}$ as its parameter) is defined as the value of the aggregate function f on the first m components of $\underline{X}$, i.e.,

$$E(f(\underline{X}), m) = f(x_1, x_2, \ldots, x_m)$$

The evaluator E of a function f with order m is monotonically increasing with respect to m iff $m \geq m' \Rightarrow E(f, m) \geq E(f, m')$.

Definition 2 Class A aggregate function (e.g., sum and max) is an aggregate function whose evaluator is monotonically increasing with respect to its order. Class B aggregate function (e.g., min) is an aggregate function whose evaluator is not monotonically increasing with respect to its order.

The aggregate functions that are described in this paper are aggregate distance functions, where the aggregation occurs on the Euclidean distance between an object and each landmark. The first holistic algorithm, H-CANN, may be applied using any aggregate distance function. However, P-CANN works only for Class A aggregate distance functions. Class A aggregate distance functions have interesting properties that will be exploited in Section 5.

3.4 Loci of aggregate distance functions

Aggregate functions whose parameters are Euclidean distances from a point to a set of landmarks have interesting loci [29]. We use the properties of these loci in the design of P-CANN.

The loci of the points in space with the same sum aggregate distance form a closed shape. The landmarks form the foci of this shape. The resulting shape for sum is smooth; i.e., it is differentiable.

Similarly, the loci of the points in space with the same max aggregate distance form a closed shape. The landmarks form the foci of this shape. For *N* foci and for an aggregate distance *d*, the resulting shape is the intersection of *N* circles with radii *d*. The centers of these circles are the foci.

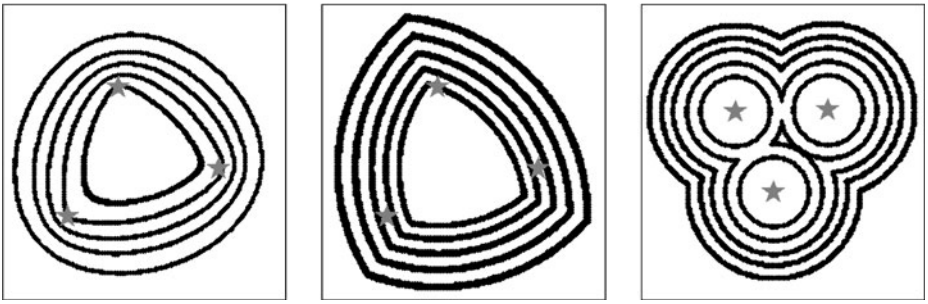
Nevertheless, the loci of the points in space with the same min aggregate distance form a shape that is not necessarily closed. The landmarks form the foci of this shape. The resulting shape may contain holes and it may be non continuous. Typically, for *N* foci and for an aggregate distance *d*, the resulting shape is the union of *N* circles with radii *d*. The centers of these circles are the foci.

These loci form a family of concentric curves. The number and location of the landmarks as well as the aggregate distance function determine the shape of the family of curves. For Class A aggregate distance functions, the loci are closed and convex. Moreover, the center of the convex shape is the centroid of the aggregate distance function.

Figure 3a–c show the families of curves for different aggregate distance functions when the number of landmarks is 3. An outer locus in these figures corresponds to a larger aggregate distance than an inner locus.

3.5 Frequently used symbols

Table 1 summarizes the primary symbols used throughout the paper.



(a) sum (b) max (c) min

Fig. 3 Family of equal aggregate distance shapes

Table 1 Frequently used symbols

Symbol	Description
k	The target answer set size
f	The aggregate distance function
O	The set of moving objects
L	The set of landmarks
M	The number of moving objects
$N = \|L\|$	The number of landmarks
O_i	The ith object
L_i	The ith landmark
d_i	The distance to the ith landmark
D_i	The aggregate distance function of the ith object
$O^{(k)}$	The kth aggregate nearest neighbor object
$D^{(k)}$	The aggregate distance function of $O^{(k)}$
m	The order of the aggregate distance function evaluator
$E(f, m)$	The evaluator of the aggregate distance function f with order m
R	H-CANN threshold: the safety margin
R_i	The threshold associated with the ith landmark
C_i	The locus associated with the ith landmark
r_i	The ith probed values in the unbounded binary search for R_i
Δ	The maximum distance between consecutive location updates of any object
δ	The grid cell length/width of the CPM algorithm

4 H-CANN: a holistic CANN algorithm

In this section, we present a holistic algorithm (H-CANN) that computes the CANN query. H-CANN gets its input as a stream of location updates of the moving objects. These locations are updated periodically. H-CANN is "holistic" in that it considers the aggregate distance to all landmarks in order to compute a threshold. This threshold will be used for deciding whether or not a moving object is part of the query answer. The output of H-CANN is a stream of the object identifiers that belong to the query answer at the current time period.

4.1 Algorithm overview

In this section, we present a high level description of H-CANN before we give the details subsequently.

H-CANN incrementally evaluates the continuous aggregate nearest neighbor queries. It keeps the current query answers in an *answer set*. The answer set is represented by a data structure called *HashedHeap* (described in Section 4.4). The *HashedHeap* is probed to get the aggregate distance of the kth nearest neighbor to the landmarks, which is used to compute a threshold, R, as given in Section 4.3.

The current answer set is updated according to the location updates of the moving objects. When a moving object, O_i, reports its location, its current location is used to compute its aggregate distance to all landmarks (D_i). If the answer set size is less than

k, this object is added to the query answer. Otherwise, the moving object's aggregate distance is compared against the threshold R. This comparison yields to the following situations:

If O_i does not belong to the answer set:

- If $D_i < R$, this object replaces the kth aggregate nearest neighbor. (case i)
- If $D_i \geq R$, this object is pruned. (case ii)

If O_i belongs to the answer set:

- If $D_i < R$, this object location is updated in the answer set. (case iii)
- If $D_i \geq R$, there might be another candidate moving object, O_j, that does not belong to the answer set but has an aggregate distance, D_j, such that $D_j < D_i$. (case iv)

We overcome the uncertainty resulting from case iv, by extending the answer set. The *extended answer set* will contain information about these candidate objects. Therefore, we can capture O_j as part of the answer set without any delays.

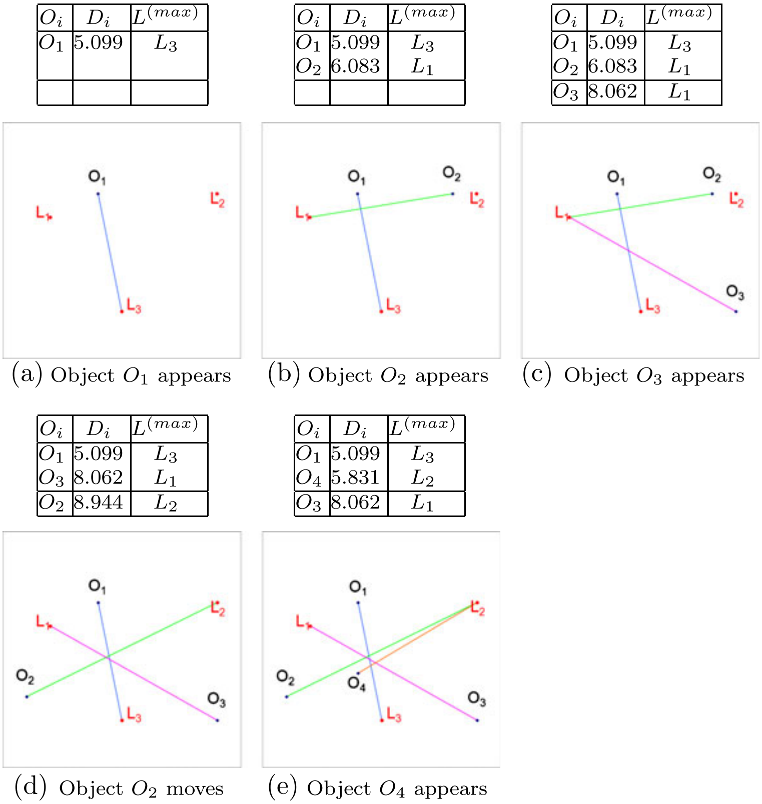
A moving object is candidate if there is a possibility that it becomes part of the answer set if it continues its motion with the same speed and direction. The *cache region* is the region that contains all the candidate objects. The *safety margin* is the threshold R that will be used to prune the objects that are outside the cache region. The computation of the cache region is given in Section 4.3. In the next section, we illustrate a running example to show how H-CANN uses the extended answer set to answer a continuous aggregate nearest neighbor query.

4.2 Running example

A running example is shown in Fig. 4a–e. The continuous aggregate nearest neighbor query Q is submitted to the system with $k = 2$. Q has 3 landmarks (L_1–L_3). The answer of Q should continuously report the current objects whose maximum distance to all three landmarks is minimal. The first five location updates are being illustrated. Each of these figures shows the execution of the algorithm upon the arrival of the location update of one moving object.

The algorithm keeps track of the distance of the farthest object (the kth top object) in the answer set from the landmarks. In fact, such object, referred to by $O^{(k)}$, is the head of the HashedHeap and $O^{(k)}$'s aggregate distance to all landmarks is $D^{(k)}$. The top set of tables in Fig. 4 shows the extended answer set after the update is handled. Each object is shown along with its aggregate distance to the landmarks. Only the first $k = 2$ objects represent the answer set. The other elements represent the cache region. A line is drawn between each moving object and its corresponding farthest landmark to show the aggregate distance: the maximum distance to the three landmarks for that object.

