Scalable Trajectory Similarity Search Based on Locations in Spatial Networks

Eleftherios Tiakas and Dimitrios Rafailidis

Department of Informatics, Aristotle University of Thessaloniki,
54124 Thessaloniki, Greece
{tiakas,draf}csd.auth.gr

Abstract. In this paper, we propose an efficient query processing algorithm that returns the trajectory results in low search time. We limit the calculation of pairwise shortest path distances between the set of query locations and the spatial nodes, by highly reducing the preprocessing requirements. Also, we introduce a spatiotemporal similarity measure, based on which the temporal-to-spatial significance of the trajectory results can be easily modified and the query locations can be spatially prioritized according to users' preferences. In our experiments with a real-world road network, we show that the proposed method has approximately ten times less preprocessing requirements than the competitive methods and reduces the search time by two orders of magnitude at least.

Keywords: Spatial/Temporal databases, Indexing methods, Spatial Networks

1 Introduction

In trajectory similarity search by locations methods [2–6], the query is considered as a small set of locations with or without an order specified, so as to find the k best connected trajectories and consequently to connect the designated locations geographically. In contrast to the conventional trajectory search, the methods of searching trajectories by locations focus on the connection provided by a trajectory to the specified query locations. The query is no longer a full trajectory or any subsequence, as it is considered in conventional trajectory similar search, but a few locations, making thus the query more flexible. This new query type has many novel applications, e.g. trip planning which demands to find trajectories that connect a few selected locations, those that are close to all the locations. For example, given a set of coordinates of locations, trajectory query by locations can help travelers in planning a trip to multiple places of interest in an unfamiliar city by providing similar routes traveled by other people for reference. In addition to location-based queries, personalized trajectory similarity search methods [3, 4, 6] have been introduced. Such methods combine the query trajectory with users' textual attributes [3, 6] or consider a user-defined significance, by weighting each location in the query [4]. Given user's textual attributes or predefined weights

for each sample point in the query trajectory, such methods retrieve the most similar trajectory, by also considering users' preferences.

Nevertheless, given the spatial network, searching trajectories by locations and personalized methods face several issues. **(I1):** The preprocessing cost is high either by building a complex spatial index [2, 5] or by including a preliminary step to precompute the all-to-all pairwise node distances in the spatial network based on the Dijkstra's algorithm [3, 4]. **(I2):** The aforementioned methods have high space requirements for preserving the spatial index or the all-to-all pairwise node distances over the trajectory similarity search, crucial for large-scale spatial networks. **(I3):** They assume that trajectories do not necessarily comply with the spatial networks constraints, by either ignoring the temporal information [5] or allowing trajectories to move in free Euclidean spaces [6].

In this paper we propose a scalable query processing algorithm for searching trajectories by locations in spatial networks. Our contribution is summarized as follows: **(C1):** The preprocessing step of our method has linear complexity, since the computation of all-to-all pairwise shortest path distances is avoided, by limiting the calculation of pairwise node distances from the small set of query locations to the nodes of the spatial graph. **(C2):** Our spatial index, an extended adjacent list representation, requires linear space, since it establishes connections directly to the original stored trajectory data on disk. The proposed extended adjacent list representation connects adjacent nodes with trajectories clusters, without the requirement of building any complex spatial index. **(C3):** Personalized searching trajectories by locations is supported by designing a spatiotemporal similarity measure which can be easily adapted to users' preferences, making the query more spatial or temporal-oriented. Additionally, weights can be set to spatially prioritize the locations.

The rest of the paper is organized as follows. Section 2 presents the fundamental concepts in trajectory similarity search in spatial networks. Section 3 details the proposed method. In Section 4 the experimental results are presented. Finally, in Section 5 the basic conclusions of our study are drawn.

2 Preliminaries

2.1 Spatial Networks & Trajectories

In this Section the fundamental concepts in trajectory similarity search in spatial networks are presented.

