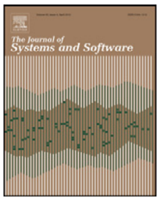
The Journal of Systems and Software 86 (2013) 1566–1579

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss



Map-matched trajectory compression

Georgios Kellaris[a,*], Nikos Pelekis[b], Yannis Theodoridis[c]

[a] Department of Computer Science and Engineering, HKUST, Hong Kong
[b] Department of Statistics and Insurance Science, University of Piraeus, Greece
[c] Department of Informatics, University of Piraeus, Greece

ARTICLE INFO

Article history:
Received 13 September 2011
Received in revised form 4 October 2012
Accepted 26 January 2013
Available online 10 February 2013

Keywords:
Compression
Network trajectory
Map-matching

ABSTRACT

The wide usage of location aware devices, such as GPS-enabled cellphones or PDAs, generates vast volumes of spatiotemporal streams of location data raising management challenges, such as efficient storage and querying. Therefore, compression techniques are inevitable also in the field of moving object databases. Related work is relatively limited and mainly driven by line simplification and data sequence compression techniques. Moreover, due to the (unavoidable) erroneous measurements from GPS devices, the problem of matching the location recordings with the underlying traffic network has recently gained the attention of the research community. So far, the proposed compression techniques have not been designed for network constrained moving objects, while on the other hand, existing map matching algorithms do not take compression aspects into consideration. In this paper, we propose solutions tackling the combined, map matched trajectory compression problem, the efficiency of which is demonstrated through an extensive experimental evaluation on offline and online trajectory data using synthetic and real trajectory datasets.

1. Introduction

A moving object database (MOD) is a collection of objects whose location changes over time. MOD management has emerged due to the integration of positioning technologies into wireless devices that appears nowadays, and has posed great challenges to the spatial database community. The trajectory of a moving object can be described as a set of tuples of the form $\langle x, y, t\rangle$, where $\langle x, y\rangle$ is the geographic location of the object at time t. The preservation of vast volumes of trajectories for future reference raises compression aspects, i.e., the *trajectory compression* problem (hereafter, called *TC*). Locations are often recorded by GPS receivers; raw data points received by a GPS receiver embed an error of some meters. Thus, there arises the problem of matching this information onto a road network, also known as the *map-matching problem* (hereafter, called *MM*). The proposed algorithms for the solution of each problem can be divided into two categories: *offline*, where the input data are known a priori, and *online*, where the data are received in real time.

In this work, we study the combined problem of compressing a moving object's trajectory keeping it at the same time matched on the underlying road network; this is the so-called *map matched trajectory compression problem* (hereafter called *MMTC*). To illustrate it, Fig. 1 presents the original trajectory (gray line) as recorded e.g. by a GPS device, its map-matched counterpart (black solid line) and a possible solution to the MMTC problem, i.e., a map-matched trajectory with fewer points than the original map-matched (dotted line).

GPS devices usually offer the option to retrieve past trips. However, due to storage space limitations these devices cannot hold large amounts of information. The use of compression algorithms, e.g. zip, could be a partial solution to this problem, but the low processing capability of these devices would make the decompression of these data a frustrating procedure for the user. Thus, there arises the need of an algorithm that could compress the data while preserving its original form. This procedure could be applied to the user's current trip (online solution) or to her previous trips when more storage space is needed for new data (offline solution).

To the best of our knowledge, this is the first work that tackles the MMTC problem sketched above. The contributions of the paper are the following:

- Taking into consideration existing offline and online TC and MM algorithms, we propose two working solutions for the combined MMTC problem as a first attack to the problem.
- Formulating MMTC as a cost-optimization problem, we present a theoretical analysis for finding the optimal solution accompanied, due to its high computational cost, by an approximate offline algorithm.

* Corresponding author. Tel.: +852 95767760.
E-mail address: gkellaris@ust.hk (G. Kellaris).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.01.071

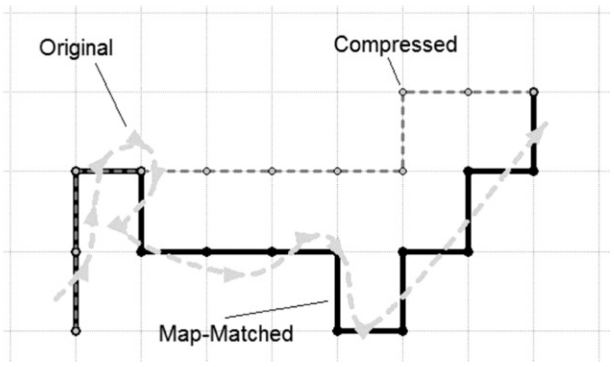


Fig. 1. MMTC problem illustration.

- In the same line, we present the corresponding approximate online algorithm.
- We perform an extensive experimental study over synthetic and real trajectory datasets demonstrating the effectiveness and the efficiency of the proposed solutions.

The rest of the paper is organized as follows. Section 2 reviews related work regarding the TC and MM problems, also discussing network-based trajectory similarity search, which is relevant to our work. Section 3 formalizes the MMTC problem in two variations: offline and online. Sections 4 and 5 present a solution based on existing TC and MM techniques followed by our novel MMTC approach, respectively. Section 6 describes an evaluation methodology of the proposed techniques, and presents the experimental results of our study. Section 7 concludes this work and suggests motivating future directions.

2. Related work

In the literature, two types of compression techniques for trajectories have been proposed, i.e., online and offline algorithms. Online algorithms (Meratnia and de By, 2004; Potamias et al., 2006) compress the trajectory data as new points are received, whereas offline algorithms, driven by line simplification techniques (Douglas and Peucker, 1973), assume the a priori knowledge of the complete trajectory (Meratnia and de By, 2004; Xu and Lee, 2007). On the other hand, for the map-matching problem, online (Bernstein and Kornhauser, 1996; Greenfeld, 2002; Quddus et al., 2003) as well as offline algorithms (Brakatsoulas et al., 2005; Yin and Wolfson, 2004) have been proposed.

In the following subsections, we review existing proposals in the domains related with the current work: methods proposed separately for TC (Section 2.1) and MM (Section 2.2) as well as for the combined MMTC (Section 2.3). We also present approaches for trajectory similarity search over the road network (Section 2.4).

2.1. Trajectory compression

Trajectory compression can be considered as an equivalent to line simplification (ignoring time dimension). As such, most TC techniques are based on the well-known Douglas–Peucker line simplification algorithm, which follows a divide-and-conquer approach to keep only the most important points of a polyline, i.e., the ones that lie far from the line that would result if these points were removed (Douglas and Peucker, 1973).