In this example, we assume that the maximum distance between two consecutive location updates for any moving object is bounded by $\Delta = 3$, which can be calculated from the maximum speed of the moving object and the frequency with which it is reporting its location. Therefore, the safety margin will be computed, as will be shown in Section 4.3, by the formula $R = D^{(k)} + \Delta$.



O_i	D_i	$L^{(max)}$
O_1	5.099	L_3

(a) Object O_1 appears

O_i	D_i	$L^{(max)}$
O_1	5.099	L_3
O_2	6.083	L_1

(b) Object O_2 appears

O_i	D_i	$L^{(max)}$
O_1	5.099	L_3
O_2	6.083	L_1
O_3	8.062	L_1

(c) Object O_3 appears

O_i	D_i	$L^{(max)}$
O_1	5.099	L_3
O_3	8.062	L_1
O_2	8.944	L_2

(d) Object O_2 moves

O_i	D_i	$L^{(max)}$
O_1	5.099	L_3
O_4	5.831	L_2
O_3	8.062	L_1

(e) Object O_4 appears

Fig. 4 Execution of H-CANN with 3 landmarks ($\{L_1, L_2, L_3\}$) and with an answer size $k = 2$ with a *max* aggregate-distance metric

Assume that we start with a fresh system, where the result set is empty and no location update has arrived yet. Upon the arrival of the first 2 location updates from O_1 and then O_2 (Fig. 4a and b), both objects will be inserted in the HashedHeap since the result set has not reached its capacity yet. Next, O_3 reports its location. The head of the HashedHeap is $O^{(k)} = O_2$, and the safety margin $R = D_2 + \Delta = 6.083 + 3 = 9.083$. O_3 will be stored in the cache region since $D^{(k)} < D_3 < R$.

Next, O_2 moves to a farther place and its location gets updated in the extended answer set. Therefore, O_3 is switched from the cache region to the answer set since $D_3 < D_2$. Notice that O_3 is the new head of the HashedHeap $O^{(k)}$. We compute the new safety margin, $R = D_3 + \Delta = 8.062 + 3 = 11.062$. Since the safety margin got increased, all elements existing in the cache region will remain in it.

Last, a new object O_4 reports its location ($D_4 = 5.831$). Because $D_4 < D_3$, O_4 makes it to the answer set and becomes the new head of the HashedHeap, pushing O_3 to the cache region. We recompute the new safety margin, $R = 5.831 + 3 = 8.831$. Then, we truncate the cache region by removing O_2 that has an aggregate distance larger than the safety margin.

H-CANN uses one threshold (the aggregate distance of the farthest object in the answer set) to prune the moving objects after it computes the aggregate distance.

In Section 5, we propose a progressive algorithm that associates a tight threshold for each landmark. These thresholds will be used to prune objects *during* the computation of the aggregate distance of the objects.

4.3 Computation of cache region

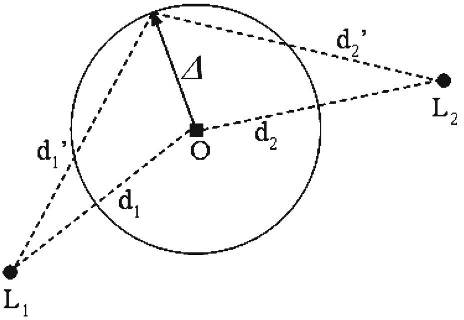
Assume that an object, say O_i, does not make it to the answer set because it has an aggregate distance larger than that of the head of the HashedHeap, say O_j. Assume that O_j moves and becomes farther from the landmarks than O_i. If we do not save the previous location update of O_i, O_i will not be able to make it to the answer set until O_i's next location update. We solve this problem by storing some moving objects in a cache region. In Fig. 5, the distance between two location updates of an object O is bounded by Δ, e.g., based on maximum object speed and the two consecutive time durations between the object updates. Therefore, the new distance of this object to any landmark (d'_i) is bounded by the sum of Δ and the old distance to this landmark, d_i, (triangle inequality). In other words, $d'_i < d_i + \Delta$. Therefore, the new aggregate distance (D'_i) might at most be larger than the old aggregate distance (D_i) by Δ in case of a min or max aggregate distance function; i.e., $D'_i < D_i + \Delta$. In the case of the sum aggregate distance function, the difference will be at most Δ times the number of landmarks, N; i.e., $D'_i < D_i + N * \Delta$.

If $D^{(k)}$ is the aggregate distance of the head of the *HashedHeap*, the safety margin, say R, will have the value of $R = D^{(k)} + \Delta$ in the case of max or min aggregate distance function. When the aggregate distance function is sum, the safety margin will be $R = D^{(k)} + N * \Delta$. An object O that does not qualify into the answer set will be inserted into the cache region if O's aggregate distance does not exceed the safety margin.

4.4 Data structures

We use a data structure, termed *HashedHeap*, to maintain the answer set of the proposed algorithms. A *HashedHeap* consists of a *descending* priority queue (a heap) and a hash table that hashes into the elements inside the priority queue. The objects are ordered in the priority queue according to the aggregate distance from a set of landmarks. The hash table provides a way to efficiently locate internal nodes in

Fig. 5 Cache region computation



the priority queue using the moving-object identifier. The head of the *HashedHeap*, which is the front of the priority queue, has the largest aggregate distance.

The answer set is extended with a cache region. The cache region stores moving objects that are not part of the answer set, but may become part of it in case a member of the answer set moves farther away from the landmarks. The cache region, which was previously used in [22], is stored in another data structure called *HashedList*. The *HashedList* consists of a sorted doubly linked list and a hash table that hashes into the elements inside the sorted doubly linked list using the moving-object identifier. The objects are ordered in an ascending order in the list according to the aggregate distance from a set of landmarks. The head of the *HashedList*, which is the front of the doubly linked list, has the smallest aggregate distance in the list. The tail of the *HashedList* has the largest aggregate distance in the list.

The extended answer set In the following, we refer to the union of the answer set (*HashedHeap*) and the cache region (*HashedList*) by the *extended answer set* (see Fig. 6). When we insert a new moving object O_i in the extended answer set, we try to insert it into the *HashedHeap* first. If the *HashedHeap* does not reach its capacity, say k, the object gets inserted successfully. Otherwise, we compare O_i with the head of the *HashedHeap*, $O^{(k)}$. The object that has a smaller aggregate distance will remain in the *HashedHeap*, whereas the other object will be inserted into the cache region, the *HashedList*. At any point in time, the cache region will be truncated according to the safety margin.

The same logic holds in the following cases: (1) if an object in the *HashedHeap* updates its location, it is compared with the head of the *HashedList*. (2) if an object in the *HashedList* updates its location, it is compared with the head of the *HashedHeap*. In these two cases, the object that has a smaller aggregate distance will remain in the *HashedHeap*, whereas the other object will be inserted into the cache region, the *HashedList*.

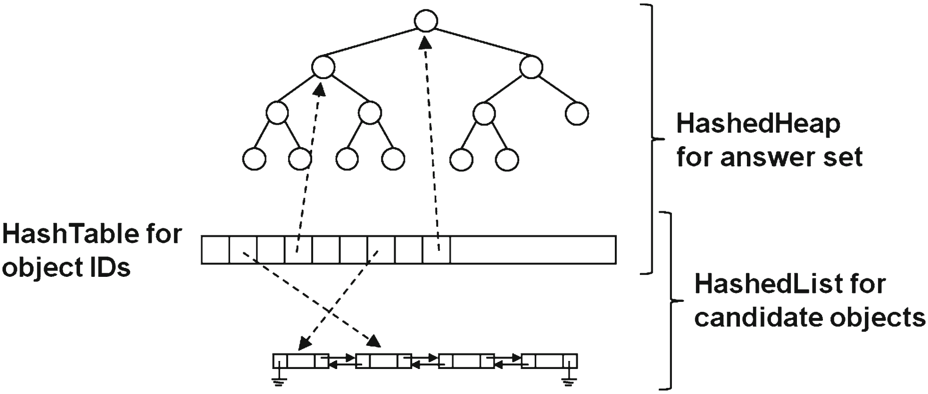


Fig. 6 The extended answer set

4.5 Algorithm details

H-CANN has three inputs. The first input is k, the answer set size. The second input is the set of moving objects that will report their location updates periodically. The third input is the set of landmarks to which the aggregate distance is computed. Algorithm 1 gives the pseudocode of H-CANN. Whenever a CANN query is issued, the extended answer set is initially empty (lines 2 and 3). For each arriving location update *loc*, the algorithm proceeds as follows (lines 5–23). Let the object O_i be the object whose location is *loc*. We compute its aggregate distance to all landmarks in line 7.