Spatial Networks: The spatial networks are represented as connected graphs. A spatial network $G(V, E)$ contains a set of vertices V and a set of weighted edges E. In their graph representation, the vertices and the edges indicate the road intersections and their connections, respectively. An edge $(v_i, v_j) \in E$ represents a road segment and the weight $w(v_i, v_j)$ is the travel distance from node v_i to node v_j. However, depending on the application, additional factors may be embedded into the edge weights; for example, travel time, availability, and other possible restrictions. For each pair of nodes $v_a, v_b \in V$, not necessary neighbors, their

distance is the shortest path (network distance), calculated by the accumulated edge weight of the shortest path between v_a and v_b.

Trajectories: In a large number of applications the objects are allowed to move only on predefined paths of a spatial network rather than moving freely to the Euclidean space (2D or 3D). Therefore, trajectories are constrained into the paths of the network providing motion restrictions. Let T be the set of the trajectories. Each trajectory $T_i \in T$ is represented as a directed sequence/set $T_i = \{(v_{i1}, t_{v_{i1}}), (v_{i2}, t_{v_{i2}}), ..., (v_{ir_i}, t_{v_{ir_i}})\}$ of r_i time-labeled spatial points, which are nodes $v_{i1}, v_{i2}, ..., v_{ir_i} \in V$ with their corresponding timestamps $t_{v_{i1}}, t_{v_{i2}}, ..., t_{v_{ir_i}}$. We assume that the spatial points of the trajectories lay on the nodes of spatial network G. Otherwise, if the spatial points of the trajectories lie on the edges, then they can be aligned to the closest nodes using map-matching methods. Each trajectory T_i may have its own length r_i of spatial points, which is called description length. Therefore, we assume that trajectories are of arbitrary description lengths in T, which means that for two different trajectories T_i, T_j, with $i \neq j$, it may hold that $r_i \neq r_j$. Finally, the multiset[1] of the spatial points from all trajectories is denoted as R, and the multiset of all trajectory edges as RE. Both multisets R, RE represent the trajectory data, where: $|R| = \sum_{i=1}^{|T|} r_i$ and $|RE| = \sum_{i=1}^{|T|}(r_i - 1) = |R| - |T|$.

2.2 Problem Formulation

Let G be a spatial network and T a dataset of trajectories. Let Q be a set of query locations $q_1, q_2, ..., q_m$ which are the spatial points (nodes of the graph G), that the resulted trajectories have to pass close. Let $qt_2, qt_3, ..., qt_m$ be the corresponding inter-arrival times which are $m-1$ tolerance time intervals, acceptable by users for travelling between the query locations, where $qt_i = \infty$ denotes the lack of time restriction for location q_i. Let $w_1 \dots w_m$ be the users' predefined weights, expressing the personal preferences to the m query locations, where $0 < w_j < 1, j = 1, ..., m$ and $\sum_{j=1}^{m} w_j = 1$. Given a similarity function $sim(Q, T_i)$ between the set Q of query locations and a trajectory $T_i \in T$, the goal is: *to find the k most similar trajectories in T with the highest similarity score to Q.*

3 Proposed Method

3.1 Spatial Index - Preprocessing

In the preprocessing step, the spatial network $G(V, E)$ is stored using an adjacency list representation. The main idea is to extend the adjacency list by using trajectory clusters. Given a node v_i and the adjacency list representation of p adjacent nodes, then for each edge $(v_i, v_j) \in E$, with $j = 1, \dots, p$, the respective cluster C_{ij} is generated. Each cluster C_{ij} consists of the trajectory Id's of the

[1] Multiset is a generalization of the notion of set in which members are allowed to appear more than once.

respective trajectories $T_h \in T$ that pass through edge (v_i, v_j). In case that an edge $(v_i, v_j) \in E$ is not included in any trajectory of the T set, the respective cluster C_{ij} remains empty. Each cluster C_{ij}, which is generated by the edge (v_i, v_j), is connected with the adjacent node v_j of node v_i. Figure 1 depicts the extended adjacency list index of node v_i, with p trajectory clusters, connected with the respective p adjacent nodes. Therefore, given $|V|$ nodes in the G graph the main index consists of $|V|$ extended adjacent lists with the adjacent nodes being connected with the respective trajectories clusters.

