Efficient Point-based Trajectory Search*

Shuyao Qi[1], Panagiotis Bouros[2], Dimitris Sacharidis[3], and Nikos Mamoulis[1]

[1] Department of Computer Science
The University of Hong Kong
{syqi2,nikos}@cs.hku.hk
[2] Department of Computer Science
Humboldt-Universität zu Berlin, Germany
bourospa@informatik.hu-berlin.de
[3] Faculty of Informatics
Technische Universität Wien, Austria
dimitris@ec.tuwien.ac.at

Abstract. Trajectory data capture the traveling history of moving objects such as people or vehicles. With the proliferation of GPS and tracking technology, huge volumes of trajectories are rapidly generated and collected. Under this, applications such as route recommendation and traveling behavior mining call for efficient trajectory retrieval. In this paper, we first focus on distance-based trajectory search; given a collection of trajectories and a set query points, the goal is to retrieve the top-k trajectories that pass as close as possible to all query points. We advance the state-of-the-art by combining existing approaches to a hybrid method and also proposing an alternative, more efficient range-based approach. Second, we propose and study the practical variant of bounded distance-based search, which takes into account the temporal characteristics of the searched trajectories. Through an extensive experimental analysis with real trajectory data, we show that our range-based approach outperforms previous methods by at least one order of magnitude.

1 Introduction

The proliferation of GPS and tracking technology has brought to availability huge volumes of trajectories from real moving objects such as mobile phone users, vehicles and animals. Searching such a collection of trajectories finds several applications, including route recommendation, behavior mining, and in transportation systems [1, 2]. Different from conventional retrieval tasks which identify similar trajectories to a given one or those crossing a specific spatial region, in this paper we focus on *point-based search*, which retrieves trajectories based on given points. In particular, taking as input a set of query points Q (e.g., a particular set of POIs), the *distance-based trajectory search* studied in [3, 4] retrieves the trajectories that pass as close as possible to all query points.

* Work supported by grant HKU 715413E from Hong Kong RGC, and by the European Social Fund and Greek National Funds through the NSRF Research Program Thales.

Specifically, the distance of a trajectory t to Q is computed by summing up, for each query point $q \in Q$, its distance to the nearest point in t.

Consider for instance a collection of touristic trajectories; a travel agency issues a distance-based query to survey or recommend popular routes that pass close to specific sightseeing attractions. As another example, query set Q could contain traffic congestion points; in this case, the traffic department seeks to discover the causes of the congestion by analyzing the trajectories that pass near the points in Q. In the context of surveillance and security applications, Q may contain locations of crime scenes, and hence the police department issues a distance-based query to investigate the correlation of these crime locations by identifying suspects who moved close to all of them.

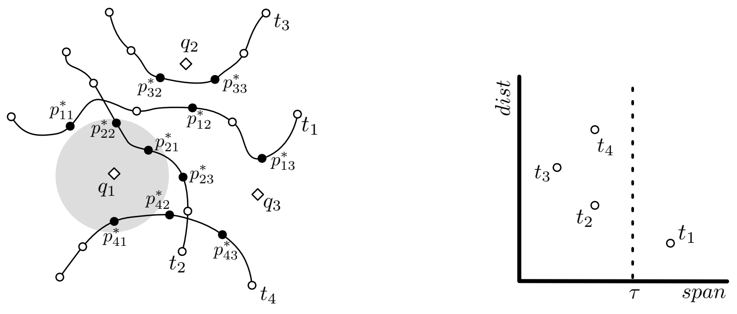
Contributions. This paper tackles two problems under the point-based trajectory search. First, we thoroughly study the efficient evaluation of *distance-based trajectory search*. We review in detail existing algorithms `IKNN` [3] and `GH/QE` [4]. These methods follow a *candidate generation* and *refinement* paradigm, and invoke a nearest neighbor (NN) search centered at each query point to examine the trajectories in ascending order of their distance to Q. By analyzing the pros and cons of these methods, we design a hybrid NN-based algorithm which consistently outperforms `IKNN` and `GH/QE` by over an order of magnitude. Going one step further, we tackle the inherent shortcomings of the NN-based approach itself, namely (a) the increased I/O cost due to independently running multiple NN searches and (b) the increased CPU cost for continuously maintaining a priority queue for each NN search. We propose a novel *spatial range-based* approach, which is up to 2 times faster than our hybrid algorithm.

Second, we observe that the distance-based search ranks trajectories solely on how close they pass to the query points in Q, ignoring however other qualitative characteristics of the retrieved results. To fill this gap, we introduce a practical variant of distance-based trajectory search, which also takes into account the temporal aspect of the trajectories. Specifically, this *bounded distance-based search* filters out non-interesting trajectories, whose points closest to Q span a time interval greater than a user-defined threshold.

Outline. The rest of the paper is organized as follows. Section 2 formally defines the distance-based and bounded distance-based trajectory search while Sections 3 and 4 address their efficient evaluation. Then, Section 5 discusses problem variants where (a) the trajectories are ranked both on their distance to the query points and the time interval they span, and (b) Q is a sequence of query points, instead of a set. Section 6 presents our experimental analysis. Finally, Section 7 outlines related work, while Section 8 concludes the paper.

2 Problem Definition

Let T be a collection of trajectories. A trajectory in T is defined as a sequence of spatio-temporal points $\{p_1, \ldots, p_n\}$, each represented by a $\langle latitude, longitude, timestamp\rangle$ triple. The input of *point-based trajectory search* over collection T is



(a) Trajectory and query points (b) *span-dist* plot of trajectories

Fig. 1. Distance-based trajectory search with 4 trajectories, $T = \{t_1, \ldots, t_4\}$, and 3 query points, $Q = \{q_1, \ldots, q_3\}$; t_1, t_2 is the result to 2-DTS(T, Q), while t_2, t_3 the result to 2-BDTS(T, Q, τ)

a set of m spatial query points $Q = \{q_1, \ldots, q_m\}$. Given a query point $q_j \in Q$ and a trajectory $t_i \in T$, we define the $\langle p^*_{ij}, q_j \rangle$ *matching pair* based on the nearest to q_j point p^*_{ij} of trajectory t_i, i.e., $p^*_{ij} = \arg\min_{p \in t_i} dist(p, q_j)$, where $dist(\cdot, \cdot)$ denotes the distance (e.g., Euclidean) between two points in space. We then define the *distance* of a trajectory to Q based on the matching pairs for every query point q_j as:

$$dist(t_i, Q) = \sum_{q_j \in Q} dist(p^*_{ij}, q_j) \tag{1}$$

Consider the example in Figure 1(a), where query points are represented as diamonds, and trajectory points as circles; filled circles indicate matched points of the trajectory to query points. For trajectory t_1, point p^*_{11} is its closest point to query point q_1, and hence $\langle p^*_{11}, q_1 \rangle$ represents a matching pair. The other matched trajectory points of t_1 are p^*_{12} and p^*_{13}. Note that it is possible for a trajectory point to be matched with multiple query points. This is the case with trajectory t_3, where p^*_{32} is the closest point to both q_1 and q_2, i.e., $p^*_{31} \equiv p^*_{32}$.