Algorithm 1 Holistic algorithm for CANN

```
 1: Function H-CANN($int\ k$, $MovingObjects\ O$, $Landmarks\ L$)
 2: $S$ is the extended answer set ($S := HashedHeap \cup HashedList$)
 3: $S = \phi$
 4: while the query is active do
 5:    look for a newly received location update $loc$
 6:    $O_i$ is the object whose location update is $loc$
 7:    $D_i \leftarrow$ the aggregate distance of $O_i$ to $L$
 8:    if $O_i \notin S$ then
 9:       $R \leftarrow$ the safety margin
10:       if $|S| < k$ or $D_i < R$ then
11:          $S = S \cup \{O_i\}$
12:          if $O_i = S.HashedHeap$.head then
13:             $R \leftarrow$ re-compute the safety margin
14:             truncate $S.HashedList$
15:          end if
16:       end if
17:    else
18:       update $O_i$'s location in $S$
19:       $R \leftarrow$ re-compute the safety margin
20:       if $S.HashedHeap$.head has changed or $D_i > R$ then
21:          truncate $S.HashedList$
22:       end if
23:    end if
24: end while
```

First, consider when O_i does not exist in the extended answer set (lines 9–16). We compute the safety margin (line 9). We have two cases: (1) the extended answer set size is less then k, and (2) the aggregate distance of O_i to all landmarks is smaller than the safety margin. In these two cases, O_i is added into the extended answer set (line 11). This means that O_i is either added to the *HashedList* (for Case 2) or to the *HashedHeap* (for Case 1) (possibly pushing an object from the *HashedHeap* to the *HashedList* as described before). In Case 1, if O_i becomes the head of the HashedHeap, we recompute the safety margin, and possibly truncate the HashedList accordingly (lines 12–15).

Next, consider when O_i already exists in the extended answer set (lines 17–23). We update the location of O_i in the extended answer set S (line 18). This update may incur moving its position within either the *HashedHeap* or the *HashedList*, or even across them. Then, we re-compute the safety margin in line 19. If the head of the HashedHeap has changed, we truncate the HashedList (line 21).

5 P-CANN: a progressive algorithm for CANN

In this section, we propose P-CANN, a best-effort progressive algorithm for computing and maintaining the answer of CANN over a period of time. P-CANN is designed for Class A aggregate functions. Having a monotonically-increasing evaluator for the aggregate function is necessary for the correctness of this algorithm that has superior performance than H-CANN (see Section 6).

P-CANN is a *best-effort* algorithm. It achieves great improvement in performance. However, it might produce *the* k' aggregate nearest neighbors, where $k' < k$. In Section 6, we show that P-CANN may produce about 95%–98% of the target answer size.

P-CANN is a progressive algorithm. A threshold is associated with each landmark. This threshold is used to prune the moving objects from being part of the query answer. In contrast to H-CANN that has one threshold for pruning the objects, P-CANN has a threshold per landmark that provides tighter bounds for pruning.

We start with an overview of P-CANN in Section 5.1. Next, we give the P-CANN algorithm that answers a continuous aggregate nearest neighbor query in Section 5.2. Finally, we show how to compute the thresholds associated with each landmark in Section 5.3.

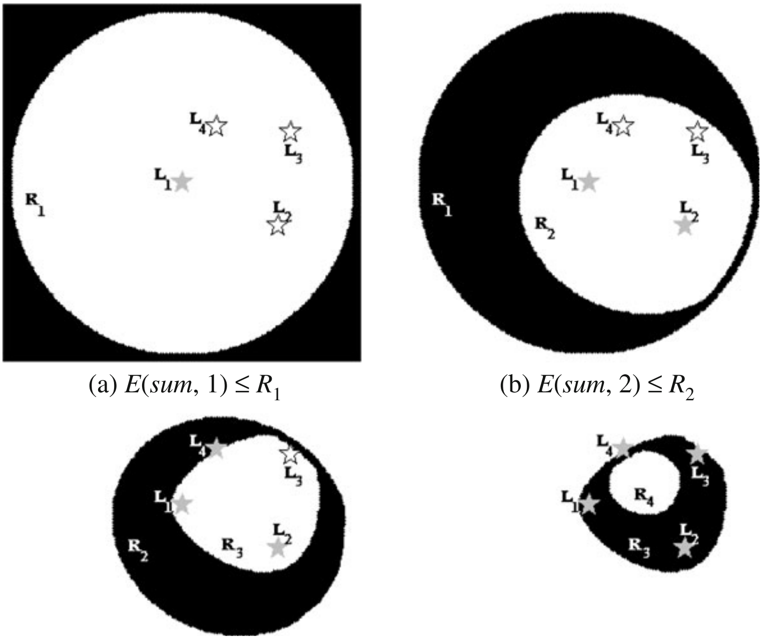
5.1 Overview of P-CANN

In this section, we present an overview of the logic of P-CANN. First, a suitable ordering of the landmarks is determined and a threshold is associated with each landmark. The thresholds are computed such that they are used to prune the objects *during*, and not *after*, the evaluation of the aggregate distance function. The landmarks are ordered in such a way to reduce the distance computations. The details of computing the thresholds and figuring out a good landmark ordering are given in Sections 5.3 and 5.4, respectively.

We illustrate, in Fig. 7a–d, the search space and its progressive reduction using a running example of four landmarks L_1, L_2, L_3, L_4 and a *sum* aggregate distance function. The thresholds associated with these landmarks are R_1, R_2, R_3, R_4, respectively. In each of the figures, the black-colored region represents the domain of the objects with an evaluator $E(sum, i)$ that has a value larger than R_i. The enclosed white-colored region is the domain of the objects with an evaluator $E(sum, i) \leq R_i$.

First, the evaluator with Order 1 is computed and compared against R_1. An object is pruned during the evaluation of its aggregate distance if it lies in a black-colored region. Otherwise, we continue computing the evaluator with the higher order. This process continues until the evaluator with Order N is reached (number of landmarks is $N = 4$). The CANN query answer consists of objects with an evaluator $E(sum, 4) \leq R_4$.

Algorithm 2 shows how the thresholds are used within the computation of the aggregate distance (COMPUTEAGGDISTORPRUNE), which will return infinity if the object may be pruned. Typically, after updating the running aggregate value (line 7),



(a) $E(sum, 1) \leq R_1$

(b) $E(sum, 2) \leq R_2$

(c) $E(sum, 3) \leq R_3$

(d) $E(sum, 4) \leq R_4$

Fig. 7 Progressive reduction of P-CANN search space. The *silver stars* represent the foci of the locus of $E(sum, i) \leq R_i$

Algorithm 2 Compute aggregate distance for P-CANN

```
1: Function ComputeAggDistOrPrune(MovingObject $O_i$, Landmarks $L$, Thresholds $R$)
2: $f$ is the aggregate function
3: $aggValue = 0$
4: $N = |L|$
5: for $k = 1 \cdots N$ do
6:     $d \leftarrow$ distance from $O_i$ to $L_k$
7:     $aggValue \leftarrow f(aggValue, d)$
8:     if $aggValue > R_k$ then
9:         /*$aggValue = E(f, k)$*/
10:        return $\infty$
11:    end if
12: end for
13: return $aggValue$
```

the running aggregate value is checked against a threshold associated with the current landmark. If the running value exceeds the threshold, the object is pruned without the need to continue its aggregate distance computation (lines 8–11).

5.2 P-CANN algorithm

Algorithm 3 gives the outline of P-CANN. The algorithm maintains the answer set in a *HashedHeap*. This set is initialized at the beginning of the algorithm (lines 2–4).

First, we start by optimizing P-CANN. This optimization consists of computing the thresholds that will be assigned to each landmark in line 5, and will be used to prune the moving objects. The algorithm proceeds by looking for the newly arriving location updates of the moving objects. The thresholds assigned to each landmark are used to prune objects upon the computation of their aggregate distance from the landmarks (line 9). For an object that belongs to the answer set, if it is to be pruned, it is removed from the *HashedHeap* (lines 11–13). However, if the object is not pruned, its location will be updated in the answer set (lines 15 and 16).

Algorithm 3 Progressive algorithm for CANN

```
 1: Function P-CANN($int\ k$, $MovingObjects\ O$, $Landmarks\ L$)
 2: $S$ is the HashedHeap containing the answer set
 3: $N = |L|$
 4: $S = \phi$
 5: $R[1 \cdots N] \leftarrow$ ComputeThresholds($L$)
 6: while stopping condition not met do
 7:    look for a newly received location update $loc$
 8:    $O_i$ is the object whose location update is $loc$
 9:    $D_i \leftarrow$ ComputeAggDistOrPrune($O_i$, $L$, $R$)
10:    if $D_i = \infty$ then
11:       if $O_i \in S$ then
12:          remove $O_i$ from $S$
13:       end if
14:    else
15:       if $O_i \in S$ then
16:          update the location of $O_i$ in $S$
17:       else
18:          if $|S| < k$ then
19:             insert $O_i$ in $S$
20:          else
21:             $O_k$ is the furthest object in $S$ from $L$
22:             $D_k \leftarrow$ the aggregate distance of $O_k$ to $L$
23:             if $D_i < D_k$ then
24:                replace $O_k$ by $O_i$ in $S$
25:             end if
26:          end if
27:       end if
28:    end if
29: end while
```

On the other hand, for the non-pruned objects, the following occurs. If the answer set contains less than k objects, the object is added to the *HashedHeap* (lines 18 and 19). Otherwise, the aggregate distance of any such object will be compared with the aggregate distance of the farthest object in the answer set. The object with the smaller aggregate distance will dominate and will remain in the answer set. The other object gets pruned (lines 21–25).