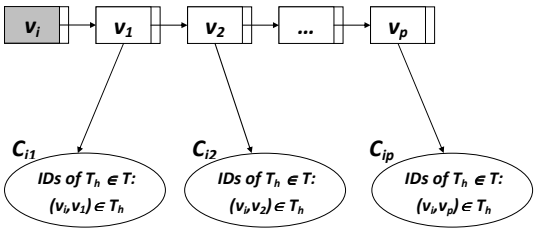

Fig. 1. The extended adjacency list index of node v_i, with p adjacent nodes and the respective p trajectory clusters.

The set of the $|V|$ extended adjacent lists is generated in the preprocessing step and consists the main index of our method. The implementation has low complexity, since the set is built during the load of the network and the trajectory data. Algorithm 1 depicts the preprocessing procedure. Since the trajectory clusters are implemented as dynamic lists, an one-pass linear scan of the trajectories data is required. During the scan, the trajectory data are retrieved through a hash function $hash(\cdot)$ which takes as argument the trajectory Id and returns the address (pointer) of the stored trajectory on disk. Then, each edge of the trajectories is traversed and the corresponding trajectory Ids that contain the specific edge are dynamically inserted into the respective cluster C_{ij} of the index. In our implementation, we used the hash function $hash(\cdot)$ of modulo: $Id \ mod \ |T|$. Alternatively, many other hash functions could be used, without affecting the performance of our method.

Algorithm 1: **Preprocessing**
Input: spatial network G, set of trajectories T, hash function $hash(\cdot)$
Output: main indexer

01. **for each** trajectory $T_i \in T$
02. allocate data of T_i through $hash(T_i.Id)$
03. **for each** edge $(v_i, v_j) \in T_i$
04. insert $T_i.Id$ in cluster C_{ij}
05. **end-for**
06. **end-for**

The time complexity of the preprocessing algorithm is $O(|V| + |E| + |RE|)$, by performing a linear scan of the trajectory data. The space complexity of the index is $O(|V|+|E|+|RE|)$ for storing both the network data and the trajectory clusters. The proposed indexing scheme has many advantages. Linear space is required to store both network data and trajectory Ids. The proposed indexing scheme requires linear building time. Additionally, the trajectory clustering approach generates smaller clusters than the recent work of [4] does. The main difference is that we propose an edge-based clustering which distributes the trajectories to many small clusters, whereas Shang et al. in [4] propose a node-based clustering which produces significantly larger clusters. For each trajectory Id a hashing key can instantly be retrieved, by directly connecting to the trajectory data which are stored on disk.

3.2 Proposed Similarity/Distance Measures

In this Section, we present the proposed distance functions $Ds(\cdot)$ and $Dt(\cdot)$ in the spatial and temporal domains respectively, along with the final spatiotemporal similarity measure $sim(\cdot)$. The proposed measures express the similarity between a trajectory $T_i \in T$ and the user-defined query locations in set Q. Therefore, the proposed measures are functions in the $|Q| \times |T|$ space, instead of the $|T| \times |T|$ space, by significantly speeding up computations.

Spatial Distance Function $Ds(\cdot)$ The spatial similarity expresses how close is a trajectory to the query locations according to the restrictions of the spatial network. Due to these restrictions we consider the main spatial distance function $d(\cdot)$ between a trajectory node and a query location, where $d(\cdot)$ is calculated based on the network/shortest-path distance, with the rest of possible network restrictions being already embedded into the edge weights of the network.

Given a single query location $q_j \in Q$ and a trajectory $T_i \in T$ which contains the nodes $v_1, v_2, ..., v_r$, the spatial distance of the trajectory T_i from a query location q_j is defined as

$$ds(q_j, T_i) = \min_{h=1,...,r} d(q_j, v_h) \tag{1}$$

Therefore, the spatial distance from a query location q_j is the minimum spatial distance from the nodes of the trajectory T_i to the query location. Let $vmin_j$ be the corresponding node of the trajectory T_i with the minimum distance from the query point q_j. Then, the spatial proximity between a query location and a trajectory is equal to $ds(q_j, T_i) = d(q_j, vmin_j)$. In trajectory similarity search by locations, a specific node of the trajectory lies close to a query location. Therefore, for each query location $q_j, j = 1, ..., m$, the corresponding nodes $vmin_j$ are detected independently[2].