We now formally define the *distance-based trajectory search* problem [3, 4].

Problem 1 (Distance-based Trajectory Search) *Given a collection of trajectories T and a set of query points Q, the k-*Distance-based Trajectory Search*, denoted by k-*DTS(T, Q)*, retrieves a subset of k trajectories $R \subseteq T$ such that for each $t \in R$ and $t' \in T \setminus R$, $dist(t, Q) \leq dist(t', Q)$ holds.*

Returning to the example of Figure 1(a), trajectory t_1 has the lowest distance to Q, followed by t_2, t_3 and t_4; hence, the result to 2-DTS(T, Q) is t_1, t_2.

Next, we introduce a novel point-based trajectory search problem by also taking into account the temporal aspect of the trajectories. Let P^*_i be the set of all matching pairs for a trajectory t_i, sorted ascending on the timestamp of the involved trajectory points. We define the *span* of trajectory t_i with respect to Q, denoted by $span(t_i, Q)$, as the length of the time interval between the first and the last pair in P_i^*, or equivalently:

$$span(t_i, Q) = \max_{q_x, q_y \in Q} (timestamp(p_{ix}^*) - timestamp(p_{iy}^*)) \quad (2)$$

Intuitively, $span(t_i, Q)$ equals the total time needed to reach as close as possible to all query points in Q, following trajectory t_i.

Problem 2 (Bounded Distance-based Trajectory Search) *Given a collection of trajectories T, a set of query points Q and a span threshold τ, the k-*Bounded Distance-based Trajectory Search*, denoted by k-*BDTS(T, Q, τ)*, retrieves the subset of k trajectories $R \subseteq T$ such that:*

- *for each $t \in R$, $span(t, Q) \leq \tau$ holds, and*
- *for each $t' \in T \setminus R$ with $span(t', Q) \leq \tau$, $dist(t, Q) \leq dist(t', Q)$ holds.*

Returning to Figure 1(a), assume for simplicity that trajectory points are reported in fixed time intervals. As a result, the span of a trajectory is proportional to the number of its points from the first to the last matched point (excluding the first). For example, $span(t_1, Q) = 4$ as there are 4 points from p_{11}^* and up to p_{13}^*. Similarly, we obtain the spans of t_2, t_3, t_4 as 2, 1, 2, respectively. Figure 1(b) plots the trajectories in the *span-dist* plane. DTS ignores the span values and simply returns the trajectories with the lowest *dist* coordinate. In contrast, BDTS introduces a threshold, e.g., $\tau = 3$, on the span of the trajectories, depicted as the dashed vertical line. Trajectories to the right of this line, i.e., t_1, do not qualify as BDTS results. Therefore, the result of 2-BDTS is t_2, t_3, i.e., the trajectories with the 2 lowest distances among those left of the line. Notice that BDTS may not return the trajectory with the lowest distance to Q if its span exceeds the threshold; e.g., t_1 in Figure 1.

Depending on the application, one may consider alternative definitions for point-based trajectory search that take into account both the distance and the span metrics. We briefly overview one of them in Section 5, where we also discuss trajectory search given a sequence of query points, instead of a set.

3 Distance-based Trajectory Search

We first discuss trajectory search based on the distance to a set of query points. Section 3.1 revisits existing work, while Sections 3.2 and 3.3 present our NN-based and spatial range-based methods, respectively.

3.1 Existing Methods

Methods `IKNN` [3] and `GH/QE` [4] have previously tackled distance-based trajectory search. Note that in [3] the problem was defined with respect to the similarity of a trajectory t_i to the set of query points Q, defined as $sim(t_i, Q) = \sum_{q_j \in Q} e^{-dist(p_{ij}^*, q_j)}$. In what follows, we describe the straightforward adaptation of the `IKNN` algorithm for the distance metric of Equation (1) (which was also

Algorithm 1: IKNN

```
  Input     : collection of trajectories $T$, set of query points $Q$, number of results $k$
  Output    : result set $R$
  Variables : candidate set $C$, $k$-th distance upper bound $UB_k$, distance lower bound $LB$
1 initialize $C \leftarrow \emptyset$, $UB_k \leftarrow \infty$ and $LB \leftarrow 0$;
2 while $UB_k > LB$ do
3     for each $q_j \in Q$ do
4         $\delta_j$-NN($q_j$) $\leftarrow$ the next $\delta_j$ nearest trajectory points to $q_j$;
5         update $C$ with $\delta_j$-NN($q_j$);
6         update $UB_k$ and $LB$;                          ▷ Equations (3) and (4)
7 $R \leftarrow$ Refine_DTS$(k, T, Q, C)$;
8 return $R$;
```

used in [4]). The adaptation of GH/QE and our methods (Sections 3.2 and 3.3) to the similarity metric of [3] is also straightforward and therefore, omitted. Moreover, the relative performance of all methods is identical independent of the metric used.

All existing methods adopt a *candidate generation* and *refinement* evaluation paradigm. During the first phase, a set of candidate trajectories is determined by incrementally retrieving the nearest trajectory points to the query points in Q. For this purpose, the methods utilize a single R-tree to index all trajectory points. A candidate trajectory t is called a *full match* if the matching pairs of t to all query points in Q have been identified; otherwise, t is a *partial* match. As soon as the candidate set is guaranteed to include the final results (even as partial matches), candidate generation is terminated, and the refinement phase is then employed to identify and output the results.

The IKNN Algorithm. Note that the IKNN algorithm comes in two flavors; in the following, we consider the one based on best-first search, as it was shown in [3] to be both faster and require fewer I/O operations. Algorithm 1 shows the pseudocode of IKNN. During candidate generation (Lines 2–6), the algorithm iterates over the points of Q in a round robin manner. For each query point q_j, the (next) batch of nearest to q_j trajectory points is retrieved using the R-tree index, in Line 4. The nearest neighbor search retrieves a different number of trajectory points δ_j per query point q_j, in order to expedite the termination of this first phase (details in [3]). Based on the newly identified matching pairs that involve q_j, the set of candidates C is then updated in Line 5 by either adding new partial matches or filling an empty slot for existing. For each partial match t_i in C, IKNN computes an *upper bound* of its distance to Q by setting the distance of t_i to every unmatched query point equal to the diameter of the space (maximum possible distance between two points): [4]

$$\overline{dist}(t_i, Q) = \sum_{q_j \in Q_i} dist(p^*_{ij}, q_j) + |Q \setminus Q_i| \cdot DIAM, \tag{3}$$

where set $Q_i \subseteq Q$ contains all the query points already matched to a point in trajectory t_i. We denote by UB_k the k-th smallest among the distance bounds

[4] Under the similarity-based definition of DTS in [3], IKNN sets empty "slots" to 0.