The technique proposed by Meratnia and de By (2004) is considered to be the most popular offline TC algorithm. It uses Douglas–Peucker and, moreover, takes the parameter of time into account. In particular, it replaces the Euclidean distance used in Douglas–Peucker by a time-aware one, called synchronous Euclidean distance (SED).

Recently, Xu and Lee (2007) presented a method called DTTC (delay-tolerant trajectory compression) for compressing moving object data in wireless sensor networks. First, they propose a cluster infrastructure for the in-network sensors. The head of each cluster compresses the object's trajectory before reporting it. In order to achieve that, the cluster heads divide the trajectory into segments and compress each segment with a simple compression function. In order to minimize the number of segments, two compression techniques are introduced: a divide-and-conquer as well as a step-wise technique. Any compression technique can then be used upon the segments of the trajectory.

Regarding the online case, Meratnia and de By (2004) propose a variation of their offline algorithm called Opening Window (OW). The algorithm works as follows. Initially, it defines a line segment between the first and the third data point. If the SED from each internal point to the segment is not greater than a given threshold, the algorithm moves the end point of the segment one point up in the data series. When the threshold is exceeded there are two options: either the data point, which causes the threshold excess, or its precedent is defined as the end point of the current segment and the start point of a new one. As long as new points arrive, the method continues as described.

Alternatively, Potamias et al. (2006) propose two algorithms, called Thresholds and STTrace, respectively, for online trajectory data compression. The algorithms use the coordinates, speed, and orientation of the current point in order to calculate a safe area where the next point might be located. If the next incoming point lies in the calculated safe area, it can be ignored. There are two options for the definition of the safe area. It is either calculated by using the data of the last point, whether the point is previously ignored or not, or by using the data of the last chosen point. In order to achieve better results, a combination of the two algorithms is also proposed. Both areas are calculated, but only their intersection is defined as the safe area.

The time complexity of the algorithms proposed in Meratnia and de By (2004) and Xu and Lee (2007) is $O(n^2)$, while the one proposed in Potamias et al. (2006) has time complexity of $O(n)$, where n is the number of points composing the trajectory.

2.2. Map-matching

Several solutions to the MM problem have been proposed. The most recent ones are those by Yin and Wolfson (2004) and Brakatsoulas et al. (2005) for the offline map-matching problem, and those by Bernstein and Kornhauser (1996), Greenfeld (2002), and Quddus et al. (2003) for the online problem.

Regarding the offline MM problem, Yin and Wolfson (2004) propose using Dijkstra's shortest path algorithm (Dijkstra, 1959) to determine the distance between a trajectory and a sequence of arcs on a map. Then, they define the route with the smallest distance from the initial trajectory as the map-matched trajectory.

On the other hand, Brakatsoulas et al. (2005) propose the following methodology: given that point P_{i-1} has already been matched to an edge e, they consider the adjacent edges of e as the candidate edges to be matched to P_i. The candidate edges are evaluated, as illustrated in Fig. 2. In this example, P_{i-1} is matched to edge c_3, hence c_1, c_2 and c_3, are the candidate edges for point P_i.

If the projection of the current point on the candidate edges does not lie between the end points of any of these edges, the algorithm does not proceed to the next point. Instead, the nearest edge of the candidates is set as part of the trajectory and then the next set of candidate edges for the same point is evaluated. In order to improve the quality of the result, a "look ahead" policy is proposed. That is, the total score of each candidate edge is calculated by adding the

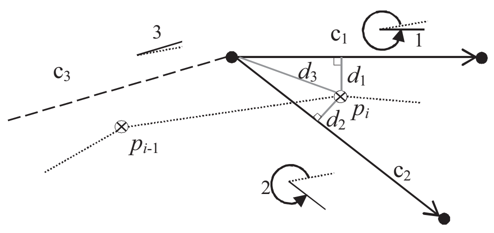


Fig. 2. Map-matching algorithm (Brakatsoulas et al., 2005).

scores of a fixed number of edges which are ahead of the current position.

The time complexity of the algorithm is $O(na^{l+1})$, where n is the number of the trajectory points, a is the number of the adjacent network road edges to each point, and l is the number of edges of the "look-ahead" policy (Brakatsoulas et al., 2005). Since a and l are in fact constants, the time complexity can be rewritten to $O(n)$.

For online data, Bernstein and Kornhauser (1996) describe several algorithms by using geometric information to induce point-to-point and point-to-curve matching. Greenfeld (2002) proposes a method based on topological analysis using the coordinates of the observed user position; however, the method assumes no knowledge of the expected traveling route and does not use any speed information. Alternatively, Quddus et al. (2003) propose an algorithm that uses GPS information such as position, velocity, and time, in order to avoid the switching of mapped locations between unconnected road links.

2.3. The combined map-matched trajectory compression

The most related approach to the problem that motivates our work is that of Cao and Wolfson (2005), who explore the combination of the map-matching and the storage-space problem. They propose a solution that uses the a priori knowledge of the road network. The outcome is a (so-called) "nonmaterialized" trajectory that needs processing in order to obtain the original trajectory in the usual format – just like the zip compression technique works. More specifically, the idea of the nonmaterialized trajectory separates the spatial component represented by the road network (i.e., the street map) from the temporal component that is specific to each trajectory. Then, the location of the moving object is inferred at any point in time by assuming constant speed motion and by using the coordinates of the streets (given by the map) traversed by the moving object. Obviously, the constant speed assumption in real application scenarios may result in significant changes to the original data at trajectory reconstruction time.

It is clear that the problem we tackle is different (recalling Fig. 1). In our setting, we require the result of our method to be trajectories over the network in order to be able to directly perform database querying. In other words, our goal is to reduce the size of the MOD that maintains the trajectory data without altering its infrastructure, but only deleting/updating trajectory tuples. Without arguing to be a lossless compression technique, it preserves the structure of the original data and, at the same time, it provides data ready to be used without any trajectory reconstruction required in order to perform querying, analysis, or mining.

2.4. Network-based trajectory similarity

When data compression is studied, it is obvious that the quality of the procedure should be measured, among others, through the similarity between the original and the compressed data. In our case, we define quality as the similarity between the map-matched counterpart of the original trajectory and the compressed (network-constrained) trajectory, which are involved in the MMTC procedure.

