Continuous Monitoring of Nearest Trajectories

Dimitris Sacharidis
Institute for the Management of Information Systems
R.C. ATHENA
dsachar@imis.athena-innovation.gr

Dimitrios Skoutas
Institute for the Management of Information Systems
R.C. ATHENA
dskoutas@imis.athena-innovation.gr

Georgios Skoumas
Knowledge and Database Systems Laboratory
National Technical University of Athens, Greece
gskoumas@dblab.ntua.gr

ABSTRACT

Analyzing tracking data of various types of moving objects is an interesting research problem with numerous real-world applications. Several works have focused on continuously monitoring the nearest neighbors of a moving object, while others have proposed similarity measures for finding similar trajectories in databases containing historical tracking data. In this work, we introduce the problem of continuously monitoring nearest trajectories. In contrast to other similar approaches, we are interested in monitoring moving objects taking into account at each timestamp not only their current positions but their recent trajectory in a defined time window. We first describe a generic baseline algorithm for this problem, which applies for any aggregate function used to compute trajectory distances between objects, and without any restrictions on the movement of the objects. Using this as a framework, we continue to derive an optimized algorithm for the cases where the distance between two moving objects in a time window is determined by their maximum or minimum distance in all contained timestamps. Furthermore, we propose additional optimizations for the case that an upper bound on the velocities of the objects exists. Finally, we evaluate the efficiency of our proposed algorithms by conducting experiments on three real-world datasets.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms

Keywords

moving objects, nearest neighbors, continuous monitoring

SIGSPATIAL'14, November 04 - 07 2014, Dallas/Fort Worth, TX, USA

ACM 978-1-4503-3131-9/14/11 ...$15.00
http://dx.doi.org/10.1145/2666310.2666408

1. INTRODUCTION

The increasingly widespread use of GPS enabled devices and other positioning technologies has made possible the tracking and monitoring of various types of moving objects, such as cars, people or animals. This has consequently lead to the study of a broad range of queries in a multitude of settings and applications. Retrieving objects whose motion is "similar" to that of a target query object is one of the most basic and useful analytical queries. In the literature, two important types of such moving object similarity queries have been proposed, the *Nearest Neighbor* (NN) and the *Nearest Trajectory* (NT), also known as trajectory similarity, queries. Both types define a similarity (or equivalently a distance) metric between moving objects, and return the top-k most similar (or equivalently least distant) objects with respect to a specified moving query object. The distinguishing characteristic is the definition of the metric. Generally speaking, NN queries, e.g., [5, 8], are concerned with the distance between *individual locations* of moving objects, i.e., at some particular time instance, whereas NT queries, e.g., [19, 21] take into account the distance between the *trajectories* of moving objects, i.e., for the sequence of locations over a time interval.

Both query types have been extensively studied for *historical* data, which can be stored on disk and indexed by specialized data structures. To the best of our knowledge, however, only NN queries have been considered in a *continuous monitoring* setting, where new object locations continuously arrive, and the result must be accordingly updated. This work introduces and studies *Continuous Nearest Trajectory* (CNT) queries. Given a *trajectory distance*, i.e., a metric aggregating individual location distances within a specified time window, a CNT query continuously returns the set of k objects that have the smallest trajectory distance to a given query object. CNT queries are a natural extension of both continuous NN queries, in the sense that the recent trajectory (and not only the last location) of objects is considered, as well as of historical NT queries, in that the result is computed and maintained in real-time.

We note that existing approaches do not extend for CNT queries. This is obvious for methods designed for historical data, as they take advantage of specialized index structures (which are not suitable for highly dynamic streaming data), and have the entire trajectory completely known upfront. Moreover, algorithms for continuous NN queries cannot be adapted for CNT. The main reason being that these methods assume that either the objects [34] or the query [31, 29, 17] is stationary, and define *validity* or *influence spatial regions*, which guarantee that the result will not change as long as the query object remains inside the region, in the former case, or that objects do not cross the region, in the latter case. Note that the latter methods can handle moving queries but only by treating them as new queries. This means that previous computations are no longer useful and the result needs to be computed from scratch, a scenario which is only tolerable when the query object changes location infrequently. Therefore, a validity/influence region-based approach is not possible for CNT queries, where both the query object and the other objects move continuously and freely. However, we show that, in a specific setting (concerning the definition of the trajectory distance and assuming maximum velocities), it is possible to determine the minimum expected time when a moving object can influence the result.

There are some other works dealing with different types of continuous queries on moving objects, which however do not extend for CNT queries either. In a setting similar to ours, [2] defines a trajectory distance metric, and continuously computes the spatiotemporal trajectory join, i.e., determines pairs of objects whose trajectory distance does not exceed a given threshold. In other words, the underlying computation is answering a *range query* under a *hard threshold*, which is always easier to process as the search space is restricted. In contrast, the k-th trajectory distance in CNT queries is not known beforehand and can be arbitrarily high, making the methods of [2] inapplicable for CNT queries. Another work [27] proposes an online method to determine groups of objects that move close together, i.e., within a disk of a given radius. In their problem, only the distance between individual locations is taken into account and the threshold is also hard, making their ideas not suitable for CNT queries.

Given these observations, we propose a generic baseline algorithm, termed BSL, for processing CNT queries. This approach makes no assumptions regarding the underlying trajectory distance function or the movement of the objects. Thus, it serves as a framework for adapting and optimizing algorithms to more specific cases.

Building upon this, we derive an optimized algorithm, called XTR, for the cases where the trajectory distance between two moving objects is defined based on the extrema (maximum or minimum) of individual location distances. The maximum-defined CNT query establishes a distance guarantee that spans the time window within which trajectories are examined, and can be used, for example, to determine how far the nearest objects have strayed. The minimum-defined CNT query determines objects that have come close to the query at any time during the recent past, and can be thought of as a continuous NN query with "memory". On the other hand, using both the minimum and the maximum location distances, gives a more informative description of the movement of an object, as it determines the tightest annulus (donut) around the query that contains the object's trajectory.

Moreover, we study the aforementioned case when a global maximum velocity for the objects is known. This is a reasonable assumption given that most moving objects have upper bounds on their attainable velocities. For this particular setting, we introduce the HRZ algorithm, which computes distance bounds in order to determine the earliest possible time, termed *horizon*, when an object may influence the result.

Our main contributions can be summarized as follows:

- We introduce and formally define the problem of continuously monitoring the objects with the k-nearest trajectories to a given query object, where trajectory distances take into consideration the objects' recent locations.
- We present a generic baseline algorithm (BSL) for the problem, which defines the types of events and operations needed for the computation.
- We propose the XTR algorithm optimized for the case when the trajectory distance is determined by the maximum or minimum individual location distance between objects. We also discuss some other related trajectory distance definitions.
- We present the HRZ algorithm that introduces further optimizations assuming that the moving objects have bounded velocities.
- We experimentally evaluate the proposed algorithms using real-world datasets.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 formally defines the problem. Then, Sections 4–6 present our algorithms. Finally, Section 7 presents our experimental evaluation, and Section 8 concludes the paper.

2. RELATED WORK

We discuss various types of queries for moving objects, distinguishing between NN variants in Section 2.1 and NT methods in Section 2.2.

2.1 NN Queries on Moving Objects

Given a (stationary) query object location and a set of (stationary) object locations, the k-Nearest Neighbor (NN) query retrieves the k objects which are closer to the query location. There are many different ways to extend the NN query for moving objects during some time interval, evident by the rich bibliography on the subject.