Algorithm 2: `GH`

```
Input      : collection of trajectories $T$, set of query points $Q$, number of results $k$
Output     : result set $R$
Variables  : candidate set $C$, global heap $H$
1 initialize $C \leftarrow \emptyset$ and $H \leftarrow \emptyset$;
2 while $C$ contains less than $k$ full matches do
3     pop $\langle p_{ij}, q_i \rangle$ from $H$; ▷ Get the globally nearest trajectory point to some query point
4     update $C$ with $\langle p_{ij}, q_i \rangle$;
5     push to $H$ the next nearest trajectory point to $q_i$;
6 $R \leftarrow$ Refine_DTS$(k, T, Q, C)$;
7 return $R$;
```

Algorithm 3: `QE`

```
Input      : collection of trajectories $T$, set of query points $Q$, number of results $k$
Output     : result set $R$
Variables  : candidate set $C$, global heap $H$, distance lower bound $LB$
1 initialize $C \leftarrow \emptyset$, $H \leftarrow \emptyset$ and $LB \leftarrow 0$;
2 while $C$ contains less than $k$ full matches with $dist(\cdot, Q) \geq LB$ do
3     pop $\langle p_{ij}, q_i \rangle$ from $H$; ▷ Get the globally nearest trajectory point to some query point
4     update $C$ with $\langle p_{ij}, q_i \rangle$;
5     push to $H$ the next nearest trajectory point to $q_i$;
6     complete the most promising partial matches in $C$;              ▷ Equation (5)
7     update $LB$;                                                      ▷ Equation (6)
8 $R \leftarrow$ Refine_DTS$(k, T, Q, C)$;
9 return $R$;
```

for the trajectories in C. In addition, `IKNN` computes a *lower bound* LB of the distance to Q for all unseen trajectories (i.e., those not contained in C), by aggregating the distance of the farthest (retrieved so far) trajectory point to each query point in Q. Formally:

$$LB = \sum_{q_j \in Q} dist(p_j^{\delta}, q_j) \tag{4}$$

where p_j^{δ} is the last trajectory point returned by the NN search centered at q_j.

The candidate generation phase of `IKNN` terminates when $UB_k \leq LB$; in this case, none of the unseen trajectories can have smaller distance to Q compared to the candidates in C. Last, `IKNN` invokes `Refine`$_{\texttt{DTS}}$ to produce the results. Briefly, the function examines candidates in ascending order of a lower bound on their distance, retrieving them from disk to compute $dist(\cdot, Q)$ (details in [3]).

The `GH`/`QE` Algorithms. Different from `IKNN`, the methods in [4] retrieve trajectory points in ascending order of the distance to their closest query point. Specifically, a *global heap* H is used to retrieve at each iteration the *globally* nearest trajectory point p_{ij} to some query point q_j, and then, to update candidate set C, accordingly. Algorithm 2 shows the pseudocode of `GH`. The candidate generation phase of `GH` is terminated as soon as set C contains k full matches (proof of correctness in [4]). Note that these full matches are not necessary among the final results identified in Line 6 during the refinement phase.

In practice, the order imposed by global heap H cannot guarantee a good performance unless both trajectory and query points are uniformly distributed in space. For instance, if a particular query point is very close to many trajectories, `GH` will generate a large number of partial matches with only that slot filled. Consequently, it will take longer to produce the k full matches needed to terminate the generation phase, and at the same time a large number of candidates would have to be refined. A similar problem occurs when a query point is located away from the trajectories.

To address these issues, Tang et al. [4] proposed an extension to `GH` termed `QE`, which periodically fills the empty slots for the partially matched trajectories with the highest potential of becoming results. These are then retrieved from disk, and their actual distance is computed. A trajectory has high potential if it has (i) few empty slots and (ii) small distance in each filled slot with respect to the next point to be retrieved for that slot. These factors are captured respectively by the denominator and enumerator of the following equation:

$$potential(t_i) = \frac{\sum_{q_j \in Q_i} \left(dist(p_j^H, q_j) - dist(p_{ij}^*, q_i)\right)}{|Q \setminus Q_i|} \tag{5}$$

where set $Q_i \subseteq Q$ contains all the query points already matched to a point in t_i, p_j^H is the next nearest trajectory point to q_j contained in heap H and p_{ij}^* is the nearest to q_j point in trajectory t_i.

Algorithm 3 shows the pseudocode of `QE`. The candidate generation phase of `QE` terminates when candidate set C contains k full matches (similar to `GH`), provided however that their distance to Q is smaller than the distance of all unseen trajectories (Line 2) (proof of correctness in [4]). To determine this, `QE` computes in Line 7, a *lower bound* LB of the distance for the unseen trajectories (similar to `IKNN`) by aggregating the distance of the next nearest trajectory point to every query point, i.e., the contents of heap H:

$$LB = \sum_{q_j \in Q} dist(p_j^H, q_j) \tag{6}$$

3.2 A Hybrid NN-based Approach

The DTS problem can be viewed as a top-k query [5, 6]. For each query point q_j, consider a sorted trajectory list T_j, where each trajectory is ranked according to its distance to the query point. Then, the objective is to determine the top-k trajectories that have the highest aggregate score, i.e., distance, among the lists. However, as these lists are not given in advance and constructing them is costly, the goal is to progressively materialize them, until the result is guaranteed to be among the already seen trajectories.

Following the top-k query processing terminology, a *sorted access* on list T_j corresponds to the retrieval of the next nearest trajectory to query point q_j, which in turn may involve multiple trajectory point NN retrievals. In contrast, a *random access* for trajectory t_i on list T_j corresponds to the retrieval of t_i from disk and the computation of its distance to q_j; in practice, once t_i is retrieved, its distance to all query points can be computed at negligible additional cost.

Methods `IKNN`, `GH` and `QE` employ various ideas from top-k query processing (an overview of this field is presented in Section 7). Particularly, `IKNN` performs only sorted accesses and prioritizes them in a manner similar to `Stream–Combine` [7]. Similarly, `GH` performs only sorted accessses but follows an unconventional strategy for prioritizing them, which explains its poor performance on our tests in Section 6. On the other hand, `QE` additionally performs random accesses following a strategy similar to the `CA` algorithm [5] to select which trajectory to retrieve.

In the following, we present the `NNA` algorithm, which combines the strengths of `IKNN` and `QE`. In short, it builds upon the `Quick–Combine` top-k algorithm [8] performing both sorted and random accesses to generate the candidate set. `NNA` has the following features. First, similar to `IKNN`, the algorithm retrieves in a round robin manner, batches of nearest trajectory points to each query point in Q. This addresses the weaknesses of `GH` when dealing with non-uniformly distributed data. Second, after performing the nearest neighbor search centered at each query point, `NNA` fills the slots of the trajectories with the highest potential according to Equation (5), similar to `QE`. Finally, `NNA` employs the termination condition of `IKNN` for the candidate generation phase. In practice, `NNA` extends Algorithm 1 by completing the most promising partial matches in C (similar to `QE`), between Lines 5 and 6. Hence, it is able to compute tighter bounds compared to `IKNN` and thus terminate the generation phase earlier. In addition, it produces fewer candidates than `IKNN`, reducing the cost of the refinement phase.