Traditional methods for trajectory similarity search (e.g. Chen et al., 2005; Frentzos et al., 2007; Vlachos et al., 2002) cannot be applied for network-constrained movement. On the other hand, Tiakas et al. (2009) have recently presented a method for trajectory similarity search under network constraints. In detail, let $c(u_i, u_j)$ be the cost to travel from node u_i to node u_j (this cost expresses any meaningful cost measure, such as travel distance or average time required to travel a path on the network). The network distance $d(u_i, u_j)$ between these two nodes is defined as (Tiakas et al., 2009):

$$d(u_i, u_j) = \begin{cases} 0, & \text{if } u_i = u_j \\ \dfrac{c(u_i, u_j) + c(u_j, u_i)}{2D_G}, & \text{otherwise} \end{cases} \tag{1}$$

where $D_G = \max\{c(u_i, u_j), \forall u_i, u_j \in V(G)\}$ is the diameter of the network graph G.

Then, the *network distance* between two trajectories T_a and T_b is defined as:

$$D_{net}(T_a, T_b) = \frac{1}{m}\sum_{i=1}^{m}(d(u_{ai}, u_{bi})) \tag{2}$$

where u_{ai} and u_{bi} are the ith points of trajectories T_a and T_b, respectively.

Regarding the temporal dimension, the *time distance* between trajectories T_a and T_b is defined as:

$$D_{time}(T_a, T_b) = \frac{1}{m-1} \cdot \sum_{i=1}^{m-1} \frac{|(T_a[i+1]\cdot t - T_a[i]\cdot t) - (T_b[i+1]\cdot t - T_b[i]\cdot t)|}{\max\{(T_a[i+1]\cdot t - T_a[i]\cdot t), (T_b[i+1]\cdot t - T_b[i]\cdot t)\}} \tag{3}$$

where $T[i]{\cdot}t$ is defined as the temporal data value of the ith point of trajectory T.

The total dissimilarity between trajectories T_a and T_b is then expressed as the weighted average of the network and time distances:

$$D_{total}(T_a, T_b) = W_{net} \cdot D_{net}(T_a, T_b) + W_{time} \cdot D_{time}(T_a, T_b) \tag{4}$$

where W_{net} and W_{time} are predefined weight factors. As proven in Tiakas et al. (2009), the above measures satisfy the metric space properties.

In the above presentation, the trajectories were considered to be of equal length. In the general case, every trajectory is decomposed to sub-trajectories of equal length (a user parameter) and the overall total distance is calculated by combining the subtotals resulted from all-to-all pairs of sub-trajectories (Tiakas et al., 2009).

3. MMTC problem formalization

Before we present our solutions for the MMTC problem, we first give some definitions utilized hereafter.

Definition 1. (Network). A road *network* is defined as a weighted graph $G = (V, E)$, where V is the set of nodes v_i of the network, each one being of the form $\langle id_i, x_i, y_i\rangle$, denoting the identifier and the point location of the node, and E is the set of edges e_k that connect nodes, each one modeled as a triplet $\langle id_i, id_j, w_{ij}\rangle$ that contains the identifiers of the connected nodes v_i and v_j and the weight (length, travel time, etc.) of an edge. □

Definition 2. (Trajectory). The trajectory T of a moving object is defined as a set of n tuples of the form $\langle x, y, t\rangle$ where $\langle x, y\rangle$ is the point location of the object at time t. The cardinality of T is defined to be the number of point locations it consists of, i.e., $|T| = n$. □

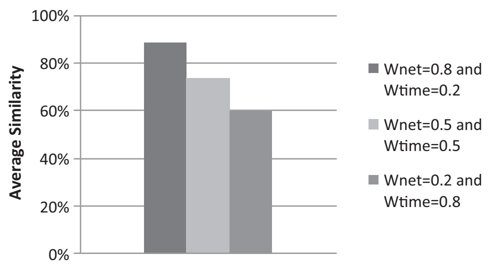


Fig. 3. Behavior of the original similarity measure for various weight values.

Definition 3. (Map-matched trajectory). A trajectory T is considered to be *map-matched* to a network map $G=(V, E)$ if all the $\langle x, y\rangle$ coordinates of its tuples belong to V and they are connected to each adjacent through an edge of E. □

Definition 4. (Map-matched counterpart of a trajectory). A trajectory T' is considered to be the *map-matched counterpart* of a trajectory T with respect to a MM algorithm, if T' is map-matched to a network map $G=(V, E)$ and it is the output of a state-of-the-art MM algorithm over T. □

Definition 5. (Compressed version of a trajectory). A trajectory T' of cardinality m is considered to be a *compressed* version of a trajectory T of cardinality n if (i) $m \le n$, (ii) $\langle x'_1, y'_1\rangle \equiv \langle x_1, y_1\rangle$ and (iii) $\langle x'_m, y'_m\rangle \equiv \langle x_n, y_n\rangle$. In other words, apart from the assumption that the start and end locations are identical, the set of intermediate locations in T' is a subset of those in T. The degree of compression between T' and T, $C(T', T)$, is measured as follows:

$$C(T', T) = 1 - \frac{m}{n} \tag{5}$$

i.e., $C(T', T)$ is a value in $[0\ldots1]$ representing the relative decrease of the number of points composing T' with respect to T. □

In order to compare a trajectory T and a compressed version of it, T', assuming that both are map-matched, a similarity measure is required. As already discussed in Section 2.3, Tiakas et al. (2009) proposed a similarity measure for this purpose, which, however, cannot be adopted as-is in our settings. Eq. (1) uses the variable D_G which is dependent on the network size and, thus, can vary from network to network while D_{time} (Eq. (3)) is network independent. We have tested the similarity measure on 500 random trajectories and their compressed map-matched versions for different W_{net} and W_{time} values. Fig. 3 clearly shows that, in our setting, the similarity measure is dependent on the weight values.

Therefore, we address the issue of network-constrained trajectory comparison by proposing the following approach. We define the cost $c(u_i, u_j)$ to be the size of the shortest path between u_i and u_j. According to this definition, $0 \le c(u_i, u_j) \le |V|$. To perform normalization in $[0\ldots1]$, we exploit on the semantics of *arctan* function.

Formally, we define the distance $d(u_i, u_j)$ between two graph nodes u_i and u_j as follows:

$$d(u_i, u_j) = \frac{2}{\pi}\arctan\left(\frac{c(u_i, u_j) + c(u_j, u_i)}{2div}\right) \tag{6}$$

where *div* is a scaling factor used in order to normalize the result of *arctan* since $c(u_i, u_j)$ is application-oriented as already discussed. It is proportional to the average edge weight. Again, we have tested the proposed similarity measure on 500 trajectories and their compressed map-matched versions (Fig. 4) and it turns out that the similarity measure is not correlated to weights W_{net} and W_{time} or parameter *div*. It is worth mentioning that the average length of a network edge does not differ much from network to network, thus the proposed D_{net} is more compatible to D_{time} than the original proposed by Tiakas et al. (2009).