A first classification is based on where the interval of interest lies with respect to the current time. If it is in the past, the queries are termed *historical*, as they concern stored trajectory data. If the interval is placed in the future, the queries are further classified into *predictive*, when it can be assumed that objects move in a known manner (i.e., with constant velocity, or along a line) and thus their future locations can be extrapolated, or *monitoring*, when no assumptions are made on the moving patterns and thus location updates are issued. Processing historical and predictive NN queries is generally less challenging compared to monitoring queries, because the former essentially have at query time the entire trajectories of the moving objects.

The second classification is based on the semantic of the NN query during an interval. A *snapshot* NN query reports the objects that are closest to the query object at *any* time instance within the interval; e.g., find the object that comes closest to some location within the next 10 minutes. A *continuous* NN query reports the objects that are closest to the query object at *every* time instance within the interval; e.g., report the objects that were at some time closest to the query object during the past 10 minutes. Note that in the data stream literature, the term continuous (or long standing) query [1] refers to the case when the result of a query must be continuously updated as streaming tuples arrive; in the context of NN queries, these requirements essentially correspond to the continuous monitoring NN query.

Regarding predictive queries, [13] presents a dual plane method for predictive snapshot NN queries, in the case that all objects move in 1-D space, or are restricted to move within the same segment (i.e., road). [23] studies continuous predictive variants for various spatial queries, including NN, and describe a method to return the initial result and its validity period (i.e., the time at which the result will change). [24] studies continuous predictive NN queries assuming that only the query is moving along a line, while all other objects are stationary. [10] and [3] also deal with continuous predictive NN queries, but they are able to handle updates on the motion patterns of objects, without computing the result from scratch.

For continuous monitoring NN queries, [22] and [34] handle the case when only the query object is moving. The former retrieves $m > k$ nearest neighbors hoping that the result at a future time is among these m objects, provided that the query does not move much. The latter returns a Voronoi-based validity region such that the result does not change as long as the query remains within the region. [31], [29] and [17] present incremental grid-based methods for general continuous monitoring NN queries, i.e., when all objects move in a non-predictive manner; the last two works feature shared execution techniques to handle multiple NN queries.

In the case of historical trajectory data, R-tree based trajectory indices (e.g., 3D R-tree [26], TB-tree [20]) are typically used to expedite the NN query processing. [6] handles historical snapshot NN queries, while [5], [8] process historical continuous NN queries.

Another line of work concerns NN queries over uncertain data. For example, [25] processes continuous monitoring NN queries for objects with uncertain locations. [9] handles continuous predictive NN queries with updates for objects with uncertain locations and speeds. [18] deal with historical snapshot and continuous NN queries for objects with uncertain locations.

Finally, there has been some interest on identifying groups of moving objects, such as moving object clusters [12], flocks [7, 27], convoys [11], and swarms [15]. Generally speaking, these groups consist of objects that are close to each other (e.g., within a disk of a given radius) at each time instant. These methods however cannot be used for processing CNT queries.

2.2 Nearest Trajectories

There exist many approaches for defining distance (or similarity) metrics for trajectories. All of them also propose methods to identify the most similar trajectory to a given query trajectory, which can be extended to retrieve the top-k similar ones, but their techniques only operate on historical data. A useful survey on the topic is included in [19].

While the Euclidean distance (or some other L_p norm) is typically used to quantify closeness of two locations, the extension for the case of multiple locations within trajectories is not straightforward. In addition, a trajectory distance must take into account the temporal aspect of the locations. [30] defines the trajectory distance as the L_2 norm of individual Euclidean location distances, after re-sampling the trajectories to account for different reporting intervals. [16] ignores the temporal dimension and defines spatial trajectory distance as the average of the Euclidean distances computed between a location in one trajectory and its closest location in the other (termed the one way distance).

The previous trajectory distances can be computed in linear time with respect to the trajectory length. On the other hand, there exist more complex metrics, inspired from sequence similarity measures, that require quadratic time. [28] uses the Longest Common Subsequence (LCSS) similarity measure, an edit distance variant, that allows the matching of locations that are close in space at different time instants, provided that they are not far in time, and also allows for locations to be unmatched, e.g., accounting thus for location imprecisions or small deviations. In a similar manner, [4] defines the Edit Distance on Real Sequence (EDR) that captures the minimum number of edit operations (insert, delete, replace locations) necessary to transform one trajectory into the other.

An approach for finding historical top-k similar trajectories is presented in [21]. The basic algorithm prioritizes object examination aiming to avoid distance computations for objects not in the result. In addition, approximate techniques are also presented.

Another related problem is trajectory clustering, where the goal is to group trajectories based on a trajectory distance metric. For this problem, however, the basic underlying operation is typically a range query (retrieve trajectories within a given distance threshold) rather than a top-k similarity query. For example, in [14] the goal is to partition historical trajectories into sub-trajectories and then group them to construct dense clusters according to a metric that composes a perpendicular, a parallel, and an angle distance.

To the best of our knowledge, continuous monitoring of top-k similar trajectories has not been addressed in the past. The only work that handles continuous monitoring of a trajectory defined query is [2], which deals with spatiotemporal trajectory joins. The underlying trajectory distance metric is the maximum among all Euclidean location distances, and the goal is to find pairs of trajectories that are within a given trajectory distance threshold. That is the core query is a range rather than a top-k similarity query. Therefore, their approach is not applicable to our problem.

3. PROBLEM DEFINITION

Consider a set O of moving objects, whose locations are continuously monitored and reported at fixed discrete times, called *timestamps*. Location updates have the form $\langle o, t, x, y\rangle$, meaning that object o at timestamp t is at location $o[t] = (x, y)$. We assume that updates always arrive in increasing order of their timestamps, but we do not assume that for each timestamp updates are received for all objects.

We denote as $\mathcal{T}(o)$ the set of timestamps at which updates for o were received. For simplicity and without loss of generality we assume that for any timestamp t' for which no update for o was received, i.e. $t' \notin \mathcal{T}(o)$, the location of o is the same as its last reported location, i.e. $o[t'] = o[t]$, where $t \in \mathcal{T}(o)$ is the latest timestamp before t'. Essentially, this corresponds to assuming that object o has not moved during time $[t', t]$; making other assumptions can also be handled accordingly, e.g., by issuing artificial updates for the objects based on inferred locations.

Definition 1. The *location distance* between two objects o and o' at timestamp t is given by the Euclidean metric, i.e.,

$$d(o, o', t) = \sqrt{(x - x')^2 + (y - y')^2}.$$

where $o[t] = (x, y)$ and $o'[t] = (x', y')$ are the respective (reported or extrapolated) locations of o and o' at t.

The above definition measures the distance between two objects at a single timestamp. However, we are interested in comparing the recent trajectories of the objects, hence their distances over a series of consecutive timestamps within a specified time window. For this purpose, we introduce the following definition.

Definition 2. Given two objects o and o', a time window w, and an aggregate function $\mathcal{G}$, the *trajectory distance* of o and o' is defined by applying $\mathcal{G}$ on the location distances of o and o' at each timestamp within the time window of length w ending at timestamp t:

$$D(o, o', t, w, \mathcal{G}) = \mathcal{G}_{\tau \in [t-w,t]} d(o, o', \tau).$$

Function $\mathcal{G}$ can be any aggregate function, e.g., minimum, maximum, average, among others.

We now formally define the problem of continuously reporting the objects with the k-nearest trajectories to a moving query object.