3.3 A Spatial Range-based Approach

We identify two shortcomings of all the NN-based methods previously described. First, each NN search is implemented independently, which means that R-tree nodes and trajectory points may be accessed multiple (up to $|Q|$) times, which increases the total I/O cost. Second, each NN search is associated with a priority queue, whose continuous maintenance increases the total CPU cost.

Our novel *Spatial Range-based* algorithm, denoted by `SRA`, addresses both these shortcomings. Similar to the NN-based approaches, it follows a generation and refinement paradigm. However, to generate the candidate set, it issues a spatial range search of expanding radius centered at each query point in Q. All searches operate on a common set N of R-tree nodes, which avoids accessing nodes more than once and hence saves I/O operations. Moreover, set N needs not be sorted according to any distance, eliminating costly priority queue maintenance tasks. The range-based search for each query point q_j is associated with *current radius* r_j, and is also assigned a *maximum radius* θ_j. As the algorithm progresses, current radius r_j increases while maximum radius θ_j decreases. Candidate generation terminates as soon as $r_j > \theta_j$ for some query point q_j.

Algorithm 4 shows the pseudocode of `SRA`. In Lines 2–4, `SRA` initializes the current and maximum radius for each query point. For the latter, an upper bound UB_k to the k-th smallest distance to Q is computed. In particular, `SRA` invokes a sum-aggregate nearest neighbor (sum-ANN) procedure [9] retrieving trajectory points in ascending order of $\sum_{q_j \in Q} dist(\cdot, q_j)$. Assuming that this procedure retrieves point p_i of trajectory t_i, the sum-aggregate value is an upper bound to

Algorithm 4: SRA

```
Input     : collection of trajectories $T$, set of query points $Q$, number of results $k$
Output    : top-$k$ list of trajectories $R$
Variables : candidate set $C$, $k$-th distance upper bound $UB_k$, current $r_i$ and maximum $\theta_i$
            search radius for each $q_i \in Q$, set of R-tree nodes $N$
1  initialize $C \leftarrow \emptyset$ and $N \leftarrow$ R-tree root node;
2  compute $UB_k$ invoking a sum-ANN($T, Q$);
3  for each $q_j \in Q$ do
4      initialize $r_j \leftarrow 0$ and $\theta_j \leftarrow UB_k$;
5  while $r_j \leq \theta_j$ for all $q_j \in Q$ do
6      select current $q_c$;
7      $r_c \leftarrow r_c + \xi$ ;                         ▷ Increase $r_c$ to expand search around $q_c$
8      expand from $N$ all nodes that intersect with the disc of radius $r_c$ centered at $q_c$;
9      $S \leftarrow$ trajectory points within spatial range $r_c$ found during expansion;
10     update $C$ with $S$;
11     update $UB_k$;                                                          ▷ Equation (7)
12     for each $q_j \in Q$ do
13         update $\theta_j \leftarrow UB_k - \sum_{q_\ell \in Q \setminus \{q_j\}} r_\ell$;      ▷ Reduce maximum radius
14 $R \leftarrow$ Refine$_{\texttt{DTS}}$($k, T, Q, C$);
15 return $R$;
```

the distance of t_i, i.e., $dist(t_i) \leq \sum_{q_j \in Q} dist(p_i, q_j)$. Hence, once points from k distinct trajectories have been retrieved, SRA can determine a value for UB_k.

During the candidate generation phase in Lines 5–13, SRA first selects the query point $q_c \in Q$ with the fewest retrieved points so far, and increases its radius by a fixed ξ[5], so that each location retrieves more or less the same number of points. Then, it extends the range search centered at q_c to new radius r_c. In particular, all nodes in N that intersect with the search frontier are expanded, i.e., replaced by their children (Line 8). During the expansion, all trajectory points within the frontier are collected in set S (Line 9). Upon completion of the expansion, set N contains no R-tree node or point within r_c distance to q_c, or with distance to q_c greater than θ_c, and N will be re-used in further iterations.

After the expansion, SRA uses the newly seen trajectory points in S to properly update candidate set C. Note that for each trajectory t_i in C, SRA keeps $|Q|$ slots storing the closest trajectory points $t_i.p_j$ seen so far to each query point q_j. A slot is marked *matched* if the corresponding matching pair has been determined, i.e., when $t_i.p_j \equiv p^*_{ij}$. SRA in Line 10 performs the following tasks for each point p_x in S; let t_i be the trajectory p_x belongs to. For each slot q_j that is not *matched*, SRA checks whether p_x is closer to q_j than $t_i.p_j$, and updates the slot with p_x if true. If the slot for the current query point q_c was among those examined, it is marked as *matched*. The benefits of this update strategy are twofold. First, it guarantees that no matching trajectory point will be missed, even though SRA does not access p_x again (removed from N) for $q_j \neq q_c$. At the same time, it also helps to derive a tighter upper bound for the distance of t_i:

$$\overline{dist}(t_i, Q) = \sum_{q_j \in Q_i} dist(p^*_{ij}, q_j) + \sum_{q_j \in Q \setminus Q_i} dist(t_i.p_j, q_j). \tag{7}$$

[5] In the future, we plan to investigate variable ξ_j values based on current radius r_j and the trajectory point density around q_j, inspired by determining δ_j value in [3].

Compared to Equation (3) utilized by `IKNN` and `NNA`, Equation (7) computes a tighter bound on unmatched slots. Based on these bounds, a tighter value for UB_k can be established (Line 11).

To better explain the procedure in Line 10, we use the example of Figure 1(a) for $k = 2$. `SRA` has just started and thus C is empty. Assume that the current query point is $q_c = q_1$, and let $r_1 = 0 + \xi$ be the radius of the shaded disk depicted in the figure. As a result, set S in Line 9 contains trajectory points $\{p^*_{21}, p^*_{22}, p^*_{41}\}$. Moreover, candidate set C contains t_2 and t_4. For trajectory t_2, p^*_{21} is settled as the matching point to q_1 because $dist(p^*_{21}, q_1) < dist(p^*_{22}, q_1)$ and no unseen point of t_2 can be closer. On the other hand, the matching points to q_2, q_3 cannot be yet determined, but we can use p^*_{21} and p^*_{22} to bound t_2's distances to q_2 and q_3. Therefore, the slots for t_2 become $\langle \mathbf{p^*_{21}}, p^*_{22}, p^*_{21}\rangle$, where bold indicates a *matched* slot. Moreover, an upper bound to the distance of t_2 is determined as $\overline{dist}(t_2, Q) = dist(p^*_{21}, q_1) + dist(p^*_{22}, q_2) + dist(p^*_{21}, q_3)$. Similarly, we obtain the slots for t_4 as $\langle \mathbf{p^*_{41}}, p^*_{41}, p^*_{41}\rangle$.