Proposition 1. *The distance $d(u_i, u_j)$ between two nodes u_i and u_j, which is calculated as in* Eq. (6), *satisfies the metric space properties.*

Proof. First, the cost $c(u_i, u_j)$ is the shortest path between the two nodes, hence $c(u_i, u_j) \ge 0$, which implies that $d(u_i, u_j) \ge 0$. Second, it holds that $d(u_i, u_j) = d(u_j, u_i)$, since the numerator in *arctan*() is symmetric. Third, we will prove that the triangular inequality $d(u_i, u_j) \le d(u_i, u_x) + d(u_x, u_j)$ also holds. Taking into account that $c(u_i, u_j)$ is the shortest path to travel from u_i to u_j, it turns out that the triangular inequality $c(u_i, u_j) \le c(u_i, u_x) + c(u_x, u_j)$ holds for function $c()$. Since *arctan*() is a monotonous increasing function, it turns out that the triangular inequality $d(u_i, u_j) \le d(u_i, u_x) + d(u_x, u_j)$ also holds for function $d()$. □

Having defined the node distance as in Eq. (6), we adopt the rest of the definitions by Tiakas et al. (2009), i.e., the distance $D_{total}(T_a, T_b)$ between two trajectories T_a and T_b is defined as in Eq. (4), where the network distance and time distance components are defined as in Eqs. (2) and (3), respectively, with the single exception that $d(u_i, u_j)$ used in Eq. (2) is defined as in Eq. (6).

Proposition 2. *The distance $D_{total}(T_a, T_b)$ between two trajectories T_a and T_b map-matched to a network map, which is calculated as in* Eq. (4) *when adopting* Eq. (6) *for the definition of the distance between nodes, satisfies the metric space properties.*

Proof. Since $d(u_i, u_j)$ has been already proved to be a metric, it is straightforward to prove that non-negativeness, symmetry and triangular inequality criteria also hold for $D_{total}(T_a, T_b)$. □

Based on the above definitions, the MMTC problem is formalized as follows:

Definition 6. (MMTC problem). Given a road network modeled as a weighted graph $G=(V, E)$ and a trajectory T (not necessarily network constrained), the *map-matched trajectory compression* (MMTC) problem asks to find a network-constrained trajectory T_{MMTC}, which is (a) a compressed version of the map-matched counterpart of T, called T', resulted by a state-of-the-art MM algorithm and (b) as similar to T' as possible.

The degree of compression $Comp(T_{MMTC}, T')$ between T_{MMTC} and T' is measured as follows:

$$Comp(T_{MMTC}, T') = \begin{cases} 1 - \dfrac{|T_{MMTC}|}{|T'|}, & \text{if} |T_{MMTC}| < |T'| \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

i.e., the compression achieved is equal to the relative decrease, if any, of the number of points composing T_{MMTC} with respect to T', while the degree of similarity $Sim(T_{MMTC}, T')$ between T_{MMTC} and T' is measured as follows:

$$Sim(T_{MMTC}, T') = 1 - D_{total}(T_{MMTC}, T') \tag{8}$$

where $D_{total}(T_{MMTC}, T')$ is calculated as in Eq. (4) using the distance between nodes as defined in Eq. (6).

The trajectory T_{MMTC} that maximizes both $Comp()$ and $Sim()$ among all possible network-constrained trajectories is the optimal solution to the MMTC *problem*. □

As already discussed, the MMTC problem involves two subproblems, namely TC and MM. Hence, a naïve approach could be to combine techniques that independently solve TC and MM. However, this is not expected to be the best one can do. Nevertheless, Section 4 discusses the naïve approach, while a novel solution that considers MMTC as a cost-optimization problem is presented in Section 5. In our setting, we do not consider the semantics of the trajectory points. Specific points of interest may be lost during the compression, since we assume that every point is equally important. However, the proposed solutions try to keep the spatio-temporal sketch of each trajectory.

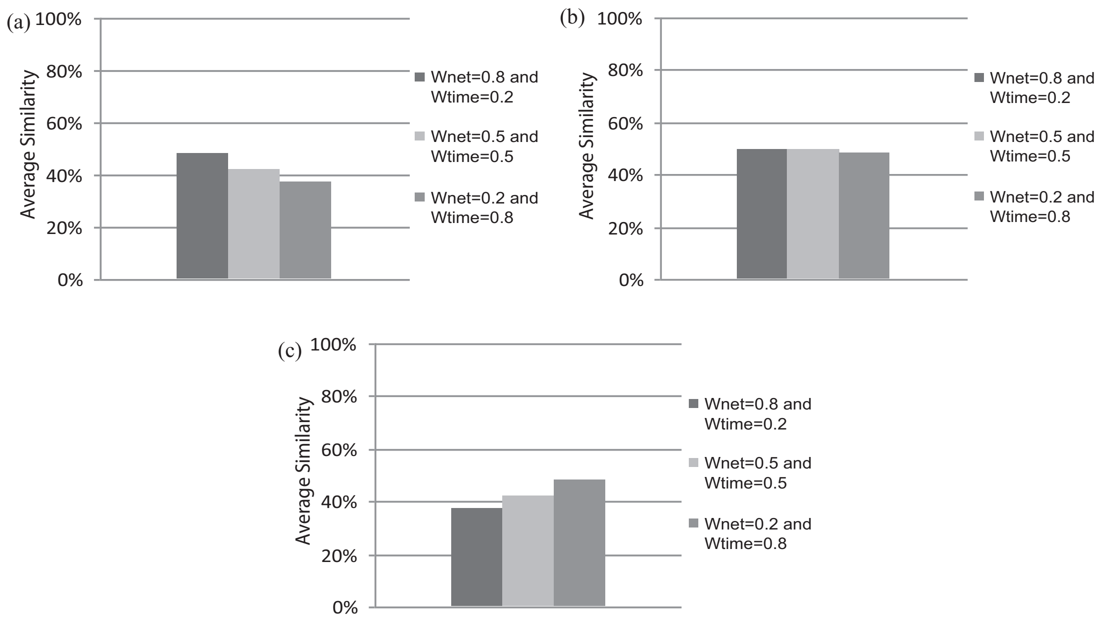


Fig. 4. Behavior of the proposed similarity measure with *div* set to be (a) 1×, (b) 2× and (c) 3× the average length of a network edge for various weight values.

4. Two naïve solutions to the MMTC problem

A first solution to the problem can be provided by utilizing existing algorithms that tackle each sub-problem separately. By combining the algorithms of data compression and map-matching we devise two straightforward alternatives:

- TC + MM: First, we apply compression to the original trajectory; second, we apply map-matching to the output of the first step.
- MM + TC + MM: First, we apply map-matching to the original trajectory; second, we apply compression to the output of the first step; third, we apply map-matching to the output of the second step (since applying TC on a map-matched trajectory does not preserve this property).