[2] Similar selection strategy of measure ds is followed by [2, 5], however, the measure is termed as *matched pairs* based on Euclidean distances, ignoring the spatial constraints.

A desired characteristic of the spatial similarity of a trajectory to the query locations is to have at least j nodes as close to q_j as possible (pairwise similarities/distances). In order to consider that, we must aggregate the spatial distances of the trajectory from each query location q_j. Therefore, we define the spatial distance of the trajectory T_i from the whole query set Q as follows:

$$Ds(Q, T_i) = \frac{1}{m} \sum_{j=1}^{m} ds(q_j, T_i) \tag{2}$$

The spatial distance of a trajectory T_i to the set Q of query locations is computed as the average distance function $ds(\cdot)$ from each query location. Additionally, in case that a user-specific spatial-priority in the spatial locations q_j must be included, the following weighted average distance is used:

$$Ds(Q, T_i) = w_1 \cdot ds(q_1, T_i) + ... + w_m \cdot ds(q_m, T_i) \tag{3}$$

where $w_j, j = 1, ..., m$, are users' predefined weights, with $0 < w_j < 1, j = 1, ..., m$ and $\sum_{j=1}^{m} w_j = 1$. A value of w_j close to 1 (or 0) means that the distance from the query location q_j contributes more (or less) to the total distance, affecting the spatial closeness or farness of q_j to T_i.

Temporal Distance Function $Dt(\cdot)$ An important characteristic of our proposed methodology is that the time information of all the resulted k trajectories are not necessarily required, which means that the time-stamps of all the trajectory nodes are not essential to the calculations of the temporal distance function. This happens because the corresponding inter-arrival times $qt_2, qt_3, ..., qt_m$, tolerance time intervals, which are acceptable by users for traveling between the locations, exclusively define the time restrictions in the temporal domain. In this Section, we present the temporal distance function $Dt(Q, T_i)$ between the set Q of query locations and a trajectory T_i.

In the calculations of the $Ds(\cdot)$ spatial distance function, the corresponding nodes $vmin_j, j = 1, ..., m$, that express the spatial proximity of T_i from Q are retrieved, for each query location q_j. Let $t_{vmin_j}, j = 1, ..., m$, be the corresponding times instances of the nodes $vmin_j$ of the trajectory T_i. We calculate the corresponding inter-arrival times $dt_2, dt_3, ..., dt_m$ between the resulted nodes $vmin_j$, which can be computed instantly by summing the inter-arrival times of the intermediate nodes, through the trajectory path. For each location $j \in 2, 3, ..., m$, there are the following cases for the inter-arrival times qt_j and dt_j, **Case 1:** $(dt_j = qt_j)$: The query tolerance time interval qt_j is equal to the actual time interval dt_j from the trajectory. Since the time intervals are identical the temporal distance must be zero. **Case 2:** $(dt_j > qt_j)$: The query tolerance time interval qt_j is less than the actual time interval dt_j from the trajectory. In this case a temporal distance between the trajectory and the query locations must be considered, since the trajectory requires more time for the travel. The temporal distance is: $|qt_j - dt_j|$. **Case 3:** $(dt_j < qt_j)$: The query tolerance time interval qt_j is greater than the actual time interval dt_j from the trajectory. Therefore,

the trajectory requires less time than the query restrictions and we must not consider any temporal distance between the trajectory and the query locations, i.e the temporal distance must be zero. In order to have a unified formula we set $dt_j = qt_j$ and the temporal distance is preserved zero, as in the case of $(dt_j = qt_j)$. Considering the aforementioned three cases, the temporal distance function $Dt(Q, T_i)$ between the trajectory T_i and the query set Q is calculated as follows:

$$Dt(Q, T_i) = \frac{1}{m-1} \sum_{j=2}^{m} \frac{|qt_j - dt_j|}{\max\{qt_j, dt_j\}} \tag{4}$$