As a last step, `SRA` updates the maximum radius for all query points with respect to the new UB_k in Lines 12–13. Observe that `SRA`'s termination condition for candidate generation is essentially identical to that of `IKNN`. Any trajectory not in the candidate set C must have distance to each q_j at least θ_j, and thus distance at least equal to $LB = \sum_{q_j \in Q} \theta_j$. The termination condition of Line 5, $r_j > \theta_j$ for some q_j, and the update of θ_j, imply that, when candidate generation concludes, $UB_k \leq LB$.

Finally, the performance of `SRA` can be enhanced following the key idea of `QE` to further improve the $\overline{dist}(t_j, Q)$ bound and therefore, UB_k. We denote this extension to the `SRA` algorithm by `SRA`+. Specifically, in between Lines 10 and 11 in Algorithm 4, `SRA`+ fills the empty slots of the trajectories in C with the highest potential as computed using Equation (5).

4 Bounded Distance-based Trajectory Search

We next address the bounded distance-based trajectory search. Recall from Section 2 that k-BDTS(T, Q, τ) is equivalent to a k-DTS(T', Q) distance-based query over the subset $T' \subseteq T$ containing only trajectories with $span(\cdot, Q) \leq \tau$. However, as $span(t, Q)$ can be computed only after all the matching pairs of a trajectory t to Q are identified, the major challenge is to limit the number of invalid partial matches generated, i.e., those with the $span(\cdot, Q) > \tau$. In the following, we address this issue in two alternative ways.

The idea behind the incremental approach, denoted as `INCREMENTAL`, is to progressively construct the result set R by utilizing the generation phase of a DTS method as a "black" box. Algorithm 5 illustrates `INCREMENTAL`; note that any of the algorithms in Section 3 can be used as the underlying DTS method. At each round, `INCREMENTAL` asks for the missing $k - |R|$ trajectories to complete the result set R in Lines 3–4. For this purpose, a λ-DTS(T, Q) search is processed, with the λ value been increased at each round by $k - |R|$; during the first round $\lambda = k$. Each time λ is updated in Line 3, the DTS method in Line 4 does not

Algorithm 5: `INCREMENTAL`

```
Input     : collection of trajectories $T$, set of query points $Q$, span threshold $\tau$, number
            of results $k$
Output    : result set $R$
Variables : candidate set $C$, number of intermediate results $\lambda$
1 initialize $C \leftarrow \emptyset$, $R \leftarrow \emptyset$ and $\lambda \leftarrow 0$;
2 while $|R| < k$ do
3     increase $\lambda$ by $k - |R|$;
4     $C \leftarrow$ next candidate set of $\lambda$-DTS$(T, Q)$;
5     $R \leftarrow R \cup \texttt{Refine}_{\texttt{BDTS}}(k, T, Q, C, \tau)$;
6 return $R$;
```

run from scratch. It continues the candidate generation using a new termination condition with respect to the updated λ in order to expand candidate set C. Last, in Line 5, `Refine`$_{\texttt{BDTS}}$ examines the new candidates to update result set R by computing their $dist(\cdot, Q)$ and eliminating trajectories with $span(\cdot, Q) > \tau$.

Intuitively, `INCREMENTAL` takes a conservative approach to bounded distance-based trajectory search. As it is unable to predict which partial matches could provide a valid trajectory (full match) with $span(\cdot, Q) \leq \tau$, a refinement phase is needed to "clean" the candidate set. Hence, `INCREMENTAL` may involve several rounds of generation and refinement phases. To address these issues, we propose the `ONE–PASS` approach which involves a single generation and refinement round. The idea is again to build upon a DTS method but by extending its candidate generation phase in two ways. First, for each partial match t_i in candidate set C, `ONE–PASS` computes a lower bound of $span(t_i, Q)$ based on the points of t_i matching the current subset of query points $Q_i \subset Q$, as follows:

$$\underline{span}(t_i, Q) = \begin{cases} 0, & \textit{if } |Q_i| = 1 \\ span(t_i, Q_i), & \textit{otherwise} \end{cases} \tag{8}$$

Every partial match with $\underline{span}(\cdot, Q) > \tau$ can be safely pruned. Second, the original termination is triggered only after candidate set C contains at least k valid full matches, i.e., with $span(\cdot, Q) \leq \tau$. This is because the k-th upper bound UB_k of existing candidates can be computed only through full matches. For example, candidate generation of `ONE–PASS` based on `SRA+` terminates as soon as at least k valid full matches are identified and $r_j > \theta_j$ holds for some query point q_j.

5 Discussion

We discuss alternative definitions and variants to the point-based search problems introduced in Section 2.

Distance & Span-based Trajectory Search. Although taking into account their temporal span, the bounded distance-based search still ranks the trajectories solely on their distance to the query points in Q. As an alternative, we may rank the results with respect to a linear combination of the *span-dist* metrics:

$$f(t, Q) = \alpha \cdot dist(t, Q) + (1 - \alpha) \cdot span(t, Q) \tag{9}$$

where α weights the importance of each metric. With Equation (9), we introduce the *k-Distance & Span-based Trajectory Search*, denoted by k-DSTS(T, Q) which returns the subset of k trajectories $R \subseteq T$ with the lowest $f(\cdot, Q)$ value.

All methods discussed in Section 3 can be extended for k-DSTS(T, Q) by replacing $dist(\cdot, Q)$ with $f(\cdot, Q)$. Note that the upper bound $\overline{f}(t, Q)$ of a partial match t can be computed by setting $\overline{span}(t, Q)$ equal to the total duration of the trajectory t. In contrast, as no matching pairs are identified for the unseen trajectories, the lower bound LB or the θ_j values are defined similar to the DTS methods, i.e., essentially setting the lower bound of span to zero. In Section 6.4, we experimentally investigate the efficient evaluation of DSTS.

Order-aware Trajectory Search. Similar to [3], we also consider a variation of the trajectory search when a visiting order is imposed for the query points. In this variation, the matched trajectory point p^*_{ij} to query point q_j, is not necessarily the nearest to q_j point of trajectory t_i. Consider for example trajectory t_2 in Figure 1. The depicted p^*_{22}, p^*_{21}, p^*_{23} for DTS cannot be the matched points in the $q_1 \to q_2 \to q_3$ order-aware DTS, as they violate the visiting order. Instead, the matched points that preserve the imposed visiting order are p^*_{22}, p^*_{22}, p^*_{23}, where p^*_{22} is matched with q_1 although $dist(p^*_{22}, q_1) > dist(p^*_{21}, q_1)$. The distance of a trajectory to sequence Q is recursively defined as follows:

$$dist_o(t, Q) = \begin{cases} \min \begin{cases} dist_o(t, T(Q)) + dist(H(t), H(Q)) - DIAM \\ dist_o(T(t), Q) \end{cases} & \text{if } t \neq \varnothing, Q \neq \varnothing \\ |Q| \cdot DIAM & \text{if } t = \varnothing \\ 0 & \text{if } Q = \varnothing \end{cases} \tag{10}$$

where $H(S)$ is the first point (head) in a sequence S, $T(S)$ indicates the tail of S after removing $H(S)$, $\varnothing$ denotes the empty sequence, and $DIAM$ represents the diameter of the space. The distance can be computed by straightforward dynamic programming [3]. To derive an upper bound on a partial matched trajectory t_i, we consider only the subsequence Q_i of Q that contains the matched query points, i.e., $\overline{dist_o}(t_i, Q) = dist_o(t_i, Q_i)$. For order-aware BDTS, distance and its upper bound are the same as in order-aware DTS. Note, however that the lower bound on span (Equation (8)) does not apply as the matching are not yet finalized. For order-aware DSTS evaluation, $f_o(t, Q)$ and its upper bound are defined in a similar manner to order-aware DTS. In Section 6, we experimentally investigate the order-aware variants of all three trajectory search problems.