Let us discuss about the expected quality and performance of the above naïve solutions. A first remark is that the resulting trajectory of either TC + MM or MM + TC + MM might not be compressed with respect to the original one, since existing map-matching algorithms do not care about compression; instead, in extreme cases (e.g. due to infrequent sampling) it could result in an output that consists of more points than the input trajectory. As a general conclusion, one could agree that the naïve approaches have not been designed to discover the optimal solution.

For offline data we can use the state-of-the-art offline algorithms for TC and MM, namely, Meratnia and de By (2004) and Brakatsoulas et al. (2005), respectively. As far as online data is concerned, we can use the OW algorithm by Meratnia and de By (2004) for the TC problem and Brakatsoulas et al. (2005) again for the MM problem. In order for the offline MM algorithm to work as online, at its starting phase it has to wait for a number of points to arrive, a number which is equal or higher to the number of the "look ahead" points it checks. Also, considering how OW works, practically we have to wait for 3 points to arrive in order for the TC algorithm to start looking for shorter line segments. Moreover, the second run of the MM algorithm (applied after the TC) needs to wait for a compressed segment of a size equal or higher than the "look ahead" value to map-match it.

4.1. Complexity of the naïve methods

Regarding the complexity of the above methods, it turns out that the complexity of both methods for the offline case is $O(n^2)$, where n is the cardinality of the input trajectory, since the complexity of Meratnia and de By (2004) and Brakatsoulas et al. (2005) is $O(n^2)$ and $O(n)$, respectively.

Since the complexity of OW is $O(n^2)$ (Meratnia and de By, 2004), it also turns out that the complexity of the online naïve solution is again $O(n^2)$.

4.2. Issues on the naïve approach

Some technical issues arise when combining MM and TC algorithms. The map-matching algorithm proposed in Brakatsoulas et al. (2005) does not describe how to retain the temporal information along with the spatial locations. However, time is critical in order to perform compression. In order to preserve the temporal information the map-matching algorithm needs to be modified; in particular, for each edge added to the MM result, the time on the corresponding nodes needs to be approximated. To do this, we proceed as follows.

Let P_1 and P_n be the two points and P_r be the node with the temporal information that needs to be calculated, as illustrated in Fig. 5. The timestamp t_r to be put on node P_r is calculated as:

$$t_r = \frac{d_1 \cdot t_n + d_2 \cdot t_1}{d_1 + d_2} \tag{9}$$

where $d_1 = \sum_{i=1}^{r-1} d(P_iP_{i+1})$ and $d_2 = \sum_{i=r}^{n-1} d(P_iP_{i+1})$ with $d(P_iP_j)$ measuring the Euclidean distance between points P_i and P_j. In fact,

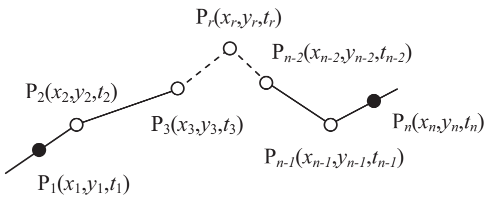


Fig. 5. Node time calculation.

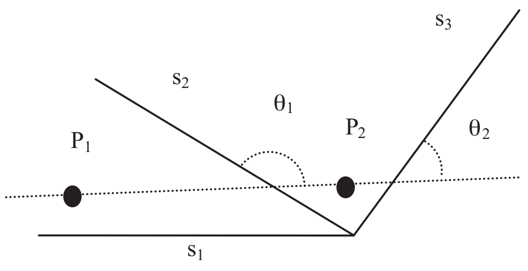


Fig. 6. Case scenario of wrong edge selection.

we approximate the timestamp by assuming that the object moves at a constant speed.

A second issue regarding Brakatsoulas et al. (2005) is that it uses the cosine of the angle formed by the line segment of the last two points and the candidate edge. More precisely, the following two measures are used for choosing among the candidates:

$$S_a(p_i, c_i) = \mu_a \cdot \cos(a_{i,j})^{na}$$
$$S_a(p_i, c_i) = \mu_a - a \cdot d(p_i, c_i)^{nd} \quad (10)$$

where μ_d, μ_a, n_a, n_d and a are predefined scaling factors (Brakatsoulas et al., 2005). Measures s_d and s_a denote distance and orientation, respectively. The higher the sum s of these measures, the better the match to this edge is.

In the experiments presented in Brakatsoulas et al. (2005), n_a is set to 4. However, if the cosine is powered by an even number, we get a positive result regardless of the sign of the angle's cosine. This could cause the algorithm to malfunction in a case such as the one illustrated in Fig. 6.

In this case, the algorithm chooses edge s_1 for the point P_1. When the algorithm proceeds to point P_2, it calculates the scores for edges s_2 and s_3. Supposing the distances from P_2 to both edges are the same, the orientation score will determine which edge will be selected. It is expected that edge s_3 would achieve better score than s_2, since $\theta_2 < \theta_1$, hence $\cos\theta_2 > \cos\theta_1$. However, by using absolute values, we get $|\cos\theta_2| < |\cos\theta_1|$ and it is edge s_2 that is selected instead of s_3, provided that no "look ahead" check is performed. In order to avoid this, we suggest using an odd number for the parameter n_a in the implementation of the naïve TC + MM and MM + TC + MM approaches.

5. A novel solution to the MMTC problem

In this section, we propose a new MMTC method. The main idea is to build the compressed trajectory by replacing some paths of a given map-matched trajectory with shorter ones. This could be done by executing a shortest path algorithm (hereafter, called *SP*) on appropriately selected points of the trajectory without considering the weights of the edges. We ignore the weights in order to get the result with the minimum number of nodes and, hence, achieve a high compression.

5.1. MMTC as a cost-optimization problem

Hereafter, we propose our methodology for finding the optimal tradeoff between compression and similarity, in other words, a method that would find the path on the road network that maximize both compression and similarity. In particular, we adopt the *Minimum Description Length* (MDL) principle. There are two components that comprise MDL, namely $L(H)$ and $L(D|H)$, where H denotes the hypothesis and D denotes the data. According to Lee (2001), "*L(H) is the length, in bits, of the description of the hypothesis; and L(D|H) is the length, in bits, of the description of the data when encoded with the help of the hypothesis*". The best hypothesis H to explain D is the one that minimizes the sum of $L(H)$ and $L(D|H)$.