Problem Statement. The *Continuous Nearest Trajectory* (CNT) query $\langle O, q, \mathcal{T}, k, w, \mathcal{G}\rangle$, where O is a set of moving objects, $q \in O$ a query object, and $\mathcal{T}$ a series of consecutive future timestamps, returns for each timestamp $t \in \mathcal{T}$ the k objects in O that have the smallest trajectory distance to q w.r.t. the time window w and the aggregate function $\mathcal{G}$, i.e., $\forall t \in \mathcal{T}$ it returns a subset $O_k \subseteq O$ of size k, such that $\forall o \in O_k, o' \in O \setminus O_k$:

$$D(q, o, t, w, \mathcal{G}) \leq D(q, o', t, w, \mathcal{G}).$$

4. BASELINE FOR CNT

We first describe a generic *baseline* (BSL) method for answering continuous nearest trajectory queries. BSL operates under *any* ag-

Algorithm 1: BSL

1 **foreach** $t \in \mathcal{T}$ **do**
2 $\quad O_A \leftarrow \emptyset$ // the set of affected objects at t
3 $\quad$ **if** *QUpd* **then**
4 $\quad\quad q.loc \leftarrow (x_q, y_q)$ // update q's current location
5 $\quad\quad O_A \leftarrow O$ // mark all objects for processing
6 $\quad$ **else**
7 $\quad\quad$ **foreach** *OUpd* and *OExp* **do**
8 $\quad\quad\quad O_A \leftarrow O_A \cup o$ // add the referred object in O_A
9 $\quad$ **foreach** $o \in O_A$ **do**
10 $\quad\quad D_o \leftarrow$ BSL_ProcessObject(o)
11 $\quad\quad$ **if** D_o has changed **then**
12 $\quad\quad\quad$ **if** o in the results R **then**
13 $\quad\quad\quad\quad$ update o's entry in R
14 $\quad\quad\quad$ **else if** D_o smaller than the trajectory distance of R's last entry **then**
15 $\quad\quad\quad\quad$ delete R's last entry
16 $\quad\quad\quad\quad$ insert an entry for o in R
17 $\quad$ **report** t, R

gregate function $\mathcal{G}$ and follows an event driven process, where the events to be handled are specified below:

- *Query location updates* (*QUpd*). This is an update $\langle q, t, x, y\rangle$ to the location of the query object q, specifying its new location (x, y) for the current timestamp t. This may result in changes in the trajectory distances of the objects, and subsequently changes in the current set of nearest trajectories (NTs). When objects are allowed to move arbitrarily, their new distances to the new query location have to be computed, and the new aggregate distances and NTs have to be evaluated.
- *Object location updates* (*OUpd*). This is an update $\langle o, t, x, y\rangle$ to the location of an object o, specifying its new location (x, y) for the current timestamp t. As a result, the current location distance of the object to the query has to be evaluated, which may affect its trajectory distance within the window w. If this changes, it may in turn affect the inclusion or not of the object in the result set. In addition, the system needs to remember to purge this location distance when it becomes obsolete, i.e., concerns a location outside the window. Therefore, it generates a corresponding expiration event that will be triggered at timestamp $t + w$ as described next.
- *Object distance expiration* (*OExp*). Unlike *QUpd* and *OUpd*, which are events received by the external environment, *OExp* events are generated and triggered by the system as part of handling *OUpd* events. *OExp* events have the form $\langle o, t\rangle$, and mean that a location distance for object o is set to expire at timestamp t (this location distance was computed for a location update received at timestamp $t - w$). Similarly to a location update, such a removal may affect the trajectory distance of the object, and consequently its inclusion in the set of NTs.

In the following, we describe in detail how BSL handles the above events to evaluate a CNT query.

BSL makes use of the following in-memory data structures. For the query, it only stores its latest location $q.loc$. For each object, it stores its latest location $o.loc$, as well as a list $o.hist$ of location distances and their corresponding timestamps, ordered by time. In addition, it uses an event queue Q to store and process *OExp* location distance expiration events, i.e., for removing distances for timestamps outside the time window w. An *OExp* event $\langle o, t\rangle$ means that at time t, BSL needs to purge an expired distance for object o. This is the least recent location distance in the list $o.hist$. Events in Q are inserted and processed in a FIFO manner, i.e. they are ordered by time. Finally, BSL maintains a results list R of size k, where each entry corresponds to an object and its trajectory distance, updated at every timestamp. Any object that does not appear in the list at time t has trajectory distance not less than the largest trajectory distance in R.

Algorithm 2: BSL_ProcessObject

1 **if** *OExp* event for o was triggered **then**
2 $\quad$ remove the expired location distance from $o.hist$
3 **if** *QUpd* or *OUpd* event for o was received **then**
4 $\quad$ update $o.loc$, if changed
5 $\quad$ compute new location distance d
6 $\quad$ add d to $o.hist$
7 $\quad$ create *OExp* event for o at timestamp $t + w$
8 update trajectory distance D_o
9 **return** D_o

Algorithm 1 shows the pseudocode for BSL. Since this is a continuous query, BSL executes in a loop for every timestamp $t \in \mathcal{T}$ (line 1), i.e. as long as the query is standing, and at each iteration it reports the current result set R (line 17). The input at each timestamp is the set of *QUpd*, *OUpd*, and *OExp* events that have been received for processing. Based on these events, BSL determines the set of *affected objects* O_A that require processing at this timestamp (line 2). Note that the set O_A contains not only objects that have received location updates, but also objects for which an expiration event was triggered, or, in the case of a query update, all objects.

If the query object has moved to a new location, then $q.loc$ is updated and new object distances need to be computed (lines 3–5). Otherwise, only those objects for which an update or expiration happened are marked for processing (lines 6–8). Subsequently, each affected object $o \in O_A$ is processed (lines 9–17). First, the procedure `BSL_ProcessObject` is invoked (line 10), which updates $o.loc$ and $o.hist$ accordingly, and recomputes the object's location distance and trajectory distance (see Algorithm 2 below). Then, BSL checks if the returned trajectory distance has changed (line 11). If so, then the result set R may need updating. In particular, if o was in the result, then its entry in R must be updated (lines 12–13) with the new trajectory distance. Otherwise, if the new trajectory distance is smaller than any trajectory distance in R, this means that o should be (tentatively) inserted in R, evicting the last entry (lines 14–16).

We next describe the procedure `BSL_ProcessObject`, shown in Algorithm 2, in more detail. If an expiration event has occurred for o, then the expired location distance is removed from $o.hist$ (lines 1–2). If the object's location has changed, $o.loc$ is updated (line 4). The new location distance of o is computed and added to the history (lines 5–6). Moreover a corresponding expiration event is added in Q (line 7). Finally, the new trajectory distance for o is computed and returned (lines 8–9).

5. EXTREMA-DEFINED CNT

In the following, we assume that the aggregate function $\mathcal{G}$ defining the trajectory distance is $\max$ or $\min$ over location distances, or, more generally, any other function taking as input only the extrema ($\max$, $\min$) location distances. In these instances, the trajectory distance is determined by one (or two) location distances within the time window. Note, however, that these location distances may change over time, as new locations arrive and old ones expire. Nonetheless, we show that processing of extrema-defined CNT queries can be streamlined. We first start our discussion considering the case of the $\max$ function; the case of $\min$ can be handled in a similar manner, and is hence omitted. We then discuss the necessary changes to process CNT queries for any extrema-defined aggregate function.

Algorithm 3: XTR_ProcessObject

1 **if** *OExp* event for o was triggered **then**
2 remove the expired location distance from $o.hist$
3 **if** *QUpd* or *OUpd* event for o was received **then**
4 update $o.loc$, if changed
5 compute new location distance d
6 add d to $o.hist$
7 remove from $o.hist$ all location distances less than d
8 $t' \leftarrow$ the earliest timestamp in $o.hist$
9 **if** Q contains *OExp* event for o **then**
10 update *OExp*'s time to $t' + w$
11 **else**
12 insert in Q the event $\langle o, t' + w\rangle$
13 **return** $D_o \leftarrow$ earliest location distance in $o.hist$

When the aggregate function $\mathcal{G}$ is $\max$, the trajectory distance of an object o is determined by the largest location distance within the time window w. In that case, we show that it is possible to discard some location distances which cannot influence the trajectory distance during their lifespan. Based on this observation, we describe the *Extrema* (XTR) algorithm, which is based on the BSL framework but reduces the number of location distances stored per object, and, consequently, the number of events generated and processed. The key observation of XTR is captured by the following lemma.