6 Experimental Analysis

We evaluate our methods for point-based trajectory search. All algorithms were implemented in C++ and the tests run on a machine with Intel Core i7-3770 3.40GHz and 16GB main memory running Ubuntu Linux.

Table 1. POIs in Beijing

category	cardinality
Restaurants	51,971
Hotels	10,620
Pharmacies	6,963
Schools	6,618
Banks	6,057
Police stations	2,509
Supermarkets	2,356
Gas stations	1,916
Post offices	1,125

Table 2. Experimental parameters (default values in bold)

description	parameter	values
Number of results	k	1, 5, **10**, 50, 100
Number of query points	$\|Q\|$	2, 4, **6**, 8, 10
Span threshold ratio	τ/τ_{min}	1, 1.5, **2**, 2.5, 3
Linear combination factor	α	0, 0.25, **0.5**, 0.75, 1

6.1 Setup

We conducted our analysis using real-world trajectories from the GeoLife Project [10–12]. The collection contains 17,166 trajectories with 19m points in Beijing, recording a broad range of outdoor movement. To generate our query sets, we considered around 90k points of interest (POIs) of various types, located inside the same area covered by the trajectories (see Table 1 for details). A query set Q is formed by randomly selecting a combination of $|Q|$ types and a particular POI from each type. We assess the performance of all involved methods measuring their CPU and I/O cost, and the number of candidates they generate over 1,000 distinct query sets Q, while varying (i) the number of returned trajectories k and (ii) the number of query points $|Q|$. In case of BDTS queries, we additionally vary the span threshold via the τ/τ_{min} ratio, where τ_{min} is the minimum possible time required to travel among the query points in Q at a constant velocity of 50km/h. Finally, for DSTS queries, we also vary the weight factor α of Equation (9). Table 2 summarizes all parameters involved in our study.

6.2 Distance-based Trajectory Search

Figure 2 reports the CPU cost, the I/O cost and the number of generated candidates for the DTS methods. As expected the processing cost of all methods goes up as the values of k and $|Q|$ increase. The tests clearly show that `SRA`+ is overall the most efficient evaluation method. We also make the following observations.

First, we observe that `IKNN` always outperforms `GH`/`QE`; note that this is the first time the methods from [3, 4] are compared. Naturally, `GH` comes as the least efficient method; due to the examination order imposed by global heap H, the algorithm is unable to cope with the skewed distribution of the real-world data. `QE` manages to overcome the shortcomings of `GH` by completing the empty slots of the most promising candidates. Yet, compared to `IKNN`, `QE` is less efficient due to its weak termination condition for the generation phase; recall that at least k full matches are needed for this purpose which also results in generating a larger number of candidates, as shown in Figures 2(c) and (f). The advantage of `IKNN` over `GH`/`QE` justifies our decision to build the hybrid `NNA` method upon the round robin-based candidate generation of `IKNN` which retrieves nearest neighbor points in batches, and its powerful threshold-based termination condition. `NNA` is indeed the most efficient NN-based method, in fact with an order of magnitude improvement over `IKNN` and `GH`/`QE` on both CPU and I/O cost. Finally, Figure 2

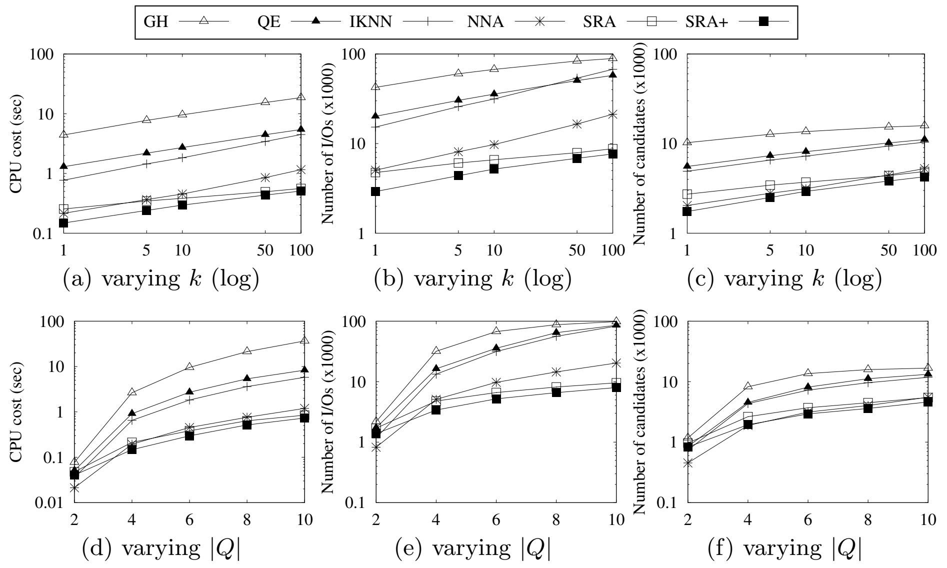


Fig. 2. Performance comparison for Distance-based Trajectory Search

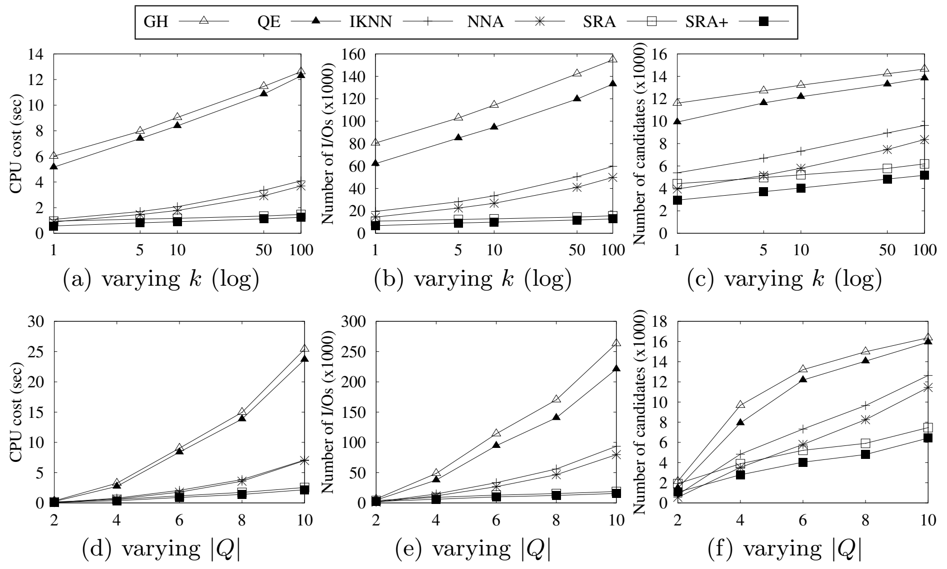


Fig. 3. Performance comparison for Distance-based Trajectory Search (order-aware)

clearly shows the advantage of the spatial range-based evaluation approach over the NN-based one. `SRA` is always faster while incurring fewer disk page accesses than `IKNN`, and in a similar manner, `SRA+` outperforms `NNA`.