Mapping the above discussion to our problem, a hypothesis corresponds to a specific compression achieved by a compressed trajectory on the network and the data are the nodes of the map-matched counterpart of the original trajectory. Therefore, finding the best compressed trajectory translates to finding the best hypothesis using the MDL principle. Thus, $L(H)$ represents the compression achieved by a compressed trajectory and $L(D|H)$ represents the difference between the compressed trajectory and the original one.

We formulate $L(H)$ as the binary logarithm of the ratio of the number of points $|T_{MMTC}|$ of the resulted trajectory over the number of points $|T'|$ of the map-matched counterpart of the original trajectory, i.e.:

$$L(H) = \log_2\left(\frac{|T_{MMTC}|}{|T'|}\right) - \log_2 0.001 \quad (11)$$

On the other hand, we formulate $L(D|H)$ as the binary logarithm of the distance between the resulted trajectory T_{MMTC} and the map-matched counterpart T' of the original one as defined in Eq. (4).

$$L(D|H) = \log_2 D_{total}(T_{MMTC}, T') - \log_2 0.001 \quad (12)$$

In both equations, the binary logarithm as well as the subtraction by a constant ($\log_2 0.001$) are used in line with Grünwald et al. (2005) because the two measures, $L(H)$ and $L(D|H)$, are real numbers. To clarify this point, we encode a real number x assuming a precision δ, so that the encoded number x_δ satisfies $|x - x_\delta| < \delta$. If x is large and $x \approx x_\delta$, $L(x) = \log_2 x - \log_2\delta$. Here, we need a precision of 3 fractional digits, so we set $\delta = 1/1000$, therefore $L(x)$ is equal to $\log_2 x - \log_2 0.001$.

Recalling Definition 6 and according to the previous discussion, the solution we envisage for the MMTC problem is the path that minimizes the sum $L(H) + L(D|H)$ adopting the MDL principle. Unfortunately, the cost of finding this path is prohibitive since we need to consider every combination of shortest paths between any two points of the original trajectory. Actually, it is similar to the problem of finding all possible acyclic paths shorter than a predefined length n between two graph nodes. In order to find all the paths shorter than n between two graph nodes, we have to use a depth-first search on the graph (Migliore et al., 1990). The branching factor b would be equal to the average number of neighboring nodes to any graph node. The graph depth on the other hand would be equal to n, since we only need the paths shorter than n. According to Korf (1985), the complexity of the depth-first search is $O(b^d)$ where d is the graph depth. In our case $d = n$, thus the complexity of finding all paths shorter than n between two graphs nodes will be $O(b^n)$, which means the time complexity of the optimal solution would be exponential.

Apparently, the computational cost of the optimal solution makes it not applicable for long trajectories (consisting of tens or hundreds of points; see Table 1 in Section 6 for realistic trajectory lengths). Since we cannot expect to find the optimal trajectory in reasonable time, we propose an algorithm that would approximate it. The two approximate solutions, for offline and online settings, respectively, are proposed in the following subsections.

5.2. Offline MMTC

The main idea of our approach is illustrated in Fig. 7 whereas the proposed approximate algorithm, called MMTC-App-offline is listed in Fig. 8. First, we calculate a map-matched counterpart of the original trajectory using a state-of-the-art MM algorithm (step 1). Then (step 2) we get all SPs between any two nodes of T' (the set of all possible shortest paths is known in advance). We set the first point of T' as the examined point (step 3). For each next point

Table 1
Dataset properties.

Dataset	Milano	S_{low_low}	S_{low_high}	S_{high_low}	S_{high_high}
# of trajectories	500	500	500	500	500
# of points	26 ... 114 (avg: 40.9)	160	40	160	40
# of edges	9 ... 216 (avg: 67.9)	40	40	40	40
Length (m)	6321 ... 64,234 (avg: 20,873)	2011 ... 26,818 (avg: 7139)	2141 ... 45,436 (avg: 7674)	2087 ... 28,068 (avg: 7147)	2045 ... 30,837 (avg: 7087)
Speed (m/s)	0.9 ... 14.2 (avg: 3.6)	0.8 ... 11.2 (avg: 3.0)	3.7 ... 77.7 (avg: 13.1)	0.9 ... 11.7 (avg: 3.0)	3.5 ... 52.7 (avg: 12.1)

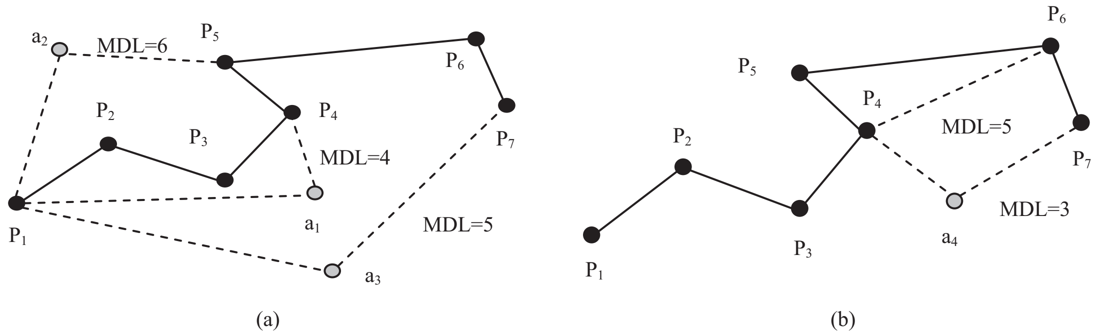


Fig. 7. Example of MMTC-App-offline algorithm: shortest paths and their MDLs (a) from the first node and (b) from the fourth node.

we compute the shortest path from it to the examined point and its MDL value (steps 6–11). The timestamps of each new node are calculated according to Eq. (9) (step 10). The *SP* with the lowest MDL is chosen and its last node is set as the new examined point (steps 12–16).

As an example, given the map-matched counterpart T_{MM} consisting of seven points $\{P_1, P_2, P_3, P_4, P_5, P_6, P_7\}$, the algorithm calculates the MDL of every path as illustrated in Fig. 7(a) and chooses the one with the lowest MDL value. Let us suppose that the lowest MDL value is given by the path from P_1 to P_4 via node a_1. Then we can replace the sub-path $\{P_1, P_2, P_3, P_4\}$ of the temporary result by $\{P_1, a_1, P_4\}$ (step 21). Our trajectory has become one point shorter.

```
Algorithm MMTC-App-offline
Input: T: given trajectory
Output: T_new: the compressed trajectory on the network
Begin
1.    Find the map-matched counterpart T′ of T
2.    Get all shortest paths between the nodes of T′
3.    Set the first node of T′ as the beginning node N
4.    While N.position<T′.length
5.      Set i=N.position and MDL=MAX_INTEGER
6.      While i<T′.length
7.        Set i=i+1
8.        Find the shortest path test between N and the i-th node of T′
9.        If test.length is shorter than the original path between N and
10.       the i-th node
11.         Calculate the time values on the new nodes
12.         Set pros as the MDL of test
13.         If pros<=MDL
14.           Set SP=test
15.           Set N=node_at_position(i);
16.           Set MDL=pros
17.       If SP is empty (no shortest path found)
18.         Add the remaining points to T_new
19.         Set N=node_at_position(T′.length)
20        Else
21          Add SP to T_new

End
```

Fig. 8. MMTC-App-offline algorithm.