Spatiotemporal Similarity Measure $sim(\cdot)$ The proposed spatiotemporal similarity measure $sim(\cdot)$ is a linear weighted aggregation of the spatial $Ds(\cdot)$ and temporal $D_t(\cdot)$ distance functions,

$$sim(Q, T) = 1 - dist(Q, T)$$

$$dist(Q, T) = a \cdot Ds(Q, T) + (1 - a) \cdot Dt(Q, T) \tag{5}$$

where $a \in [0, 1]$ is a weighting parameter, which expresses the temporal to spatial preference. Values of a close to 0 and 1 denote user's preference to temporal and spatial domains, respectively. The final distance and similarity measures ($dist(\cdot)$ and $sim(\cdot)$) are real numbers in the interval $[0, 1]$. Also, parameter a of Eq. (5) can be used to define the importance of temporal to spatial similarity. In case of $a = 1$ the spatial similarity solely contributes to the final similarity score of $sim(\cdot)$ or distance of $dist(\cdot)$. On the contrary, in case of $a = 0$ the temporal similarity exclusively contributes to the final score. Therefore, a user can select the desired value of the a parameter, depending on the application.

3.3 Query Processing By Locations

A preliminary step to the query processing algorithm is required, where the computation of the pairwise shortest path distances between the query locations and the nodes of the graph is performed. In the preliminary step, the calculations of the shortest-path distances are limited to the $m \times |V|$ pairwise distances between the user-selected query locations $q_1, q_2, ..., q_m$ and the nodes of the spatial network G. This comes in contrast to previous works [3, 4] where the computation of all-to-all pairwise node distances ($|V| \times |V|$) is performed. The distances are calculated based on the Dijkstra algorithm in $O(m * (\log |V| + |E|))$ time, equipped with a Fibonacci Heap structure. The shortest-path distances are stored in memory, requiring $O(m * |V|)$ space.

The main strategy of the proposed algorithm consists of the following steps: *(a) from each query location, perform a Dijkstra expansion step incrementally, following a round-robin strategy; (b) collect the trajectory Ids that are included in the trajectory clusters of the visited edges; (c) compute the spatiotemporal similarities based on Eq. (5) and generate the top-k results.* The algorithm of the query processing by locations is presented in Algorithm 2.

The five main steps of the proposed algorithm are the following, **(S1):** From each query location, in a round-robin fashion (initially $v_j = q_j$), each neighbor node v_i of v_j is retrieved in the Dijkstra expansion step (lines 5-16). Then, the candidate trajectories T_h are collected from the corresponding edge cluster C_{ij} of the extended adjacency list index (line 17). **(S2):** The spatiotemporal distances $dist$ between the collected candidate trajectories T_h and the query location set Q are calculated (Eq.(5)). In order to avoid recalculations in any step of the algorithm, a bit-set B with $|T|$ bits in memory is used where the corresponding bit of each calculated trajectory distance is enabled on-the-fly (lines 18-23). Therefore, during the query processing, the distances are calculated only once for each trajectory. The currently top-k calculated trajectory distances and their corresponding trajectory Ids are preserved and updated in a priority heap H (ordered by $dist$) on-the-fly (line 22). Heap H has a limited size of k items, since only the most similar k results are required. **(S3):** Then, a threshold L is updated (initially set to 0), according to the average sum of the network distances between the query locations and the set of $vmin_j$ nodes: $L = \frac{a}{m} \sum_{j=1}^{m} d(q_j, vmin_j)$, where $vmin_j$ is the closest node to query location q_j in the current Dijkstra expansion level, i.e. $vmin_j$ has the shortest path distance to q_j among all the detected nodes in the current round from q_j. The threshold L is a lower bound of the final distance function $dist$, and it is used for generating the results. In each round, L is increased, (when the expansion level is changed), by comparing the current $Lcurr$ value with the previously calculated one. In particular, if the currently computed $Lcurr$ value is greater than the previous L value of the last round, then the $Lcurr$ value of the current round is updated accordingly (lines 27, 28). Since the temporal distances Dt are aggregated with the spatial distances Ds in the final distance function $dist(\cdot)$, L is a lower bound for both spatial and spatiotemporal distances. Moreover, in case that w_j weights are used (Eq. (3)), then threshold L is calculated as: $L = a \cdot \sum_{j=1}^{m} w_j \cdot d(q_j, vmin_j)$. **(S4):** After the end of each round, the trajectories in the current top-k list in heap H are examined based on condition that they have a distance $dist$ lower than L. If the condition is satisfied for a subset of trajectories in H, then these trajectories are instantly added to the top results list (lines 29-31). The trajectory insertion proceeds progressively until L reaches a value greater than the distance of the k-th element in H or in the extreme case that the spatiotemporal distances of all trajectories in T have been calculated (stopping condition, line 32). **(S5):** In case that not all top-k results have been retrieved, the algorithm proceeds to the next expansion round, where the algorithm repeats the loop in lines 5-33.