We also experimented with the order-aware variant of DTS. Figure 3 depicts similar results to Figure 2; the spatial range-based evaluation approach is again superior to the NN-based and overall, `SRA`+ is the most efficient method. Nevertheless, it is important to notice that the advantage of completing the most proposing candidates is smaller compared to Figure 2, in terms of the CPU cost. Specifically, observe how close is the running time of `GH` to `QE`, of `IKNN` to `NNA` and of `SRA` to `SRA`+, in Figures 3(a) and (d). This is expected as completing partial matches employs dynamic programming to compute $dist_o(\cdot, Q)$.

6.3 Bounded Distance-based Trajectory Search

Next, we investigate the evaluation of BDTS queries while varying the k, $|Q|$ and τ/τ_{min} parameters. Based on the findings of the previous section, we use the `SRA`+ algorithm as the underlying DTS method. Note that due to lack of space we omit results for the order-aware variant of BDTS; the results however are similar. Figure 4 clearly shows that `ONE–PASS` outperforms `INCREMENTAL` in all cases. As expected, the conservative approach of `INCREMENTAL` generates a larger number of candidates by performing multiple rounds of generation and refinement which results in both higher running time and more disk page accesses. Last, notice that the evaluation of BDTS becomes less expensive for both methods while increasing τ/τ_{min}, as the number of invalid candidates progressively drops.

6.4 Distance & Span-based Trajectory Search

Finally, we study the evaluation of DSTS queries. For this experiment, we extended the most dominant method from [3, 4], i.e., `IKNN`, and our methods `NNA`, `SRA` and `SRA`+ following the discussion in Section 5. The results in Figure 5 demonstrate, similar to the DTS case, the advantage of both the spatial range-based approach and the `SRA`+ algorithm which is overall the most efficient evaluation method. Due to lack space, we again omit the figure for the order-aware variant of DSTS as the results are identical to Figure 5.

7 Related Work

Apart from the studies [3, 4] for distance-based search on trajectories detailed in Section 3.1, our work is also related to *top-k* and *nearest neighbor* queries.

Top-k Queries. Consider a collection of objects, each having a number of scoring attributes, e.g., rankings. Given an aggregate function γ (e.g., SUM) on these scoring attributes, a top-k query returns the k objects with the highest aggregated score. To evaluate such a query, a *sorted* list for each attribute a_i organizes the objects in decreasing order of their value to a_i; requests for *random accesses* of an attribute value based on object identifiers may be also possible. Ilyas et al. overviews top-k queries in [6] providing a categorization of the proposed methods. Specifically, when both sorted and random accesses are possible, the `TA/CA` [5] and `Quick–Combine` [8] algorithms can be applied. `TA` retrieves objects from the sorted lists in a round-robin fashion while a priority queue to organizes the

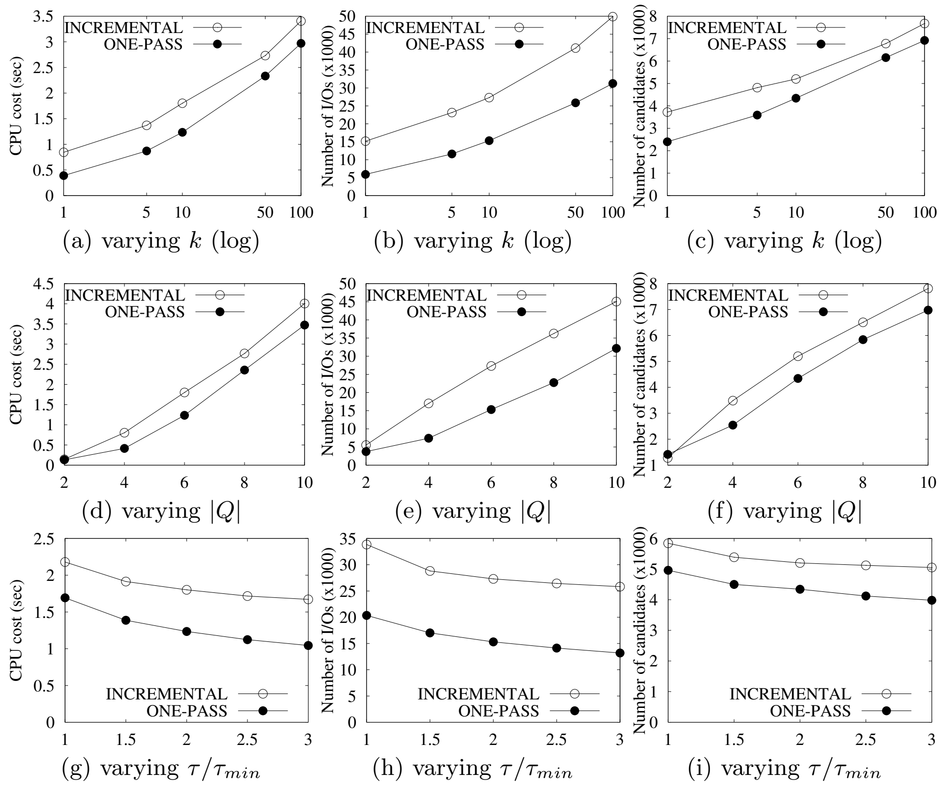


Fig. 4. Performance comparison for Bounded Distance-based Trajectory Search

best k objects so far. Based on the last seen attribute values, the algorithm defines an upper score bound for the unseen objects, and terminates if current k-th highest aggregate score is higher than this threshold. `TA` assumes that the costs of the two different access methods are the same. As an alternative, `CA` defines a ratio between these costs to control the number of random accesses, which in practice are usually more expensive than sorted accesses. Hence, the algorithm periodically performs random accesses to collect unknown values for the most "promising" objects. Last, the idea behind `Quick–Combine` is to favor accesses from the sorted lists of attributes which significantly influence the overall scores and the termination threshold. In contrast, when only sorted accesses are possible, the `NRA` [5] and `Stream–Combine` [7] algorithms can be applied. Intuitively, `Stream–Combine` operates similar to `Quick–Combine` without performing any random accesses. In Section 3.1, we discuss how the methods in [3, 4] build upon previous work on top-k queries to address distance-based search on trajectories.