Computing tight thresholds for the landmarks to prune many objects is essential for P-CANN to achieve high performance. This process involves two main steps. The first step is to determine the order of the landmarks that will be used in computing the aggregate function in ComputeAggDistOrPrune. The second step is computing the thresholds given the landmarks order determined in the first step. We start with the second step in Section 5.3, and defer the details of the first step to Section 5.4.

5.3 Computing the thresholds

5.3.1 Overview

The order of the landmarks that is used in the aggregate distance evaluator is reversed when we compute the thresholds. For N landmarks, R_N is computed before R_{N-1}, which in turn is computed before R_{N-2}, and so on. We highlight the main idea of such computation as:

– R_N is computed such that at least k moving objects have their aggregate distance less than R_N; i.e., their evaluator $E(f, N) < R_N$.
– For the threshold R_i, where $1 \leq i < N$: Any moving object, whose evaluator $E(f, i+1) < R_{i+1}$, must have its evaluator $E(f, i) < R_i$. Otherwise, the moving object will be incorrectly pruned in ComputeAggDistOrPrune.

Therefore, with the help of a selectivity estimator (details below), R_N may be calculated by searching for it such that the locus of $D_N = E(f, N) < R_N$ forms a shape that contains at least k objects. Moreover, for $1 \leq R_i < N$, R_i may be calculated by searching for it such that the locus of $E(f, i) < R_N$ forms a shape that contains the shape representing the locus of $E(f, i+1) < R_{i+1}$. Figure 8a–d illustrate the steps to compute the thresholds in our running example.

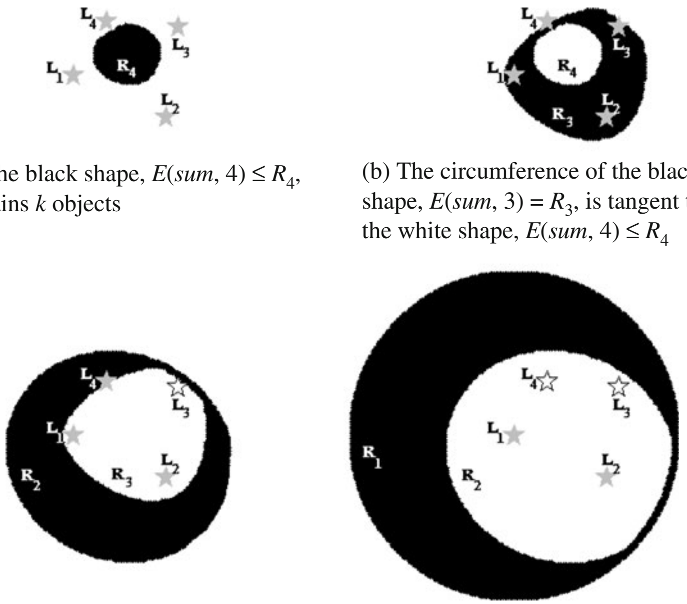
5.3.2 Running example

In line 7 of ComputeAggDistOrPrune, the threshold of any landmark L_i is compared against $f(d_1, d_2, \ldots, d_i)$, where d_i is the distance between the moving object and the ith landmark.

We show the details of the computation of the thresholds using two parallel example queries in the case of *sum* and *max* aggregate distance functions. Figure 9a and b show the loci of aggregate distance functions in the case of the landmarks $\{L_1, L_2, L_3, L_4\}$ for both functions, sum and max, respectively. The light gray curve, C_4, represents the locus of the points in space whose aggregate distance from the landmarks $\{L_1, L_2, L_3, L_4\}$ is R_4. The black curve, C_3, represents the locus of the points in space whose aggregate distance from $\{L_1, L_2, L_3\}$ is R_3. Similarly, the white curve, C_2, represents the locus of the points in space whose aggregate distance from $\{L_1, L_2\}$ is R_2. Finally, the dark gray curve, $C1$, represents the locus of the points in space whose aggregate distance from landmark L_1 is R_1 ($N = 4$ in this example).

5.3.3 Details: computing R_N

We start by searching for R_N using a spatio-temporal selectivity estimator called the ST-histogram [11]. The ST-histogram performs an unbounded binary search [6] to



(a) The black shape, $E(sum, 4) \leq R_4$, contains k objects

(b) The circumference of the black shape, $E(sum, 3) = R_3$, is tangent to the white shape, $E(sum, 4) \leq R_4$

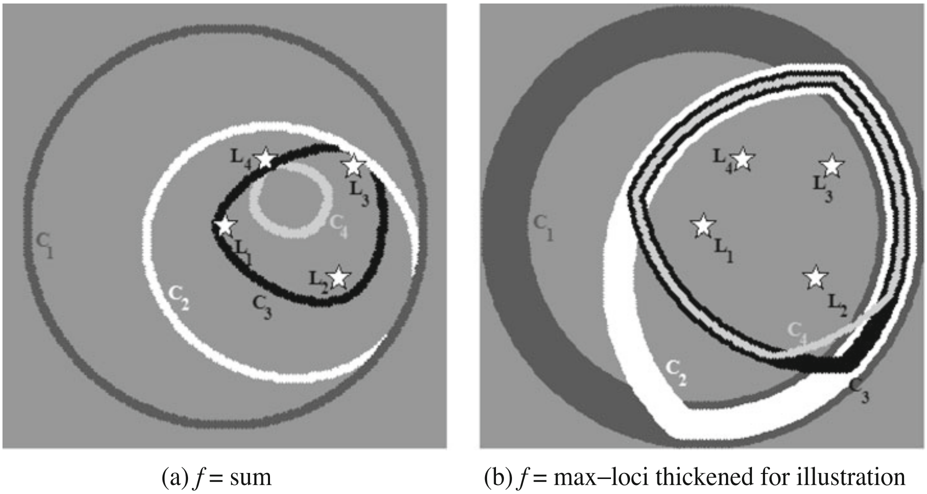
(c) The circumference of the black shape, $E(sum, 2) = R_2$, is tangent to the white shape, $E(sum, 3) \leq R_3$

(d) The circumference of the black shape, $E(sum, 1) = R_1$, is tangent to the white shape, $E(sum, 2) \leq R_2$

Fig. 8 Calculating the thresholds: overview

find out the value R_N of the aggregate distance function $f(d_1, d_2, \ldots, d_N)$ whose locus contains at least k objects. The unbounded binary search is a logarithmic algorithm. In this search, the current probed value of the aggregate distance defines the current probed locus. Figure 10 shows an example of applying the unbounded binary search to find a threshold R_N, where the probed values of the aggregate distance are $r_1, r_2, \ldots, r_6$.

The ST-histogram estimates the selectivity of the moving objects that exist in any polygon. A polygon approximating the locus of each aggregate distance function is sent to the ST-histogram to return its selectivity. The vertices of the polygon are sampled from the locus. For a Class A aggregate function, the center of the convex shape is the centroid of the aggregate distance function. We perform a rotational plane sweep from the centroid to get the samples. After each degree of rotational sweep, one sample is collected using another unbounded binary search starting from the centroid and in the direction of the sweep. Points that exist inside the locus will have a smaller aggregate distance value, whereas points outside the locus will have a larger aggregate distance value. A point P is a sample vertex of the polygon when P minimizes the distance between the aggregated value of the probed locus and P's aggregate distance on the segment from the centroid and in the direction of the sweep. The other vertices will be retrieved in the same way after rotating more degrees in the rotational sweep.



(a) f = sum (b) f = max–loci thickened for illustration

Fig. 9 Use of the loci of aggregate distance functions in computing the thresholds. C_i represents $E(f, i) = R_i$

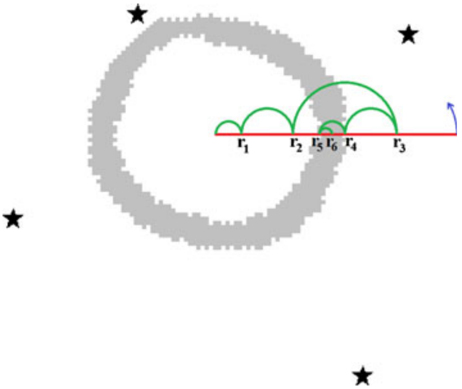
5.3.4 Details: computing R_i, $1 \leq i < N$

In the case where the aggregate distance function is max, the locus of the aggregate distance function is the intersection of N circles, all of which have the same radius. Therefore, the thresholds of the subsequent landmarks, $R_1, R_2, \ldots, R_{N-1}$, are the same as the threshold of the Nth landmark.

On the other hand, for the sum aggregate distance function, the thresholds of the subsequent landmarks are computed as follows in the order $R_{N-1}, \ldots, R_2, R_1$. First, the Nth landmark is eliminated and we compute R_{N-1} from the other landmarks. Next, the $(N-1)^{\text{st}}$ landmark is also eliminated and we compute R_{N-2} from the remaining landmarks, and so on.

To compute R_j, we perform an unbounded binary search [6] to find out the value of $R_i = f(d_1, d_2, \ldots, d_j)$ whose locus contains the locus of

Fig. 10 Unbounded binary search to compute R_N



$R_{j+1} = f(d_1, d_2, \ldots, d_{j+1})$. Consequently, R_j will not prune any moving object that passes R_{j+1}.