The algorithm stores this result and continues by considering the last node of the already found shortest path as the examined point (step 15) and by checking the shortest paths from this node to its next ones. In our running example, P_4 is checked against its next points. Finally, the shortest path with the best score is the one from P_4 to P_7 via a_4, as illustrated in Fig. 7(b). Since P_7 is the end point of the trajectory, the algorithm terminates by returning the compressed trajectory $\{P_1, a_1, P_4, a_4, P_7\}$.

On the other hand, i.e., in case no shortest path is found in a sub-path which is under evaluation, the algorithm adds the remaining points to the output (step 18) and, then, terminates. The result is a map-matched trajectory as defined since it only consists of nodes of adjacent edges.

Being an approximate algorithm, MMTC-App does not guarantee to find the optimal path. This is because the MDL of the shortest path between the first and the last node may be higher than the MDL of the shortest path between e.g. the first and the fourth point, but, finally, the MDL of the path formed by the calculated sub-paths (in our example $\{P_1, a_1, P_4, a_4, P_7\}$), may be higher than the initial MDL of the first and the last point (in our example, $\{P_1, a_3, P_7\}$).

As mentioned in the previous discussion (recall step 2), the MMTC-App algorithm requires the shortest paths from the original nodes to all others to be pre-calculated. This is an offline procedure of general purpose, the execution time of which depends on the network size and sparsity. Theoretically, the time complexity for running an all-pair shortest path algorithm on a network $G(V, E)$ is $O(|V|^2 \log|V|)$ (Johnson, 1977), when the network is sparse or, in other words, the density D of the network graph, defined as follows:

$$D = \frac{2|E|}{|V| \cdot (|V| - 1)}, \quad 0 \le D \le 1 \tag{13}$$

is very low (close to zero), which is indeed the case in real-world traffic networks. In this case, the execution time practically ranges from a few seconds to several minutes (as it will be discussed in detail in the experimental study in Section 6).

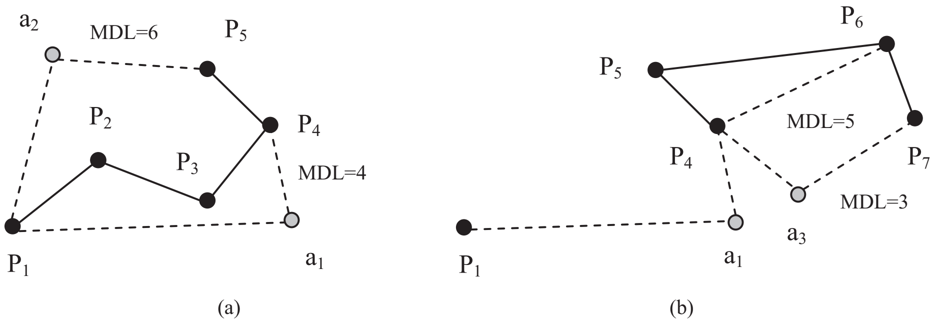


Fig. 9. Example of the MMTC-App-online algorithm: shortest paths and their MDLs (a) from the first node and (b) from the fourth node.

With this assumption in hand, we provide a complexity analysis of the proposed approximate algorithm.

Lemma 1. *Assuming that shortest paths between nodes have been pre-calculated, the complexity of MMTC-App-offline algorithm is $O(n^2 \log n)$ on average, where n is the number of points composing the trajectory.*

Proof. The cost for calculating the MDL for each shortest path is linear to n, since it is only the equivalent points of each trajectory that are checked. In the best case, only the MDLs of the SPs from the first trajectory point to all other are checked, and the SP from the first to the last point is selected. On the average, the best shortest path would be that from the examined point to the middle point of the respective sub-trajectory. Thus, on the average $n \log n$ sub-trajectories are expected to be checked, resulting in $O(n^2 \log n)$ average complexity. □

In fact, the cost of the algorithm is expected to be even lower than quadratic since not all the points of a route have a shorter path between them than the original. This claim is supported by the results of our experimental study to be presented in Section 6.

5.3. Online MMTC

In case of online data, the algorithm works as follows. Initially, it map-matches every incoming point. When the map-matching consists of 3 network nodes (i.e., 2 adjacent edges), it checks if the shortest path between the first and the third node is shorter than the path composed of the 3 nodes. If not, it waits for another node and checks again for a shorter path between the first and the new node, and so on. When it finds a path shorter than the original, a counter is increased by one and the algorithm continues the same process until the counter reaches a predefined threshold r. When the threshold is reached, the MDL of each shortest path from the initial node is calculated, and the algorithm chooses the best path, as described previously. Then, it stores that path; the end node of this path is considered as the examined point; the counter is set to zero and the algorithm continues until no more points arrive. It is obvious that the higher the value of the threshold r is, the longer time the algorithm waits to gather enough points to process in order to return a result.

Fig. 9 illustrates an example of the above procedure while Fig. 10 presents the proposed MMTC-App-online algorithm. In our example, let us suppose that threshold $r = 2$. The algorithm map-matches incoming points until we get 3 network nodes (P_1, P_2 and P_3) (steps 3–5). Then, it checks for a SP that is shorter than the original path $\{P_1, P_2, P_3\}$ (i.e., a direct connection between P_1 and P_3) (step 11). None is found, so more incoming points are map-matched until we get another network node (P_4). Then the algorithm checks for a SP shorter than $\{P_1, P_2, P_3, P_4\}$, and it finds path $\{P_1, a_1, P_4\}$. A counter is increased by one (step 12) and checked whether it is equal to threshold r (step 15). It is not, so more incoming points are map-matched and another network node is found, P_5. A SP that is shorter than the original path is then found between P_1 and P_5 through a_2. The counter is increased again and now it is equal to r. The MDLs of the two SPs are calculated and the one that gives the best score (path $\{P_1, a_1, P_4\}$) is chosen to replace the original $\{P_1, P_2, P_3, P_4\}$ (steps 16–19).