Nearest Neighbor Queries. There is an enormous amount of work on the *nearest neighbor* (NN) query (also known as similarity search), which returns the object that has the smallest distance to a given query point; k-NN queries output the k nearest objects in ascending distance. Roussopoulos et al. proposed

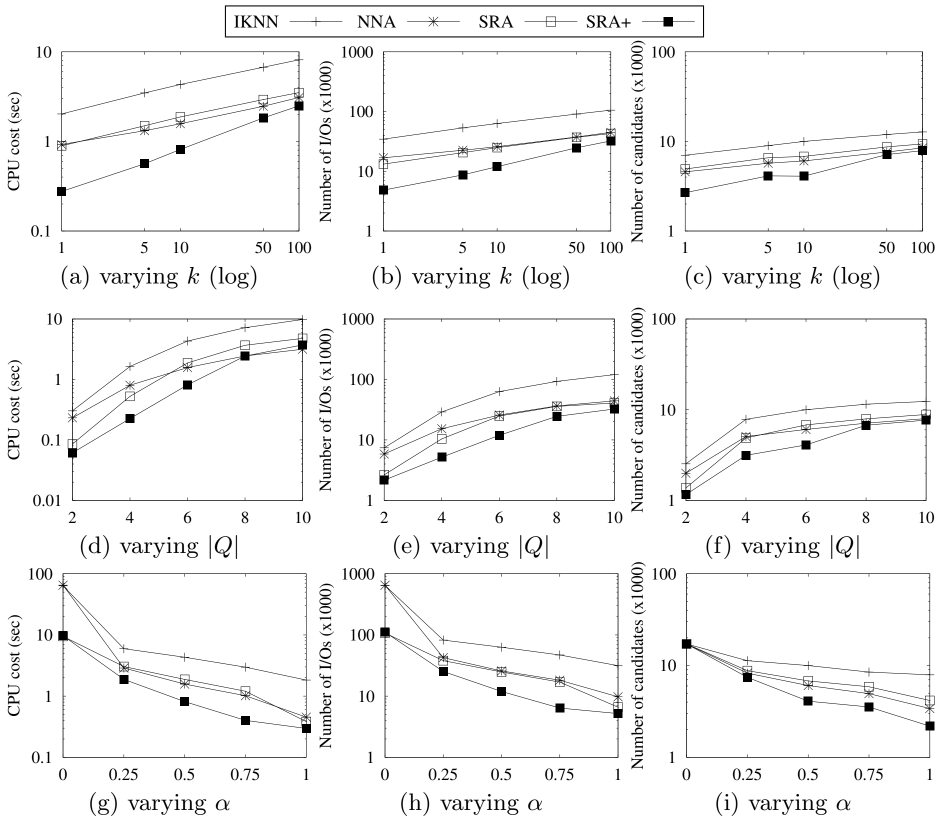


Fig. 5. Performance comparison for Distance & Span-based Trajectory Search

a depth-first approach to k-NN query in [13] while Hjaltason et al. enhanced the evaluation with a best-first search strategy in [14]. An overview of index-based approaches can be found in [15]; efficient methods for metric spaces, e.g., [16], and high-dimensional data, e.g., [17], have also been proposed.

For a set of query points, the *aggregate nearest neighbor* (ANN) query [9] retrieves the object that minimizes an aggregate distance to the query points. As an example, for the MAX aggregate function and assuming that the set of query points are users, and distances represent travel times, ANN outputs the location that minimizes the time necessary for all users to meet. In case of the SUM function and Euclidean distances, the optimal location is also known as the Fermat-Weber point, for which no formula for the coordinates exists.

8 Conclusions

In this paper, we studied the efficient evaluation of point-based trajectory search. After revisiting the existing methods (`IKNN` and `GH/QE`), which examine the trajectories in ascending order of their distance to the queries points, we devised a hybrid algorithm which outperforms them by a wide margin. Then, we proposed a spatial range-based approach; our experiments on real-world trajectories showed that this approach outperforms any NN-based method. Besides improving the performance of distance-based search, we also introduced and investigated the evaluation of a practical variant for point-based trajectory search, which also takes into account the temporal aspect of the trajectories. As a direction for future work, we plan to consider additional types of annotated data on the trajectories in point-based search, such as textual and social information.

References

1. Zheng, Y.: Trajectory data mining: An overview. ACM Transaction on Intelligent Systems and Technology (September 2015)
2. Zheng, Y., Zhou, X., eds.: Computing with Spatial Trajectories. Springer (2011)
3. Chen, Z., Shen, H.T., Zhou, X., Zheng, Y., Xie, X.: Searching trajectories by locations: an efficiency study. In: SIGMOD. (2010) 255–266
4. Tang, L.A., Zheng, Y., Xie, X., Yuan, J., Yu, X., Han, J.: Retrieving k-nearest neighboring trajectories by a set of point locations. In: SSTD. (2011) 223–241
5. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: PODS. (2001) 102–113
6. Ilyas, I.F., Beskales, G., Soliman, M.A.: A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. **40**(4) (2008)
7. Güntzer, U., Balke, W., Kießling, W.: Towards efficient multi-feature queries in heterogeneous environments. In: ITCC. (2001) 622–628
8. Güntzer, U., Balke, W.T., Kießling, W.: Optimizing multi-feature queries for image databases. In: VLDB. (2000) 419–428
9. Papadias, D., Tao, Y., Mouratidis, K., Hui, C.K.: Aggregate nearest neighbor queries in spatial databases. ACM Trans. Database Syst. **30**(2) (2005) 529–576
10. Zheng, Y., Zhang, L., Xie, X., Ma, W.Y.: Mining interesting locations and travel sequences from gps trajectories. In: WWW. (2009) 791–800
11. Zheng, Y., Li, Q., Chen, Y., Xie, X., Ma, W.: Understanding mobility based on GPS data. In: UbiComp 2008: Ubiquitous Computing, 10th International Conference, UbiComp 2008, Seoul, Korea, September 21-24, 2008, Proceedings. (2008) 312–321
12. Zheng, Y., Xie, X., Ma, W.: Geolife: A collaborative social networking service among user, location and trajectory. IEEE Data Eng. Bull. **33**(2) (2010) 32–39
13. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995. (1995) 71–79
14. Hjaltason, G.R., Samet, H.: Distance browsing in spatial databases. ACM Trans. Database Syst. **24**(2) (1999) 265–318
15. Böhm, C., Berchtold, S., Keim, D.A.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Comput. Surv. **33**(3) (2001) 322–373
16. Jagadish, H.V., Ooi, B.C., Tan, K., Yu, C., Zhang, R.: iDistance: An adaptive b$^+$-tree based indexing method for nearest neighbor search. ACM Trans. Database Syst. **30**(2) (2005) 364–397
17. Tao, Y., Yi, K., Sheng, C., Kalnis, P.: Quality and efficiency in high dimensional nearest neighbor search. In: SIGMOD. (2009) 563–576