The loci of $f(d_1, d_2, \ldots, d_{j+1}) = R_{j+1}$ and of $f(d_1, d_2, \ldots, d_j) = R_j$ are approximated with polygons during the containment test.

For a convex polygon to be contained in another convex continuous locus, only the vertices of the first polygon need to be tested for containment inside the locus. The test for containment is simply to check the sign of the difference between the current threshold, R_j, and the aggregate distance function to the first j's landmarks, $f(d_1, d_2, \ldots, d_j)$. If the sign is positive, the vertex is inside the second locus, and vice versa.

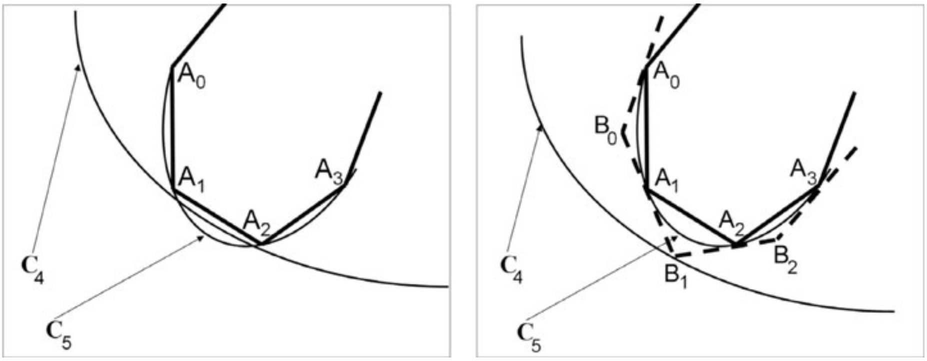
Figure 11a shows a containment test of a locus C_5 inside a locus C_4. This figure shows that the approximation of the locus C_5 using an inscribed polygon $A_0A_1A_2A_3\ldots$ may produce a false-positive containment test. This is why we use the circumscribing polygon that is formed by the tangents to the locus at the sample locus points. Figure 11b shows the containment test of C_5 inside C_4 using the circumscribing polygon $B_0B_1B_2\ldots$, which does not produce any false-positive tests.

We compute the tangents of the locus of the sum aggregate distance function analytically as follows. Let X be a sample point on the locus of $f(X) = \sum ||X - L_i||$. For any point $X = (x, y)$ and landmark $L_i = (L_i.x, L_i.y)$, the following equations are used to compute the slope of the tangent at X.

$$f(x, y) = \sum_{i=1}^{N} \sqrt{(x - L_i.x)^2 + (y - L_i.y)^2}$$

$$\frac{\partial f(x, y)}{\partial x} = \sum_{i=1}^{N} \frac{x - L_i.x}{\sqrt{(x - L_i.x)^2 + (y - L_i.y)^2}}$$

$$\frac{\partial f(x, y)}{\partial y} = \sum_{i=1}^{N} \frac{y - L_i.y}{\sqrt{(x - L_i.x)^2 + (y - L_i.y)^2}}$$



(a) Inscribed polygon $A_0A_1A_2A_3\cdots$ may produce false-positive containment test

(b) Circumscribing polygon $B_0B_1B_2\cdots$ does not produce false-positive containment test

Fig. 11 Approximating the locus of the curve for the containment test

$$\bigtriangledown f(x,y) = \left[\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right]$$

$$\frac{dy}{dx} = -\frac{\partial f(x,y)/\partial x}{\partial f(x,y)/\partial y}$$

In Section 6, we show that the time required to compute the thresholds and to figure out the order of the landmarks is very small.

5.4 Determining the order of the landmarks

The order of the landmarks in computing the aggregate distance function in ComputeAggDistOrPrune determines how many moving objects are pruned after computing the distance to each landmark. Different orderings lead to different loci, which lead into a different total number of distance computations. Notice that the moving objects that are located inside any locus have their aggregate distance within the corresponding threshold. Therefore, all moving objects are tested against R_1. Only those that are located inside the dark gray circle in Fig. 9 are tested against R_2; the others are pruned. Similarly, only those moving objects inside the white curve are tested against R_3, and so on. Consequently, the number of distance computations against any threshold is proportional to the selectivity of the enclosing locus.

As pointed out in Section 5.3, we need to figure out an elimination order for the landmarks. The N landmarks can be ordered in $N!$ different ways. However, it is prohibitively expensive to perform an exhaustive search on the order amongst the $N!$ to find the order producing the least total execution cost. We propose three elimination order policies to achieve a practical elimination order for the landmarks. We term the three policies are *first-fit*, *best-fit*, and *worst-fit* elimination orders.

Definition 3 The first-fit elimination order is an elimination policy in which, at each elimination decision, the first landmark in the remaining landmarks is chosen for elimination.

Definition 4 The best-fit elimination order is an elimination policy in which, at each elimination decision, the landmark, whose elimination results in a shape with smallest number of distance computations, is chosen for elimination.

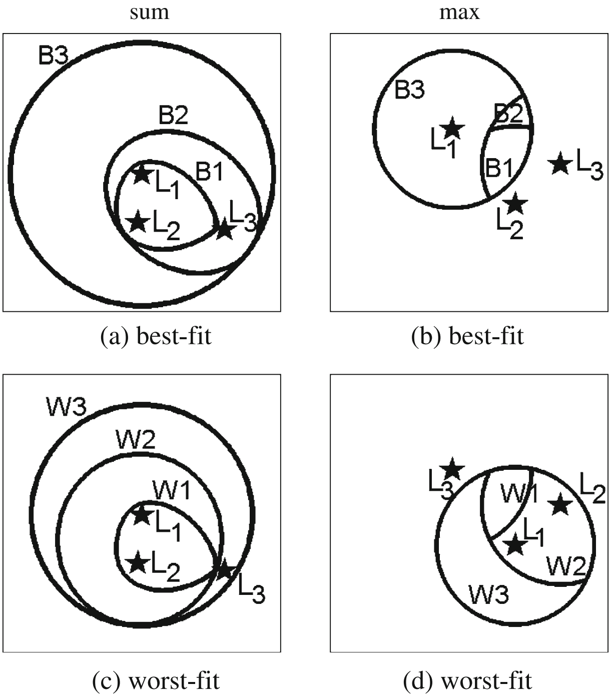
Definition 5 The worst-fit elimination order is an elimination policy in which, at each elimination decision, the landmark, whose elimination results in a shape with largest number of distance computations, is chosen for elimination.

The first-fit elimination order picks the next to-be-eliminated landmark in constant time, and thus the total time for determining the landmarks order is $O(N)$. On the other hand, the best-fit and worst-fit elimination orders pick the next to-be-eliminated landmark in a linear time (with respect to the number of landmarks), and hence require a total time for determining the order to be $O(N^2)$.

The rational behind the best-fit elimination policy is to try to locally minimize the number of distance computations for the current landmark given the already determined landmarks. Nevertheless, the rational behind the worst-fit elimination policy is to minimize the ratio of the number of distance computations performed against a landmark and the number of distance computations performed against the prior landmark.

For the sake of presentation, and without loss of generality, we assume in this paragraph that the moving objects are uniformly distributed in the space. For the general case, ST-histogram is consulted to get the selectivity of any locus as described earlier in the paper. Under such assumption, the number of distance computations against a landmark is proportional to the area of the locus associated with prior landmark. Figure 12a–d show the shapes corresponding to three landmarks L_1, L_2, L_3 when the best-fit and worst-fit elimination policies are adopted for both aggregate distance functions. In the best-fit elimination policy, the landmarks are eliminated in the order L_2 and then L_3 producing the loci $B1$, $B2$, and $B3$ respectively. Consequently, the thresholds corresponding to the landmarks L_1, L_3, and then L_2 are probed in this order. On the other hand, in the worst-fit elimination policy, the landmarks are eliminated in the order L_3 and then L_2. Therefore, the thresholds corresponding to the landmarks L_1, L_2, and then L_3 are probed in this order.

Fig. 12 Examples of different elimination orders for sum (**a**, **b**) and max (**c**, **d**)



(a) best-fit

(b) best-fit

(c) worst-fit

(d) worst-fit

5.5 Re-optimizing P-CANN

The P-CANN query optimization process results in the execution plan that will be used to evaluate the continuous aggregate nearest neighbor query answer. The output of the optimization consists of:

- The order of the landmarks that will be used in computing the aggregate distance of the moving objects.
- The thresholds associated with each landmark.

This optimization process depends on a selectivity estimator. Since CANN is a continuous query, the selectivities of the moving objects will change over time. Consequently, the query execution may be sub-optimal for two reasons: (1) There might be too many objects that move towards the region containing the query answer. Therefore, the pruning of many objects will be deferred during ComputeAggDistOrPrune until more landmarks are probed. (2) If many objects, whose aggregate distance is below R_N, move farther away from the region containing the query answer, the query answer size might be much less than the target k.

In these two cases, the P-CANN query needs to be re-optimized to get a better execution plan. The frequency upon which we will need to re-optimize depends on how the selectivities change over time. The first case is detected when the condition in line 23 of Algorithm 3 is invalid too frequently, which may be verified with a higher selectivity estimation of the query answer region. The second case is trivially detected when the query answer size gets much less than k for an extended period.