Lemma 1. Given an object o, where $d < d'$ are two location distances at timestamps $t < t'$ for $t' - t \leq w$, the location distance d does not contribute to the trajectory distance of o for any timestamp after t'.

Proof. Location distance d is valid, i.e., may contribute to the trajectory distance, during its lifespan ending at timestamp $t+w$. During the time interval $[t', t+w]$, location distance d' is also valid and greater, and thus dominates d. As a result, the trajectory distance, i.e., the maximum location distance, must be at least $d' > d$. □

The XTR algorithm uses the same data structures and variables as BSL and performs the same main operations described in Algorithm 1. However, XTR differs from BSL in the way it processes objects. In particular, we discern the following main differences. First, XTR only keeps the non-dominated location distances in $o.hist$, as Lemma 1 suggests. Second, at any time t, the event queue Q contains only a single entry per object o, and its semantics can be viewed differently: it now schedules trajectory distance recomputations rather than location expirations. By purging a priori those earlier location distances that are smaller than d, XTR avoids unnecessary triggering of the corresponding *OExp* events, thus avoiding unnecessary processing of objects whose trajectory distance cannot yet change.

The processing for an object o in XTR is handled by the procedure `XTR_ProcessObject` outlined in Algorithm 3. Its first tasks, removing expired location distances, updating the object's location, and recomputing the object's location distance to the query, are identical to BSL's (lines 1–6). In addition, based on Lemma 1, XTR removes any location distances less than d from $o.hist$ (line 7). Let t' be the earliest timestamp that remains (line 8). XTR inserts in Q an event to expire the location distance at t', if no event for o in Q already exists, otherwise it resets the scheduled time of the existing event (lines 9–12). This event essentially schedules the next trajectory distance recomputation necessary for o (assuming that the trajectory distance is not affected by newer location updates until then). Finally, the trajectory distance is set to the earliest location distance and returned (line 13).

We now discuss the general case where the aggregate function $\mathcal{G}$ is some function over the extrema ($\min$ and $\max$) location distances. One example of such a function is the average of the minimum and maximum location distances recorded for an object within the current time window. Recall that Lemma 1 identifies location distances which are irrelevant for the $\max$ case; an analogous lemma holds for the $\min$ case. Therefore, when both the $\max$ and the $\min$ location distance contribute to the trajectory distance, we can discard location distances which are irrelevant for both extrema cases. Following this observation, we propose the following changes to the `XTR_ProcessObject` algorithm. For each object o, we maintain its minimum and maximum location distances for the current time window, denoted as $o.min$ and $o.max$, respectively, i.e. $o.min = \min\{o.hist\}$ and $o.max = \max\{o.hist\}$. In addition, we keep a time marker t_m which is the earliest timestamp of either $o.min$ or $o.max$. In the event queue Q, we still need to keep only one entry for each object o, set to $\langle o, t_m + w\rangle$, to trigger a reevaluation of its trajectory distance when either $o.min$ or $o.max$ expires. Moreover, the early removal of unnecessary entries in $o.hist$ is now done as follows. When $o.min$ or $o.max$ changes, and t_m is set accordingly, we remove all entries from $o.hist$ with timestamp earlier than t_m. The reason for this is that for any location distance d with timestamp $t < t_m$ it holds that $o.min < d < o.max$ (otherwise, d would be the current min or max) and d cannot become a future $o.min$ or $o.max$ since it expires before them.

6. EXPLOITING BOUNDED VELOCITIES FOR EXTREMA-DEFINED CNT

This section considers extrema-defined trajectory distances and assumes that there exists a global upper bound v_{max} on the velocity of a moving object[1]. Under this realistic assumption, we show that it is possible to derive a more efficient algorithm than XTR for processing CNT queries. The proposed *Horizon* (HRZ) algorithm takes advantage of the velocity bound to further reduce the number of location updates that need to be processed. Similar to Section 5, we assume that the trajectory distance is the maximum location distance within the time window; the case of $\min$ is similar, while the more general case of extrema-defined functions can be handled in a straightforward manner.

The basic idea behind HRZ is the following. For ease of exposition, assume $k = 1$ and consider two objects o and o'. Let $\underline{D}[t]$ and $\overline{D}'[t]$ denote, respectively, a lower and an upper bound on the trajectory distances of o and o' to the query q at time t. Clearly, if $\underline{D}[t] > \overline{D}'[t]$ for any timestamp t within a time interval, then o cannot be in the result during that interval. Hence, in Section 6.1, we derive lower and upper bounds on trajectory distances. Then, in Section 6.2, we discuss the computation of the time horizon, which determines a time interval during which a particular object may not be a result. Finally, in Section 6.3, we put our ideas together and present the HRZ algorithm.

6.1 Bounds on Trajectory Distances

Let t be the current timestamp, and consider an object o for which the most recent location distance is d received at timestamp $t_d \leq t$, and its current trajectory distance is $D \geq d$, valid since timestamp $t_D \leq t_d$. A lower bound for the trajectory distance of o at any future timestamp $t' > t$ can be computed assuming that object o moves at maximum velocity v_{max} towards the query q, while q also moves at maximum velocity v_{max} towards o. As a result, since the last known update at t_d, the location distance of o to q is decreasing at a maximum rate of $2v_{max}$. Notice however that this will affect its trajectory distance only after both D and d have

[1]Note that the extension to differing maximum velocities across objects is straightforward and thus omitted.

expired. The trajectory distance in this setting is clearly a lower bound for the trajectory distance of o for any possible motion of o and q. Thus, we derive the following lemma.

Lemma 2. Given an object o at current timestamp t, with latest location distance d at timestamp $t_d \leq t$ and current trajectory distance $D \geq d$ valid since $t_D \leq t_d$, its trajectory distance for any future timestamp $t' \geq t$ is lower bounded by the function:

$$\underline{D_o}[t'] = \begin{cases} D & \text{if } t \leq t' \leq t_D + w \\ d & \text{if } t_D + w < t' \leq t_d + w \\ \max\{d - 2v_{max} \cdot (t' - t_d), 0\} & \text{if } t' > t_d + w. \end{cases}$$

Proof. First, observe that at time t the history of the object (i.e., during the time interval $[t - w, t]$) certainly contains a location distance with value D at time t_D (determining the current trajectory distance) and another with value d at time t_d. It may also contain other location distances, which however must have values between d and D. Consequently, it is easy to see that the lemma holds for the first two clauses.

Regarding the third clause, we need to show that for any future timestamp $t' > t$, the lower bound holds. Consider the location distances valid during the future time window $[t' - w, t']$; recall that location distance d is no longer valid. Let d_m be the largest valid location distance with timestamp $t_m \in [t' - w, t']$. Therefore, the trajectory distance at time t' is defined as d_m. Due to the bound on the velocity of objects, it holds that any object, o or the query q, from timestamp t_d (of o's known location update in the past) up to timestamp t_m cannot have traveled a distance greater than $v_{max} \cdot (t_m - t_d)$. As a result, the location distance of o cannot have decreased more than $2v_{max} \cdot (t_m - t_d)$ (but not become less than zero), which is the case that o and q travel towards one another (and travel together once they reach each other). Therefore, the location distance at t_m cannot be less than $d_m \geq d - 2v_{max} \cdot (t_m - t_d)$, and is also greater than zero. Since $t_m \leq t'$, the lower bound holds. $\square$

In a similar way, we can also derive an upper bound for the trajectory distance of o in a future timestamp t'. This can be computed assuming that object o moves at maximum velocity v_{max} away from the query q, while also q moves at maximum velocity v_{max} away from o. As a result, since the last known update at t_d, the location distance of o to q is increasing at a rate of $2v_{max}$. Again, any updates will come into effect only as long as there exists no previous value that is still valid and greater. The trajectory distance in this setting is clearly an upper bound for the trajectory distance of o for any possible motion of o and q. Thus, we derive the following lemma.