The total time complexity of the proposed query processing algorithm is $O(m * (|V| \log |V| + |E|) + |RE|)$. An $m * (|V| \log |V| + |E|)$ cost is required for the Dijkstra expansion from the m query locations. The candidate trajectories are collected from the clusters C_{ij} of the extended adjacency list index; nevertheless, the total amount of trajectory Ids that exist into the clusters is $|RE|$, since RE contains all the trajectory edges of the dataset, i.e. $\sum_{i,j} |C_{ij}| = |RE|$. The existence of the bit-set B avoids recalculations when trajectories are dis-

Algorithm 2: **Progressive Query Processing Algorithm**
Input: the spatial network G,
the set of trajectories T,
the set of the query locations Q, the number of results k
Output: top-k trajectories (progressively)

01. $L = 0, Lcurr = 0, top = 1$
02. initialize $vQ_j = q_j, \forall j = 1, ..., m$
03. initialize $distQ_j[vQ_j] = 0, \forall j = 1, ..., m$
04. $HQ_j.insert(vQ_j, 0), \forall j = 1, ..., m$
05. **while** $HQ_j.size > 0, \forall j = 1, ..., m$
06. **for** $j = 1$ to m
07. $vQ_j = HQ_j.extractMin()$, $vmin_j = vQ_j$
08. **for each** neighbor uQ_j of vQ_j in the adjacency list
09. **if** $distQ_j[uQ_j] > distQ_j[vQ_j] + w(vQ_j, uQ_j)$ **then**
10. $distQ_j[uQ_j] = distQ_j[vQ_j] + w(vQ_j, uQ_j)$
11a. **if** $(inqueQ_j[uQ_j] <> 0)$ **then**
11b. $HQ_j.decreaseKey(uQ_j, distQ_j[uQ_j]$
12. **end-if**
13. **if** $(inqueQ_j[uQ_j] = 0)$ **then**
14. $HQ_j.insert(uQ_j, distQ_j[uQ_j])$
15. $inqueQ_j[uQ_j] = 1$
16. **end-if**
17a. **for each** Traj.-Id h (trajectory T_h) in cluster
17b. $C_{(vQ_j, uQ_j)}$ from edge (vQ_j, uQ_j)
18. **if** $B[h] = false$ **then**
19. retrieve data of trajectory T_h through $hash(T_h.Id)$
20. compute $dst = dist(Q, T_h)$
21. $B[h] = true$
22. $H.insert(T_h.ID, dst)$
23. **end-if**
24. **end-for**
25. **end-for**
26. **end-for**
27. $Lcurr = \frac{a}{m} \sum_{j=1}^{m} d(q_j, vmin_j)$
28. **if** $Lcurr > L$ **then** $L = Lcurr$
29. **for** $i = top$ to k
30. **if** $H[i].dst < L$ **then** top++, return trajectory $H[i].ID$
31. **end-for**
32. **if** $L > H[k].dst$ or $B.count = |T|$**then** stop
33. **end-while**

covered from different locations explaining the additional $|RE|$ cost in the total complexity.