During the mid-query re-optimization, the execution for P-CANN is suspended. The query is then re-optimized before the execution is resumed with the new plan. P-CANN queries have a state associated with it: the *answer set* (*HashedHeap*). The new query evaluation plan will have the same state of the old plan. However, they differ in the thresholds associated with the landmarks, and possibly the order of the landmarks used in the aggregated distance computation.

Consider a continuous aggregate nearest neighbor query that runs for a period T without the need for re-optimization. In Section 6.5, we show that the optimization cost (in time units) is noticeably small compared to T. For instance, a query might remain optimal during a rush hour, continues to execute, and then needs to be re-optimized during a non rush hour.

6 Experimental analysis

We perform extensive experiments to evaluate the performance of the proposed algorithms. In these experiments, we compare the algorithms proposed in this paper with the state-of-the-art algorithm for the continuous aggregate nearest neighbor queries for moving objects; the conceptual partitioning (CPM) algorithm [26].[1] We

[1] The authors of conceptual partitioning kindly provided us with their code of CPM.

implemented the proposed algorithms in PLACE, a prototype spatio-temporal data stream management system [23]. We use the Network-based Generator of Moving Objects [7] to generate a set of moving objects. The generator generates 50,000 moving objects that move on the road network of Berlin, Germany for 100 time units with additional 1,000 objects per time unit. Other generator parameters are set to their default values. Moving objects can be vehicles, cyclists, pedestrians, etc. Table 2 summarizes the parameters under investigation, their ranges and their default values. For any experiment, we vary one parameter and set the other parameters to their default values. For all experiments, we use a Xenon 2.0 GHz CPU with 1 GB of RAM.

We generate 600 continuous queries. Each of these queries is defined by k, f, and the set of landmarks L. The landmarks are randomly selected with size $|L|$ from the dataset of work sites that is used in BerlinMOD, a benchmark for spatio-temporal database management systems [10].

In the experiments, we compare among the conceptual partitioning (CPM) algorithm, H-CANN, and P-CANN, where applicable. Our performance metrics are: (1) The throughput, i.e., the number of update input tuples processed per second (*better performance corresponds to a higher throughput value*). The average time to process an update input tuple is the reciprocal of the throughput. In a dynamic system where the moving objects report their locations frequently, the throughput shows how fast the system can respond to the rate of input tuples. (2) The optimization time. (3) Memory requirements. We also study how approximate P-CANN is; i.e., what the average output size of P-CANN is. For the CPM algorithm, we use a 128 × 128 grid as proposed in [26] as the good tradeoff between the CPU time and the space requirements. We show the experiments for the max and min cases. Since most of the experiments for the sum case give similar results to those of the MaxDist case, the performance results for SumDist are omitted for space limitation. We show the experimental results of SumDist only when there is a difference between the two cases.

6.1 Effect of number of landmarks

First, we study the effect of the number of landmarks $|L|$ on the performance of the proposed algorithms. Figure 13a gives the throughput (tuples processed per second) when the number of landmarks $|L|$ varies for the max aggregate function. The ratio of throughput of any of the proposed algorithms and the CPM algorithm is between 280% and 360%. Figure 13a gives the throughput of the three algorithms. P-CANN outperforms all other algorithms. When the set of landmarks gets larger,

Table 2 System parameters (ranges and default values)

Parameter	Range	Default
No. of landmarks: $\|L\|$	5, 10, 15, 20	5
Answer size: k	20, 40, 60, 80	20
Elimination order	Worst-ever-fit (WEF), first-fit (FF), best-fit (BF), worst-fit (WF)	FF

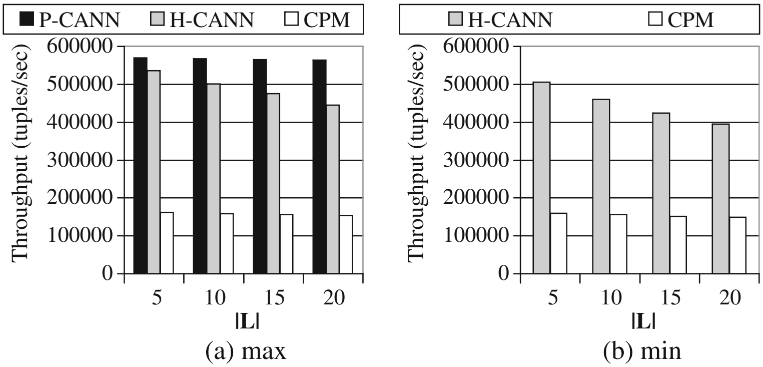


Fig. 13 Effect of $|L|$ on throughput

the throughput of all operators gets lower (around 24% for H-CANN and 1% for P-CANN). The ratio of throughput of H-CANN is at least 280% that of CPM.

P-CANN performs much fewer distance computations than H-CANN and the CPM algorithm. P-CANN allows for the progressive pruning of the search space without the need to compute the distance to all individual landmarks.

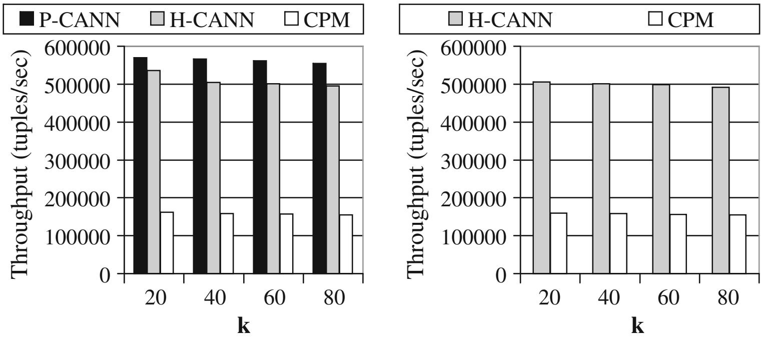
For the min aggregate function case, there is no P-CANN corresponding to this aggregate function since there is no monotonically increasing evaluator for min (see Section 3). The performance of the min case when the number of landmarks changes is given in Fig. 13b for the applicable techniques (H-CANN and CPM). In Fig. 13b, the throughput of H-CANN is more than 250% the throughput of the CPM algorithm. Both techniques get lower throughput when the number of landmarks gets larger. However, the throughput of the CPM algorithm decreases slower than the decrease in the throughput of H-CANN.

Apart from the trivial case of only few objects reporting their location, we notice from the experiments that the locus containing the query answer is usually much smaller than the minimum bounding rectangle of the landmarks. This is why the search space for CPM is much larger than the search space of P-CANN where most non-qualifying objects are pruned very early. Similarly, for the case of H-CANN, the data structures are not modified for almost all non-qualifying objects, in contrast with CPM where the grid index, with which the moving objects are associated, is maintained. Moreover, the CPM book-keeping information (and data structures) is modified with the updates while the influence region is much larger than the actual region containing the answer set.

It is worth pointing out that the CPM algorithm may be improved by reducing the influence region using similar techniques as in P-CANN. The influence region may be the region containing the answer set, which is computed as the innermost locus in Fig. 9.

6.2 Effect of the output size (k)

Figure 14a gives the throughput when the number of nearest neighbors k varies for the max aggregate function. The ratio of throughput of any of the proposed algorithms and the CPM algorithm is between 320% and 360%. Figure 14a gives



(a) max

(b) min

Fig. 14 Effect of *k* on throughput

the throughput of the three algorithms. P-CANN outperforms all other algorithms. As *k* gets larger, the throughput of the two proposed algorithms decreases by 2–4%. The ratio of throughput of H-CANN is at least 320% that of CPM.

P-CANN performs much fewer distance computations than H-CANN and the CPM algorithm. The number of distance computations for H-CANN is invariant to the answer size for a fixed number of landmarks. For any positive input update to H-CANN, the distance of the moving object to all landmarks is computed before any other processing. Moreover, for the CPM algorithm, the distance from any moving object in the influence region to all landmarks is computed when this object reports its location update. We have also discovered from the experiments that the influence region of the query is larger than the region contained in the loci of P-CANN. This is because the loci are aggressively computed such that they prune most of the objects that are not part of the answer as quickly as possible.

Figure 14b gives the throughput as a function of the answer size *k* for H-CANN and the CPM algorithm when the aggregate distance function is min. From Fig. 14b, the ratio of the throughput of H-CANN and the CPM algorithm is more than 300%. When *k* increases from 20 to 80, the throughput decreases by only 3% (see Fig. 14b). Nevertheless, the number of distance computations for H-CANN is invariant to the answer size similar to the sum and max cases.

From the experiments, we see that the performance of P-CANN is much better that the performance of H-CANN with the increase in the answer set size. We can see that P-CANN outperforms H-CANN with a factor of 126% when $k = 80$. P-CANN takes milliseconds for the optimization process, which directly affects the pruning strategies. Once the query is optimized, the execution of P-CANN is very fast as the pruning process itself is simply comparing a running aggregate distance against a threshold.