Lemma 3. Given an object o at timestamp t, with latest location distance d at timestamp $t_d \leq t$ and current trajectory distance $D \geq d$ valid since $t_D \leq t_d$, its trajectory distance for any future timestamp $t' \geq t$ is upper bounded by the function:

$$\overline{D_o}[t'] = \begin{cases} \max\{D, d + 2v_{max} \cdot (t' - t_d)\} & \text{if } t \leq t' \leq t_D + w \\ d + 2v_{max} \cdot (t' - t_d) & \text{if } t' > t_D + w. \end{cases}$$

Proof. Consider the first clause, and a timestamp $t' \in [t, t_D + w]$; the corresponding time window is $[t' - w, t']$ and D is still valid. Let d_m denote the largest valid location distance with timestamp $t_m \in [t' - w, t']$. Trivially, if d_m is D, the upper bound holds. Assume otherwise, i.e., $d_m > D$. Using similar reasoning as in Lemma 2, the location distance of o from timestamp t_d up to timestamp t_m cannot have increased more than $2v_{max} \cdot (t_m - t_d)$. Therefore, $d_m \leq d + 2v_{max} \cdot (t_m - t_d)$, and the upper bound also holds for this case because $t_m \leq t'$. The second clause is proved in a similar way, given that D has now expired. $\square$

6.2 Time Horizon of Objects

We now proceed to derive the minimum time required for an object $o \notin R$ to enter the result set. We refer to this as the *time horizon* of an object, corresponding to the earliest time for which the object's trajectory distance may become equal to (or less than) the trajectory distance of some object in R. Using Lemmas 2 and 3, the time horizon is formally defined as follows.

Definition 3. Given the current result set R at timestamp t, the *time horizon* t_h for an object $o \notin R$ is defined as the earliest possible time that the trajectory distance of o becomes lower than that of any object in R, i.e.:

$$t_h = \min\{t' \geq t \mid \exists o' \in R : \underline{D_o}[t'] \leq \overline{D_{o'}}[t']\}$$

An important remark regarding the previous definition is that it does not suffice to just consider the trajectory distance upper bound of the k-th object in R. As location updates may not occur at all timestamps, it is possible for two objects $o_i, o_j \in R$ with trajectory distances $D_i < D_j$ to have at some future timestamp t' upper trajectory bounds such that $\overline{D_i}[t'] > \overline{D_j}[t']$. This can occur, for example, when the objects' last location distances and timestamps satisfy the conditions $d_i > d_j$ and $t_i < t_j$.

As a result, computing the time horizon for an object requires considering the trajectory distance upper bounds for all objects in R, which is time consuming given that the time horizon needs to be computed at each timestamp for each affected object not in the result set. We thus propose an alternative method for determining the time horizon. The key idea is the following lemma, which derives a single upper bound on the trajectory distance of any object in the result set R.

Lemma 4. Consider a set of objects R at current timestamp t, where, for the i-th object, d^i denotes its latest location distance at timestamp t_d^i and $D^i \geq d^i$ denotes its current trajectory distance valid since timestamp $t_D^i \leq t_d^i$. Define object $o^+ \in R$ to be the one with the largest trajectory distance, and object $o^* \in R$ to be the one that can have the largest possible location distance at current timestamp t, i.e.,

$$o^+ = \operatorname*{argmax}_{o_i \in R} D^i \text{ and } o^* = \operatorname*{argmax}_{o_i \in R} \left(d^i + 2v_{max} \cdot (t - t_d^i)\right).$$

Then, the trajectory distance of any object in R for any future timestamp $t' \geq t$ is upper bounded by the function:

$$\overline{D_R}[t'] = \begin{cases} \max\{D^+, d^* + 2v_{max} \cdot (t' - t^*)\} & \text{if } t \leq t' \leq t + w \\ d^* + 2v_{max} \cdot (t' - t^*) & \text{if } t' > t + w, \end{cases}$$

where D^+ is the trajectory distance of o^+, and d^* is the latest location distance of o^* computed at timestamp t^*.

Proof. It suffices to show that the upper bound on the trajectory distance of each object in R according to Lemma 3 is always (i.e., for any $t' > t$) not greater than the upper bound provided by this lemma. Consider an object $o_i \in R$ and its trajectory distance upper bound:

$$\overline{D^i}[t'] = \begin{cases} \max\{D^i, d^i + 2v_{max} \cdot (t' - t_d^i)\} & \text{if } t \leq t' \leq t_D^i + w \\ d^i + 2v_{max} \cdot (t' - t_d^i) & \text{if } t' > t_D^i + w. \end{cases}$$

First note that $t_D^i < t$, and consider a future timestamp t' during the time interval $[t, t_D^i + w]$. Comparing the first clause of the two bounds, we can see that $D^+ \geq D^i$ from the definition of object o^+. On the other hand, from the definition of o^* we derive that $d^* + 2v_{max} \cdot (t - t^*) \geq d^i + 2v_{max} \cdot (t - t_d^i)$. Adding $2v_{max} \cdot (t' - t)$ to both sides of the inequality, we derive that the lemma holds.

Algorithm 4: HRZ

1 **foreach** $t \in \mathcal{T}$ **do**
2     $O_A \leftarrow \emptyset$ // the set of objects marked for processing at t
3   **if** *QUpd* **then**
4         $q.loc \leftarrow (x_q, y_q)$ // update q's current location
5         $O_A \leftarrow O$ // mark all objects for processing
6   **else**
7   **foreach** *OUpd* and *OExp* **do**
8             $O_A \leftarrow O_A \cup o$ // add the referred object in O_A
9   **foreach** $o \in O_A \cap R$ **do**
10         $D_o \leftarrow$ HRZ_ProcessObject(o)
11   update o's entry in R
12   identify objects o^+ and o^* in R // from Lemma 4
13   **foreach** $o \in O_A \setminus R$ **do**
14   **if** $t < o.t_h - w$ **then continue**
15         $D_o \leftarrow$ HRZ_ProcessObject(o)
16   **if** D_o smaller than the trajectory distance of R's last entry **then**
17   delete R's last entry
18   insert an entry for o in R
19   **report** t, R

Next, consider a future timestamp t' during the time interval $[t_D^i + w, t + w]$, and compare the second clause of $\overline{D^i}[t']$ to the first clause of $\overline{D_R}[t']$. With similar reasoning as before, we have that $d^* + 2v_{max} \cdot (t' - t^*) \geq d^i + 2v_{max} \cdot (t - t_d^i)$, and since the first clause of $\overline{D_R}[t']$ is always greater than the left-hand side of the inequality, the lemma holds.

In the case of a future timestamp $t' > t + w$, when the second clauses of the bounds apply, it is easy to see, using similar reasoning as before, that the lemma holds. ☐

Using the bound on the trajectory distance of any object in R, it is possible to efficiently compute a timestamp that never overestimates the time horizon, as the next lemma suggests.