4 Experimental Evaluation

4.1 Settings

In our experiments we used the North America Road Network[3] **(NA)**, which contains 175,813 nodes and 179,179 edges. We compared the proposed method against the following personalized methods for searching trajectories by locations: (a) "Two-phase PTM", (b) "PTM without heuristic" and (c) "Balanced", on the common evaluation data set of the North America Road Network, recently presented in [4]. "Two-phase PTM" is the personalized matching method

[3] The Digital Chart of the World Server at http://www.maproom.psu.edu/dcw/

of Shang et al. [4]. "PTM without heuristic" and "Balanced" methods are two naive approaches, also used in [4] for comparison. The following evaluation metrics were used: (a) the required CPU-time for the main query processing algorithm and (b) the number of visited trajectories during the trajectory similarity search. Following the evaluation protocol of [4], the number of visited trajectories represents the required accesses to trajectory data that are stored on disk. The number of visited trajectories reflects to the real disk I/O cost at a certain degree, since the systems may have hidden buffers and cash-memories, making thus difficult to measure the real I/O cost more accurately.

To generate the trajectories in the North America Road Network we used the Brinkhoff's generator [1], which defines the velocity in the trajectory parts of each moving object/vehicle and categorizes the vehicles in classes according to these velocities. For each generated trajectory data set, the average vehicle velocity is computed and then, for each query the tolerance time intervals qt_i are set equal to the fraction of the network distances between the query locations q_i and the average velocity. Moreover, since the query locations are randomly selected, we set equal w_i weights to $1/m$. Following the personalized trajectory matching method of "Two-Phase PTM", the weight w of each sample point in a query trajectory was randomly generated (integers in [1, 5]), to evaluate the performance of the proposed method. Nevertheless, in our experiments we observed that the performance of the proposed method is preserved either considering equal or randomly generated and different weights in the query locations. Therefore, in the experimental results equal weights are considered.

In order to perform fair comparison against the personalized trajectory matching methods of [4], 'Two-phase PTM", "PTM without heuristic" and "Balanced", on the common evaluation data set of North America Road Network, we regenerated the trajectories in the data set by setting the same parameters to the Brinkhoff's generator with the work of [4]. We varied the description lengths r between 20-100 spatial points, which were mapped to the nodes of the spatial graph. For the number of selected trajectories $|T|$ we performed the same variation of: $10K$, $15K$, $20K$, $25K$, $30K$. We set $k = 1$, similar to the examined algorithms in [4], where the case of the most similar trajectory (top-1) is considered. Additionally, we set the default value of $m = 60$. Finally we set $a = 0.5$, equal to the default λ value for the spatiotemporal similarity measure of [4], which also expresses the temporal-to-spatial significance. The resulted values of the common evaluation metrics of CPU-Time and number of visited trajectories, are averaged by 50 queries with random selected query locations.

All experiments have been conducted on a machine with Quad Core 3GHz CPU, 2TB SATA3 Hard Disk, running Windows 7 64bit.

4.2 Results

In Table 1, the preprocessing requirements of the examined methods are depicted. The proposed method has lower preprocessing requirements than the competitive methods. This happens because the preprocessing time of the proposed method is to build the spatial index (18.687 sec) and to compute the

$m \times |V|$ shortest path distances from the m =60 query locations (5.327 sec), whereas the competitive methods require to compute the $|V| \times |V|$ all-to-all shortest path distances (380.8 sec), also mentioned in [4]. Additionally, the proposed method requires 80.5 MB to preserve the precalculated distances in the memory, added by 20.3 MB for the spatial index space. The competitive methods require a memory space of 630.6 MB to maintain the all-to-all pairwise distances, added by the required space during the query processing for (a) preserving the extra labels in the trajectories to determine if they are touched in the spatial or the temporal domain (partial or full match) from each query location; and (b) using the priorities to select the candidates and perform the appropriate scheduling, resulting in an additional space of 886.3, 365.7 and 365.7 MB for the "Balanced", "PTM without heuristic" and 'Two-phase PTM" methods, respectively. Therefore, the proposed method has approximately 10 times less preprocessing requirements than the competitive methods.

Table 1. Preprocessing cost.