For the implementation effort of both algorithms, P-CANN uses a HashedList to maintain the answer set. H-CANN uses an additional data structure, HashedList, to maintain the cache region. The optimization of P-CANN involves many functions (e.g., an unbounded binary search, a test for polygon inclusion, optimizing multivariates functions) that are readily available in many libraries (e.g., OOL [1], LOPTI [2], NEWUOA [3], CONDOR [4], and Extreme Optimization [5]).

6.3 Actual output size of P-CANN

Since P-CANN depends on a selectivity estimator to compute the thresholds associated with each landmark, it is intrinsic that the output will be approximate. This approximation does not produce wrong answers. In fact, the query answer is the aggregate nearest neighbors of the landmarks. However, for a CANN query that asks for the k aggregate nearest neighbors, P-CANN may produce less than k if the selectivity estimator underestimates the selectivity of the inner locus in Fig. 9. If the selectivity estimator overestimates the selectivity, the k aggregate nearest neighbors are output.

Figure 15a gives the effect of this approximation with different values of requested k. We measure this effect by the average percentage of k that gets output of P-CANN. In the case of $k = 20$, the average value of the output size is 19.45; i.e., 97.25%. When $k = 80$, the average value of the output size is 76; i.e., 95%. The number of landmarks does not have a noticeable effect on the average percentage of k that gets output from P-CANN as shown in Fig. 15b. The standard deviation of the percentage of k that gets output does not exceed 4% and the obeserved minimum percentage does not fall below 88% for the cases of $k \in \{20, 40, 60, 80\}$ or $|L| \in \{5, 10, 15, 20\}$.

In Section 6.5, we show the low optimization cost that will be incurred if the application wants to re-optimize to get better values for the thresholds in case the selectivity estimator is refined and produces more accurate estimates over time.

6.4 Elimination order policies

Next, we study the performance of the various elimination order policies when using P-CANN (see Section 5.4). The optimal elimination order is the order that produces the least number of distance computations. For comparison purposes only, the optimal order is achieved using an exhaustive search on the $n!$ different elimination orders. This experiment computes the penalty incurred for using an elimination order for P-CANN other than the optimal order. With a 99% confidence level, the confidence interval for the relative additional distance computations when

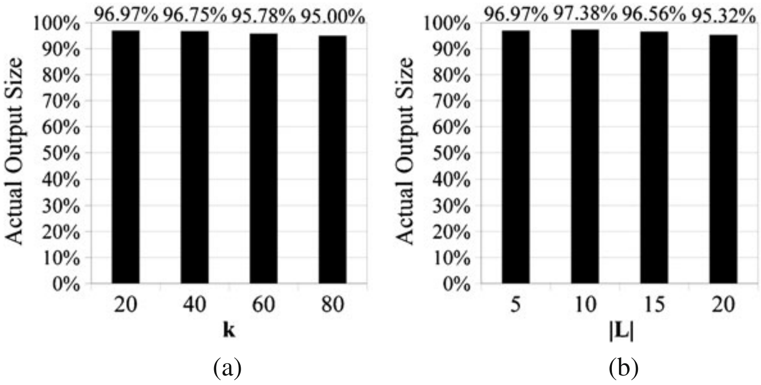


Fig. 15 Percentage of k that is output of P-CANN (sum)

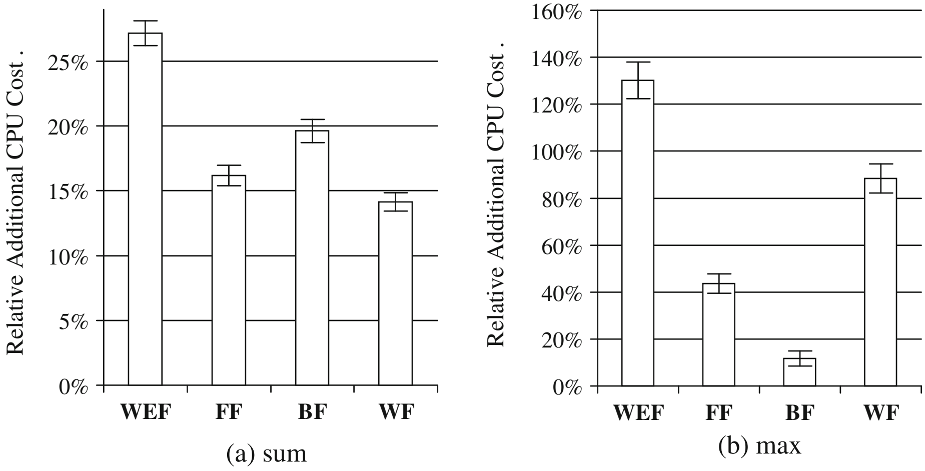
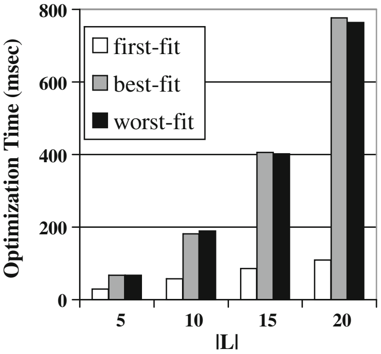


Fig. 16 Performance of different elimination order policies (99% confidence level)

using the first-fit, best-fit, worst-fit policies as well as the worst-ever-fit order is computed. The worst-ever-fit order is the elimination order that produces the worst number of distance computations. While searching for the optimal order, we also find the worst-ever-fit order. In this experiment, more than 3,600 random sets of landmarks are used. For any set, the size of the set is random, and the relative locations of the landmarks are also random. To make the results of this experiment independent of the moving objects behavior and selectivity in the city, we assume a uniform distribution of the moving objects. Under this assumption, the number of distance computations inside each operator, which is proportional to the selectivity of the moving objects inside the shape associated with the operator, turns out to be proportional to the area of that shape. Since it is prohibitively expensive to get the optimal order, a penalty is incurred when we use a sub-optimal elimination order.

Figure 16a gives the confidence interval for the additional distance computations for the sum case. The additional distance computations directly reflects on the additional CPU time required to process the location updates. The penalty is less than 27% in the worst case when we use the worst-ever-fit order. The first-fit

Fig. 17 Optimization time (sum)



elimination policy, which calls for the elimination of the first remaining landmark (in a random landmark order) incurs 16% more distance computations. The worst-fit elimination policy (despite its name) outperforms all the other policies in the figure and it only incurs 14% additional distance computations, whereas the best-fit elimination policy incurs 20% penalty.

Interestingly, the case for max is different. Figure 16b gives the confidence interval for the additional CPU cost for the max case. The worst-ever-fit order results in additional 130% CPU time over the optimal order. The first-fit elimination policy incurs 43% more distance computations. The best-fit elimination policy outperforms all the other policies in the figure and it only incurs 12% additional distance computations, whereas the worst-fit elimination policy performs badly and incurs a penalty of 88%.

6.5 Optimization cost

Figure 17 gives the time required for optimizing the P-CANN algorithm for sum aggregate distance function. This optimization consists of the time needed to retrieve the best elimination order and the thresholds corresponding to each landmark. Recall from Section 5.4 that the optimization cost using the first-fit elimination order policy is $O(N)$ and is $O(N^2)$ for either the best-fit or worst-fit elimination order policies, which is the price shown in this figure. For all elimination order policies, the cost of optimization is in milliseconds. It takes less than 120 ms for optimizing the query pipeline in the case of adopting the first-fit elimination order. This figure illustrates the applicability of the P-CANN in the domain of continuous queries.

7 Conclusion

In this paper, we proposed two algorithms to be used for continuously answering the aggregate nearest neighbor queries. H-CANN is a holistic algorithm. It decides whether to prune an object after computing its aggregate distance to all landmarks. P-CANN is a best-effort progressive algorithm that associates thresholds with the individual landmarks. These thresholds are used to eagerly prune the moving objects. Different elimination order policies are identified to specify the order of the landmarks in the computation of the aggregate distance in P-CANN. The optimization of P-CANN and how to assign the thresholds are addressed.

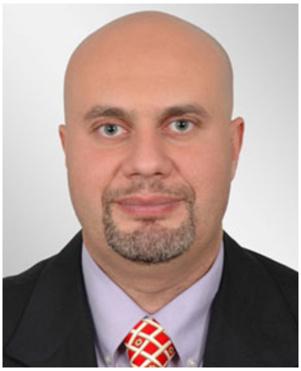
From the performed extensive experiments, we achieve cases whose performance is improved by up to a factor of 3 from the state-of-the-art algorithm. P-CANN outperforms both H-CANN and the CPM algorithm (the state of the art). For the optimization of P-CANNs, the worst-fit elimination policy gives the least penalty for sum (additional 14% CPU cost away from optimal) when we do not use the prohibitively expensive optimal order. On the other hand, the best-fit elimination policy gives the least penalty for the max case. The optimization time of P-CANN is about 100 msec for typical CANN queries. P-CANN, which is a best-effort algorithm might produce 95% of the required output size on the average.

As for future work, we will investigate the continuous aggregate nearest neighbor queries on moving objects in road networks. On the one hand, we would like to develop similar incremental algorithms using the road network distance instead of the Euclidean distance between the objects. On the other hand, we would like to make a first-class operator in a data stream management system such that the CANN operator is considered while optimizing the query evaluation plan.