Lemma 5. Given the current result set R at timestamp t, the time horizon t_h for an object $o \notin R$ is not less than the following value:

$$t_h \geq \min\{t' \geq t \mid \underline{D_o}[t'] \leq \overline{D_R}[t']\}.$$

Proof. Denote as A the set from Definition 3, i.e., $A = \{t' \geq t \mid \exists o' \in R : \underline{D_o}[t'] \leq \overline{D_{o'}}[t']\}$, and as B the set from this lemma, i.e., $B = \{t' \geq t \mid \underline{D_o}[t'] \leq \overline{D_R}[t']\}$. We claim that $B \subseteq A$ to prove the lemma. Since it holds that $\overline{D_R}[t'] \geq \overline{D_{o'}}[t']$ for any $o' \in R$ from Lemma 4, the condition of set B is harder to satisfy than that of A, and thus the claim $B \subseteq A$ holds. ☐

Lemma 5 suggests that we can compute, in constant time, a timestamp not greater than the time horizon as the solution of the equation $\underline{D_o}[t'] = \overline{D_R}[t']$. Henceforth, to simplify the presentation of HRZ, whenever we refer to the time horizon t_h or its computation, we mean the solution of this equation instead of Definition 3.

6.3 The HRZ Algorithm

Having a method to compute the time horizon of an object, we next detail the HRZ algorithm, highlighting its differences with respect to XTR. The data structures and variables that HRZ uses are as in XTR, with the exception that for each object HRZ additionally stores its time horizon t_h indicating the time after which the object may appear in the result set R. The computation of t_h is based on Lemma 5. The HRZ algorithm takes advantage of the time horizon to reduce the number of events processed as follows. At any timestamp before $t_h - w$, HRZ ignores updates for the particular object. During the time interval $[t_h - w, t_h]$, HRZ only stores the locations and location distances, since these are necessary to compute the trajectory distance at time t_h. However, it does not compute the trajectory distance, since it is guaranteed to be greater than those in R, and it does not add any events in Q. After the time horizon t_h, HRZ operates similar to XTR.

Algorithm 5: HRZ_ProcessObject

1 **if** *OExp* event for o was triggered **then**
2   remove the expired location distance from $o.hist$
3 **if** *QUpd* or *OUpd* event for o was received **then**
4   update $o.loc$, if changed
5   compute new location distance d
6   **if** $o \notin R$ **then**
7   compute t_h
8   **else** $t_h \leftarrow t$
9   **if** $t < t_h - w$ **then**
10   clear state of o
11   remove o's entry in Q
12   **return** $D_o \leftarrow \infty$
13   **else**
14   add d to $o.hist$
15   remove from $o.hist$ all location distances less than d and with timestamps before $t - w$
16   **if** $t < t_h$ **then**
17   **return** $D_o \leftarrow \infty$
18   **else**
19             $t' \leftarrow$ the earliest timestamp in $o.hist$
20   **if** Q contains *OExp* event for o **then**
21   update *OExp*'s time to $t' + w$
22   **else**
23   insert in Q the event $\langle o, t' + w\rangle$
24 **return** $D_o \leftarrow$ earliest location distance in $o.hist$

Algorithm 4 shows the pseudocode for HRZ. The main difference from BSL and XTR is that it handles the processing of affected objects in two phases. In the first phase (lines 9–11), HRZ considers only objects that are in R, i.e., objects that were reported as results in the previous timestamp. For these objects, the processing (handled by HRZ_ProcessObject) is essentially identical to XTR, as we later explain. Once processing is completed, the object's entry in R is updated if its trajectory distance changed.

Between the first and second phase, HRZ scans all objects in R, and determines objects o^+ and o^* as defined in Lemma 4 (line 12). Then, during the second phase (lines 13–18), HRZ considers the remaining affected objects, i.e., not in R. If the current time is more than w timestamps before the time horizon $o.t_h$ of an object o, HRZ essentially ignores o (line 14). For each other affected object, its processing (handled by HRZ_ProcessObject at line 15) differs significantly from XTR. Once it concludes, HRZ checks whether the object should be included in the result set R provided that its trajectory distance has sufficiently decreased (lines 16–18).

We next describe the HRZ_ProcessObject procedure, shown in Algorithm 5. As in XTR, the procedure removes expired location distances if an event from Q was triggered (lines 1–2). The main operations of the procedure occur when either an object or a query location update were received (lines 3–23). First, the object's location is updated, if it changed, and its location distance is computed (lines 4–5). If the object o under processing did not belong in the result at the previous timestamp (line 6), the procedure computes the time horizon t_h by applying Lemma 5 (line 7); otherwise t_h is set to current time (line 8), meaning that object o may belong in the result. Since the time horizon is now recalculated taking into account the object's current location distance, it is necessary to check again if the object should be ignored (line 9). If the check succeeds, all stored information for object o is cleared, its entry in the event queue is removed and an infinite trajectory distance is returned (lines 10–12).

In the following operations (lines 14–23), it holds that the current time is $t \geq t_h - w$, hence HRZ needs to store locations and location distances. The object's current location distance d is stored (line 14), and all location distances less than d are removed (line 15) as in XTR. Subsequently, if the current time falls in the interval $[t_h - w, t_h]$ (line 16), finding the actual trajectory distance during this interval is not necessary, as the object is guaranteed to not be in the result set. Therefore, HRZ simply returns an infinite trajectory distance (line 17) and, to increase efficiency, it does not create a corresponding expiration event. A consequence is that at future timestamps after the current time horizon, there may exist expired location distances. Therefore, the procedure may also have to remove such distances (line 15). Otherwise, if the current time is not before the time horizon (line 18), the processing is identical to XTR. That is, the earliest timestamp is identified, and the event queue is properly updated (lines 19–22). The last operation is to compute the trajectory distance from the earliest location distance and return it (line 24). As a final note, observe that if the object was not in the result at the previous timestamp, its processing is identical to XTR, as its time horizon is set to current time (line 8).

7. EXPERIMENTAL EVALUATION

To evaluate the efficiency of the proposed algorithms for the continuous nearest trajectories query, we conduct experiments using three real-world trajectory datasets. In the following, we first present the datasets used for the evaluation and then we report the results of our experiments.

7.1 Datasets

To cover a variety of cases regarding the shapes of trajectories, the type of the objects, and the speed and type of movement, we use three different real-world datasets in our experiments. We refer to these datasets as *Beijing taxis*, *Aegean ships*, and *Athens vehicles*. These datasets vary in their characteristics, ensuring that our methods are robust across diverse settings. For example, in the *Beijing taxis* dataset, the shape of the trajectories exhibits a relatively high regularity due to the grid-like structure of the underlying road network. At the other end, the Athens road network is highly irregular, resulting in diverse trajectories with constantly varying headings. Finally, the *Aegean ships* trajectory dataset comprises relatively long trajectories with medium degree of heading variations.

A typical issue in trajectory datasets is the often high variation of the sampling rate, caused, for example, by weak GPS signal, or when the user manually switches off their personal tracking devices (e.g., to save battery or for privacy). In our datasets, to reduce such gaps, when the time interval between two consecutive reported locations exceeds a specified threshold (set to 30 seconds) but is not greater than a maximum threshold (set to 120 seconds), we use linear interpolation to create intermediate location updates.

We next detail the used datasets.

- *Beijing taxis.* These trajectories are from the T-Drive trajectory dataset, which contains GPS tracking data from taxis moving in the area of Beijing [32, 33]. A total of 1,023,924 trajectories are used. These trajectories belong to a total of 569 taxis recorded in the period 2/2/2008 – 4/2/2008. Each trajectory comprises on average 3,017 points (i.e. location updates).
- *Aegean ships.* This dataset contains GPS tracking data from ships moving in the Aegean sea[2]. A total of 986,275 trajectories are used, obtained from 887 ships in the period 31/12/2008 – 02/01/2009. On average, each trajectory comprises 1,101 points.
- *Athens vehicles.* This dataset contains GPS tracking data from vehicles moving in the area of Athens, recorded in the context of the SimpleFleet project[3]. 667,421 trajectories are used, coming from 2,497 vehicles on 01/10/2012. Each trajectory comprises 157 points on average.