	Preproc. Time (sec)	**Mem. Space (MB)**
Proposed method	24.014	100.8
Balanced	380.8	1516.9
PTM without heuristic	380.8	996.3
Two-phase PTM	380.8	996.3

In the experimental results of Figure 2, the proposed method outperforms all the competitive methods in terms of CPU-Time and number of visiting trajectories. In the query processing algorithm of the most competitive method of "Two-phase PTM", for each query location the respective most similar trajectory is visited by searching in the spatial and temporal domains separately. Therefore, by incrementally retrieving the candidate most similar trajectories for all query locations the algorithm terminates when the Lower Bound (LB) exceeds the Upper Bound (UB) [4]. In doing so, "Two-phase PTM" may perform multiple visits for the same trajectories, by significantly increasing the CPU-time and the number of accesses to the trajectories. Moreover, our proposed method performs an edge-based clustering which distributes the trajectories to many small clusters, whereas the "Two-phase PTM" method performs a node-based clustering which produces larger clusters. The different searching strategies can explain the high performance of the proposed method, which achieves a speed up factor of 100 against the most completive method of "Two-phase PTM".

5 Conclusions

In this paper, we presented a scalable query processing by locations algorithm, by also taking into account the spatial importance of the query locations based on the users' preferences. In our experiments, we showed that the proposed method

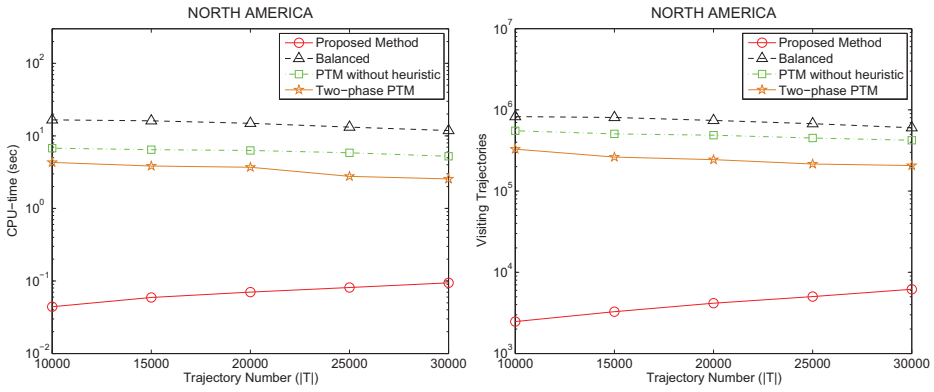

Fig. 2. Methods comparison in terms of CPU-time (sec) and number of visiting trajectories.

significantly outperformed competitive personalized trajectory similarity search methods, where the proposed method has approximately ten times less preprocessing requirements and reduces the query runtime by two orders of magnitude at least. An interesting research direction is to extend the proposed method to an approximate algorithm based on probabilistic bounds, in order to perform similarity search in spatial networks for uncertain trajectories [7].

References

1. T. Brinkhoff, "A framework for generating network-based moving objects.", *Geoinformatica*, Vol.6, No.2, pp.153-180, 2002.
2. Z. Chen, H. T. Shen, X. Zhou, Y. Zheng, and X. Xie, "Searching Trajectories by Locations: An Efficiency Study.", *In Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pp.255-266, 2010.
3. S. Shang, R. Ding, B. Yuan, K. Xie, K. Zheng, and P. Kalnis, "User Oriented Trajectory Search for Trip Recommendation", *In Proc. of the 15th Int. Conf. on Extending Database Technology*, pp.156-167, 2012.
4. S. Shang, R. Ding, K. Zheng, C. S. Jensen, P. Kalnis, and X. Zhou, "Personalized Trajectory Matching in Spatial Networks", *The VLDB Journal*, Vol.23, No.3, pp.449-468, 2014.
5. L. Tang, Y. Zheng, X. Xie, J. Yuan, X. Yu, and J. Han, "Retrieving k-Nearest Neighboring Trajectories by a Set of Point Locations", *In Proc. of the 12th Int. Conf. on Advances in Spatial and Temporal Databases*, pp.223-241, 2011.
6. H. Wang, and K. Liu, "User Oriented Trajectory Similarity Search", *In Proc. of the ACM SIGKDD Int. Workshop on Urban Computing*, pp.103-110, 2012.
7. K. Zheng, G. Trajcevski, X. Zhou, and P. Scheuermann, "Probabilistic range queries for uncertain trajectories on road networks". *In Proc. of the 14th Int. Conf. on Extending Database Technology,*, pp. 283-294, 2011.