References

1. Open Optimization Library. http://ool.sourceforge.net/
2. LOPTI - Mathematical Optimization Library. http://volnitsky.com/project/lopti/
3. SCILAB-NEWUOA Interface. http://www.inrialpes.fr/bipop/people/guilbert/newuoa/newuoa.html
4. Matlab-Condor. http://www.applied-mathematics.net/optimization/CONDORdownload.html
5. Extreme Optimization Numerical Libraries for .NET. http://www.extremeoptimization.com/
6. Bentley JL, Yao AC-C (1976) An almost optimal algorithm for unbounded searching. Inf Process Lett:5(3):82–87
7. Brinkhoff T (2002) A framework for generating network based moving objects. Geoinformatica 6(2):153–180
8. Cai Y, Hua KA, Cao G (2004) Processing range-monitoring queries on heterogeneous mobile objects. In: Proceedings of the international conference on mobile data management, MDM
9. Cho H-J, Chung C-W (2005) An efficient and scalable approach to CNN queries in a road network. In: Proceedings of the international conference on very large data bases, VLDB, pp 865–876, Trondheim, Norway
10. Düntgen C, Behr T, Güting RH (2009) BerlinMOD: a benchmark for moving object databases. VLDB J (The International Journal on Very Large Data Bases) 18(6):1335–1368
11. Elmongui HG, Mokbel MF, Aref WG (2005) Spatio-temporal histograms. In: Proceedings of the international symposium on advances in spatial and temporal databases, SSTD
12. Gedik B, Liu L (2004) MobiEyes: distributed processing of continuously moving queries on moving objects in a mobile system. In: Proceedings of the international conference on extending database technology, EDBT
13. Hadjieleftheriou M, Kollios G, Gunopulos D, Tsotras VJ (2003) On-line discovery of dense areas in spatio-temporal databases. In: Proceedings of the international symposium on advances in spatial and temporal databases, SSTD, pp 306–324, Santorini Island, Greece
14. Hu H, Xu J, Lee DL (2005) A generic framework for monitoring continuous spatial queries over moving objects. In: Proceedings of the ACM international conference on management of data, SIGMOD
15. Huang Z, Lu H, Ooi BC, Tung AK (2006) Continuous skyline queries for moving objects. IEEE Trans Knowl Data Eng (TKDE) 18(12):1645–1658
16. Iwerks GS, Samet H, Smith K (2003) Continuous K-nearest neighbor queries for continuously moving points with updates. In: Proceedings of the international conference on very large data bases, VLDB, pp 512–523, Berlin, Germany
17. Jensen CS, Lin D, Ooi BC (2004) Query and update efficient B^+-tree based indexing of moving objects. In: Proceedings of the international conference on very large data bases, VLDB
18. Jensen CS, Lin D, Ooi BC, Zhang R (2006) Effective density queries on continuously moving objects. In: Proceedings of the international conference on data engineering, ICDE, Atlanta, GA
19. Kang J, Mokbel MF, Shekhar S, Xia T, Zhang D (2007) Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In: Proceedings of the international conference on data engineering, ICDE, pp 806–815, Istanbul, Turkey
20. Lazaridis I, Porkaew K, Mehrotra S (2002) Dynamic queries over mobile objects. In: Proceedings of the international conference on extending database technology, EDBT
21. Li H, Lu H, Huang B, Huang Z (2005) Two ellipse-based pruning methods for group nearest neighbor queries. In: Proceedings of the ACM symposium on advances in geographic information systems, ACM GIS.
22. Mokbel MF, Aref WG (2005) GPAC: generic and progressive processing of mobile queries over mobile data. In: Proceedings of the international conference on mobile data management, MDM
23. Mokbel MF, Aref WG (2005) PLACE: a scalable location-aware database server for spatio-temporal data streams. Data Eng Bull 28(3):3–10

24. Mokbel MF, Aref WG (2008) SOLE: scalable online execution of continuous queries on spatio-temporal data streams. VLDB J (The International Journal on Very Large Data Bases) 17(5):971–995
25. Mokbel MF, Xiong X, Aref WG (2004) SINA: scalable incremental processing of continuous queries in spatio-temporal databases. In Proceedings of the ACM international conference on management of data, SIGMOD
26. Mouratidis K, Papadias D, Hadjieleftheriou M (2005) Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. In: Proceedings of the ACM international conference on management of data, SIGMOD
27. Mouratidis K, Yiu ML, Papadias D, Mamoulis N (2006) Continuous nearest neighbor monitoring in road networks. In: Proceedings of the international conference on very large data bases, VLDB, pp 43–54, Seoul, Korea
28. Papadias D, Shen Q, Tao Y, Mouratidis K (2004) Group nearest neighbor queries. In: Proceedings of the international conference on data engineering, ICDE
29. Papadias D, Tao Y, Mouratidis K, Hui CK (2005) Aggregate nearest neighbor queries in spatial databases. ACM Trans Database Syst (TODS) 30(2):529–576
30. Song Z, Roussopoulos N (2001) K-nearest neighbor search for moving query point. In: Proceedings of the international symposium on advances in spatial and temporal databases, SSTD
31. Tao Y, Papadias D (2003) Spatial queries in dynamic environments. ACM Trans Database Syst (TODS) 28(2):101–139
32. Tao Y, Papadias D, Shen Q (2002) Continuous nearest neighbor search. In: Proceedings of the international conference on very large data bases, VLDB
33. U LH, Mamoulis N, Yiu ML (2007) Continuous monitoring of exclusive closest pairs. In: Proceedings of the international symposium on advances in spatial and temporal databases, SSTD, Boston, MA
34. Xia T, Zhang D (2006) Continuous reverse nearest neighbor monitoring. In: Proceedings of the international conference on data engineering, ICDE, Atlanta, GA
35. Xiong X, Mokbel MF, Aref WG (2005) SEA-CNN: scalable processing of continuous K-nearest neighbor queries in spatio-temporal databases. In: Proceedings of the international conference on data engineering, ICDE
36. Yiu ML, Mamoulis N, Papadias D (2005) Aggregate nearest neighbor queries in road networks. IEEE Trans Knowl Data Eng (TKDE) 17(6):820–833
37. Yu X, Pu KQ, Koudas N (2005) Monitoring K-nearest neighbor queries over moving objects. In: Proceedings of the international conference on data engineering, ICDE



Hicham G. Elmongui (BS, MS: Alexandria University, 1998, 2001; MS, PhD: Purdue University, 2003, 2009) is assistant professor of Computer and Systems Engineering at Alexandria University, Egypt. He is also adjunct professor with the department of Electrical and Computer Engineering and the department of Computer Science at Virginia Tech, USA. His research interests are in query processing and optimization, location-based services, data security, and software engineering. His work has been published in major database conferences including VLDB, ACM SIGMOD, and IEEE ICDE and journals including ACM TODS. He is author of two books and has two patent applications pending with the United States Patent and Trademark Office. He was awarded several excellence and outstanding teaching and service awards from Purdue University, USA. He is member of IEEE, ACM, Upsilon Pi Epsilon, CERIAS Alumni, and Microsoft Research Alumni Network. For more information, please visit: http://eng.alexu.edu.eg/~elmongui/

Mohamed F. Mokbel (PhD: Purdue University, 2005; BS, MS: Alexandria University, 1996, 1999) is an associate professor in the Department of Computer Science and Engineering, University of Minnesota. His current main research interests include database systems, location-based services, and GIS. He is the main architect for the PLACE, Casper, and CareDB systems that provide a database support for location-based services, location privacy, and personalization, respectively. He has over 90 publications in major database and GIS conferences including VLDB, ACM SIGMOD, and IEEE ICDE and journals including ACM TODS, IEEE TKDE, and VLDB Journal. His research work has gained more than 2,200 citations (per Google Scholar) and has been recognized by three Best Paper Awards at IEEE MASS 2008, MDM 2009, and SSTD 2011. He is a recipient of the NSF CAREER award in 2010; the National Science Foundation's most prestigious award in support of junior faculty in USA. He has served as the general co-chair for SSTD 2011, program co-chair for ACM SIGSPATIAL GIS 2008-2010, MDM 2011, DMSN 2011, and LBSN 2011, and proceeding chair for ACM SIGMOD 2010. He is in the editorial board of IEEE Data Engineering Bulletin, Distributed and Parallel Databases Journal, and Journal of Spatial Information Science. He is a member of ACM and IEEE and a founding member of ACM SIGSPATIAL. For more information, please visit: www.cs.umn.edu/mokbel.

Walid G. Aref is a professor of computer science at Purdue. His research interests are in extending the functionality of database systems in support of emerging applications, e.g., spatial, spatio-temporal, multimedia, biological, and sensor databases. He is also interested in query processing, indexing, data mining, and geographic information systems (GIS). Professor Aref's research has been supported by the National Science Foundation, the National Institute of Health, Purdue Research Foundation, CERIAS, Panasonic, and Microsoft Corp. In 2001, he received the CAREER Award from the National Science Foundation and in 2004, he received a Purdue University Faculty Scholar award. Professor Aref is a member of Purdue's CERIAS and Discovery Park Cyber Center. He is a senior member of the IEEE, and a member of the ACM. Professor Aref is the current chair of the ACM Special Interest Group on Spatial Information (SIGSPATIAL). For more information, please visit: http://www.cs.purdue.edu/people/faculty/aref/.