[2]http://www.chorochronos.org/?q=node/8

[3]http://www.simplefleet.eu/

7.2 Results

The goal of the experimental evaluation is to study the efficiency of the proposed algorithms, and in particular to compare the speedup achieved by the XTR and HRZ algorithms with respect to the more generic baseline BSL algorithm. For this purpose, we conduct a series of experiments, using the datasets previously described. The trajectory distance metric used in all experiments is the maximum of all valid location distances. We note that the performance of BSL is identical for all metrics, as the method is distance agnostic. On the other hand, XTR and HRZ have roughly the same performance for any extremum-defined trajectory distance metrics.

The main performance metric is the total execution time, i.e., the time spent processing a CNT query over its entire lifespan. To better investigate the performance gains of XTR and HRZ with respect to BSL, we also report their relative improvement in execution time, and the percentage of events (location updates and expirations) that they process compared to BSL. The investigated parameters affecting the performance of the algorithms is the number k of nearest trajectories requested, the size w of the time window, and the number $|O|$ of objects. In all settings, the reported performance metrics (time and number of events) are the average of 10 executions involving randomly selected query objects. The answer to a CNT query is calculated at each timestamp that an update or an expiration event occurs.

7.2.1 *Varying the number of nearest trajectories*

In this experiment, we measure the total execution time of each of the three algorithms with respect to the number k of nearest trajectories returned. The total monitoring time $\mathcal{T}$ is set to 60 minutes, and the size w of the time window for keeping each object's history is set to 5 minutes. The results are presented in Figure 1.

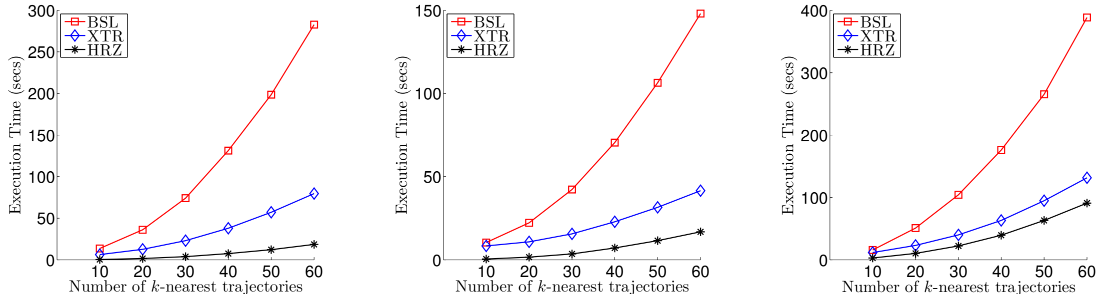
The first important observation is that for all datasets, the execution times of both XTR and HRZ are significantly lower than for BSL, clearly showing in practice the effectiveness of the corresponding optimizations for these cases. Furthermore, HRZ has also a clear benefit over XTR. The differences are more pronounced in the *Beijing taxis* dataset, which shows that, due to the regularity in the movement of objects imposed by the underlying grid-like structure of the Beijing road network, more effective pruning of location updates and distance recomputations can be achieved. In contrast, the differences become relatively smaller in *Athens vehicles*, where the road network is less uniform.

A second observation is that for all algorithms the execution time increases with k. This is expected since k regulates the size of the ordered list R that has to be maintained by the algorithm at each timestamp. However, this increase is lower for XTR and, even more so for HRZ, which is an additional evidence that XTR and HRZ need to process fewer events, and hence perform fewer lookup and sort operations on R.

7.2.2 *Varying the size of the time window*

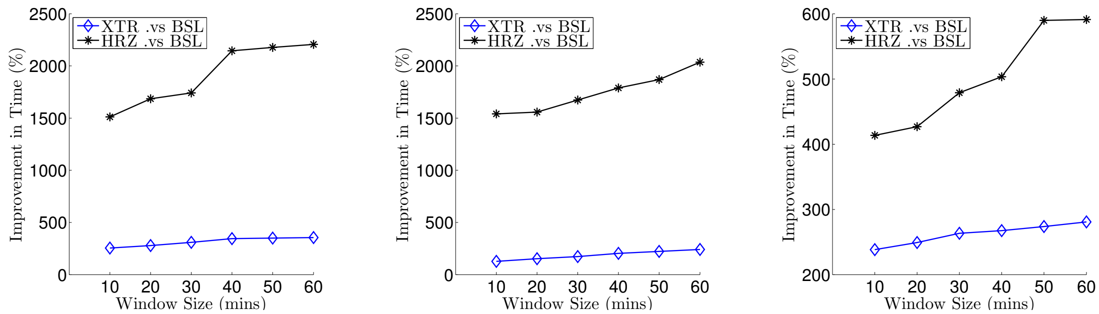
In the next experiment, we compare the execution time of the three algorithms with respect to the window size w during which the past location distances of an object remain valid and contribute to the trajectory distance. As previously, the total monitoring time $\mathcal{T}$ was set to 60 minutes, and k was set to 10. To better illustrate the improvement in execution time achieved by XTR and HRZ with respect to BSL, we plot the speedup of XTR and HRZ compared to BSL. The results are shown in Figure 2.

As illustrated, XTR shows a speedup of almost up to 5 times over



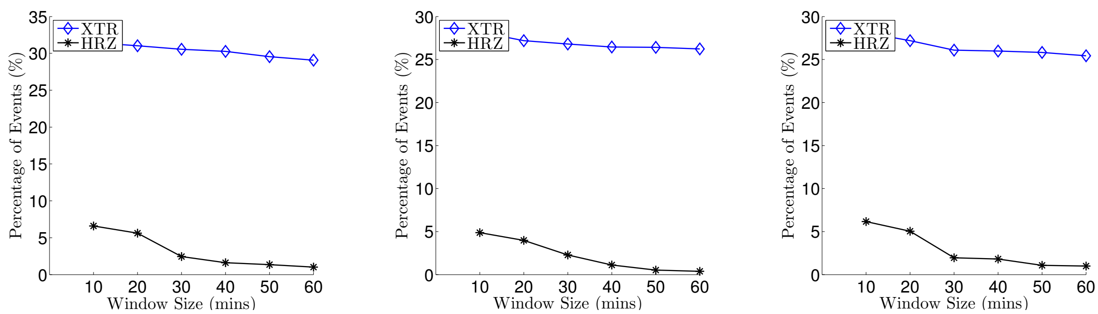
(a) Beijing taxis (b) Aegean ships (c) Athens vehicles

Figure 1: Execution time of BSL, XTR and HRZ w.r.t. the number k of nearest trajectories returned at each timestamp.



(a) Beijing taxis (b) Aegean ships (c) Athens vehicles

Figure 2: Execution time speedup of XTR and HRZ compared to BSL w.r.t. the size w of the time window.



(a) Beijing taxis (b) Aegean ships (c) Athens vehicles

Figure 3: Percentage of events handled by XTR and HRZ compared to BSL w.r.t. the size w of the time window.

BSL, while for HRZ it is even higher, in the range of 15× to 22× for the first two datasets and 4× to 6× for the *Athens vehicles*. Notice that in this setting $k = 10$, so when these results are considered in conjunction with those illustrated in Figure 1, these speedups are expected to be increasingly higher for higher values of k.

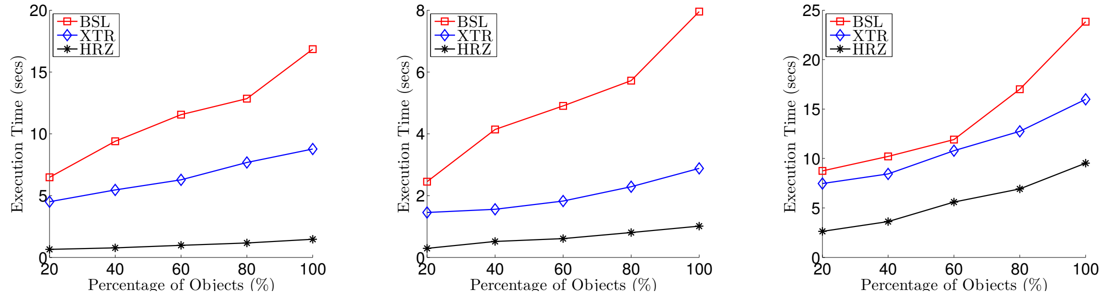
Moreover, the speedup for both algorithms increases as the window size w increases. This behavior is because XTR and HRZ only consider the maximum or minimum value in each object's history, so the gain is higher for larger time windows. The gain for HRZ is even higher as w increases, since HRZ is able to set time horizons for objects later in the future, thus ignoring more location updates and further decreasing the total events to be handled.

To better illustrate the reduction of the number of events that XTR and HRZ process, and how this is affected by the size of the time window, we also report the number of events in the event queue Q that are created and processed by XTR and HRZ with respect to those by BSL. The results are shown in Figure 3. Indeed, the results are in agreement with those in Figure 2, showing that XTR needs to process only about 30% of the events processed by BSL, while HRZ fewer than 5%.

7.2.3 Varying the number of objects

In the last set of experiments, we measure the performance of the algorithms with respect to the number of objects. For this purpose, we create subsets of the original datasets, containing a specific portion of randomly selected objects, and ran the algorithms on these subsets. The other parameters are set to $\mathcal{T} = 60$ minutes, $k = 10$, and $w = 5$ minutes. The results are plotted in Figure 4.

As expected, the execution time of all algorithms increases as the size of the dataset increases. However, XTR and, especially, HRZ show better scalability. Especially HRZ for the cases of the *Beijing taxis* and the *Aegean ships*, where the movement of the objects is relatively more regular, proves to be quite robust with respect to the total number of objects, which verifies that it can successfully avoid unnecessary examinations of objects that cannot qualify as candidates for the result set.



(a) Beijing taxis (b) Aegean ships (c) Athens vehicles

Figure 4: Execution time of BSL, XTR and HRZ w.r.t. the size $|O|$ of the dataset.

8. CONCLUSIONS

This paper introduced and studied the problem of continuously reporting moving objects with similar recent trajectories to a given query object. This problem extends the case of continuous nearest neighbor monitoring and of discovering similar trajectories in historical data. We proposed a generic baseline method that operates for any aggregate trajectory distance metric; the extension to other metrics is left as future work. Then we turned our attention to instances where the distance between the trajectories of two objects is determined by the extrema (minimum and maximum) of their individual location distances. For these instances, we described two more efficient algorithms, with the latter taking into account a given bound on the velocities of objects. Our experimental study on real-world datasets showed that our methods exhibit up to 22 times performance gain compared to the baseline.

Acknowledgements This work was supported by the EU FP7 Projects GEOCROWD (FP7-PEOPLE-2010-ITN-264994) and GEOSTREAM (FP7-SME-2012-315631), and by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

9. REFERENCES

[1] S. Babu and J. Widom. Continuous queries over data streams. *SIGMOD Record*, 30(3), 2001.
[2] P. Bakalov and V. J. Tsotras. Continuous spatiotemporal trajectory joins. In *GSN*, 2006.
[3] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis. Nearest and reverse nearest neighbor queries for moving objects. *VLDB J.*, 15(3), 2006.
[4] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, 2005.
[5] E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis. Algorithms for nearest neighbor search on moving object trajectories. *GeoInformatica*, 11(2), 2007.
[6] Y. Gao, C. Li, G. Chen, L. Chen, X. Jiang, and C. Chen. Efficient *k* -nearest-neighbor search algorithms for historical moving object trajectories. *J. Comput. Sci. Technol.*, 22(2), 2007.
[7] J. Gudmundsson and M. J. van Kreveld. Computing longest duration flocks in trajectory data. In *GIS*, 2006.
[8] R. H. Güting, T. Behr, and J. Xu. Efficient *k*-nearest neighbor search on moving object trajectories. *VLDB J.*, 19(5), 2010.
[9] Y.-K. Huang, S.-J. Liao, and C. Lee. Efficient continuous k-nearest neighbor query processing over moving objects with uncertain speed and direction. In *SSDBM*, 2008.
[10] G. S. Iwerks, H. Samet, and K. P. Smith. Continuous k-nearest neighbor queries for continuously moving points with updates. In *VLDB*, 2003.
[11] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. *PVLDB*, 1(1), 2008.
[12] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *SSTD*, 2005.
[13] G. Kollios, D. Gunopulos, and V. J. Tsotras. Nearest neighbor queries in a mobile environment. In *STDBM*, 1999.
[14] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *SIGMOD*, 2007.
[15] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining relaxed temporal moving object clusters. *PVLDB*, 3(1), 2010.
[16] B. Lin and J. Su. Shapes based trajectory queries for moving objects. In *GIS*, 2005.
[17] K. Mouratidis, M. Hadjieleftheriou, and D. Papadias. Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In *SIGMOD*, 2005.
[18] J. Niedermayer, A. Züfle, T. Emrich, M. Renz, N. Mamoulis, L. Chen, and H.-P. Kriegel. Probabilistic nearest neighbor queries on uncertain moving object trajectories. *PVLDB*, 7(3), 2013.
[19] N. Pelekis, I. Kopanakis, G. Marketos, I. Ntoutsi, G. L. Andrienko, and Y. Theodoridis. Similarity search in trajectory databases. In *TIME*, 2007.
[20] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches in query processing for moving object trajectories. In *VLDB*, 2000.
[21] G. Skoumas, D. Skoutas, and A. Vlachaki. Efficient identification and approximation of k-nearest moving neighbors. In *SIGSPATIAL/GIS*, 2013.
[22] Z. Song and N. Roussopoulos. K-nearest neighbor search for moving query point. In *SSTD*, 2001.
[23] Y. Tao and D. Papadias. Time-parameterized queries in spatio-temporal databases. In *SIGMOD*, 2002.
[24] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *VLDB*, 2002.
[25] G. Trajcevski, R. Tamassia, H. Ding, P. Scheuermann, and I. F. Cruz. Continuous probabilistic nearest-neighbor queries for uncertain trajectories. In *EDBT*, 2009.
[26] M. Vazirgiannis, Y. Theodoridis, and T. K. Sellis. Spatio-temporal composition and indexing for large multimedia applications. *Multimedia Syst.*, 6(4), 1998.
[27] M. R. Vieira, P. Bakalov, and V. J. Tsotras. On-line discovery of flock patterns in spatio-temporal data. In *GIS*, 2009.
[28] M. Vlachos, D. Gunopulos, and G. Kollios. Discovering similar multidimensional trajectories. In *ICDE*, 2002.
[29] X. Xiong, M. F. Mokbel, and W. G. Aref. Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *ICDE*, 2005.
[30] Y. Yanagisawa, J. Akahani, and T. Satoh. Shape-based similarity query for trajectory of mobile objects. In *MDM*, 2003.
[31] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *ICDE*, 2005.
[32] J. Yuan, Y. Zheng, X. Xie, and G. Sun. Driving with knowledge from the physical world. In *KDD*, 2011.
[33] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: driving directions based on taxi trajectories. In *GIS*, 2010.
[34] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based spatial queries. In *SIGMOD*, 2003.