We already have one network edge (P_4, P_5) in our hands, so the algorithm map-matches incoming points until we get one more network node (P_6). The counter is reset to zero and a SP between P_4 and P_6 is found that is shorter than $\{P_4, P_5, P_6\}$ (it is a direct connection between P_4 and P_6). The counter is increased by one, but it is smaller than r so more incoming network nodes are needed. Incoming points are map-matched until we get P_7 and then we check for a SP. The SP $\{P_4, a_3, P_7\}$ that was found is shorter than the original path, so we increase the counter by one and now it is equal to r. The algorithm checks the MDLs of the two SPs and it chooses the path $\{P_4, a_3, P_7\}$ as a substitute to $\{P_4, P_5, P_6, P_7\}$. If there are no

```
Algorithm MMTC-App-online
Input: r: predefined threshold
Output: T_new: the compressed trajectory on the network
Begin
1.    Set counter=0
2.    While incoming_points_stream!=NULL
3.      If T'.length<3 or counter>1
4.        Map-match the incoming point
5.        Store any new node to T'
6.      Else
7.        Set N as the first node of T'
8.        Get the shortest paths from N
9.        Set i=T'.lastelement.position
10.       Set SP[counter].path as the shortest path between N and the i-
      th node of T'.
11.       If SP[counter].path.length<T'.length
12.         Set counter=counter+1
13.         Calculate the time value on the new nodes in
            SP[counter].path
14.         Set SP[counter].MDL as the MDL of SP[counter].path
15.         If counter=r
16.           Find the position k of the element in SP with the best
          MDL
17.           Add SP[k].path to T_new
18.           Remove the substituted nodes from T'
19.           Set counter=0
20.     Add the remaining nodes to T_new

End
```

Fig. 10. MMTC-App-online algorithm.

more incoming points, path $\{P_1, a_1, P_4, a_3, P_7\}$ is considered as the map-matched trajectory of the moving object.

Regarding all-shortest-paths, they can be pre-calculated as described in the previous section. However, in case this is not possible due to storage limitations, the needed shortest paths can be calculated on the fly. For sparse graphs, Dijkstra's algorithm (Dijkstra, 1959) can be implemented more efficiently by storing the graph in the form of adjacency lists and using a binary heap, pairing heap, or Fibonacci heap as a priority queue to implement the Extract-Min function efficiently. With a binary heap, the algorithm requires $O(|V|^2 \log |V|)$ time (Barbehenn, 1998).

Lemma 2. *Assuming that shortest paths between nodes have been pre-calculated, the complexity of MMTC-App-online algorithm is $O(rn \log r)$ on average, where n is the number of points composing the trajectory and r is the predefined threshold of the SPs to be examined at every step.*

Proof. In the worst case scenario, where all nodes checked have a shorter path than the original one, the algorithm needs to calculate the MDL of r paths for every r points. The cost for calculating the MDL for each path is linear to r, thus, for r SPs the complexity would be $O(r^2)$. On the average, the best shortest path would be that from the anchor point to the middle point of the respective sub-trajectory. Thus, if the original trajectory has a cardinality of n, $n/r \log r$ sub-trajectories will be checked, resulting in $O(rn \log r)$ average complexity. □

5.4. Controlling the compression rate

In this section we present a simple yet effective way to tune the inevitable tradeoff between compression and similarity in the MDL formalization of the MMTC problem. We can control the rate of compression of the MMTC algorithm by replacing $L(H)$ of Eq. (11) with the following equation:

$$L(H) = \log_2 \left(\frac{|T_{MMTC}|}{|T'|} \right)^p - \log_2 0.001 \quad (14)$$

where p is a parameter for controlling the rate of the compression. Values of p higher than 1 amplify the difference between different compression rates, thus making compression more significant to the alternate path selection. As a result, paths that offer higher compression are preferred, resulting in a high compressed final output. On the other hand, p values lower than 1 make the differences between different compression rates less significant, thus making similarity more important for the path selection. This way the result is less compressed and more similar to the initial trajectory. The effect of p on the resulting compression will be demonstrated in Section 6 for both the offline and online case.

6. Experimental study

In this section, we first present the methodology we followed in order to evaluate the algorithms we proposed for the MMTC problem (namely TC + MM, MM + TC + MM and MMTC-App, either offline or online). Then, we provide the results of our experimental study that demonstrate the performance of the proposed techniques in terms of quality of the results and processing time.

6.1. Datasets and experimental settings

In the MOD literature, synthetic trajectory datasets are often used to evaluate techniques and methodologies. In our study, we used Oldenburg road network (Brinkhoff) in order to generate several versions of trajectory datasets. The network consists of 6106 nodes and 7036 edges. With respect to Eq. (13), its density is $D = 0.00038$ resulting in 3 min running time to pre-calculate all shortest paths between any two nodes of the network using Johnson's algorithm (Johnson, 1977). Regarding the synthetic datasets, we built several of them varying the number of trajectories produced, their length, speed, and agility. More specifically, in order to create random movements of either low or high agility we built trajectories as follows: for the starting point, we pick a node randomly; the next node is decided among the end nodes of the adjacent edges according to a probability, which is either (a) proportional or (b) inversely proportional to the resulting deviation in direction, measured in angle degrees ranging in $[0^\circ \ldots 180^\circ]$. (To avoid returning to the previously visited node, we ignore it, assuming we have more choices, i.e., we have not reached a dead-end.) These two options (i.e. a and b) result in low versus high agility, respectively.

Apart from synthetic trajectory data, we have included in our experimental study a real dataset gathered from the city of Milano, which includes movement of vehicles in the city (Agenzia Milanese Mobilitá Ambiente). The Milano network consists of 14,021 nodes and 26,849 edges (density $D = 0.00027$, 53 min to pre-calculate all shortest paths).

In Table 1, we list the specifications of the datasets used in our experiments; the real dataset is labeled as 'Milano' whereas the synthetic datasets built on the Oldenburg road network are labeled as S_{x_y} where x denotes low/high agility and y denotes low/high speed.

We have used the following methodology in order to evaluate the proposed techniques:

- a set of trajectories is created (in case of synthetic data) or collected (in case of real data);
- (in case of synthetic data only) random noise is added to each trajectory in order to simulate real world (e.g. GPS recordings);
- for each trajectory, a compressed and map-matched version of it is constructed, following one of the above techniques;

The resulting trajectories are compared against the original ones (or their map-matched counterparts, in case of Milano dataset) with respect to (a) their similarity as defined in Eq. (8), (b) the compression achieved as defined in Eq. (7), and (c) the execution time required to produce the results.

Regarding step (2) above, we add Gaussian noise to the coordinates of every point with mean $\mu = 0$ and deviation $\sigma = 2$, as argued to be typical noise values for a GPS receiver (Pfoser and Jensen, 1999), and we set the sampling rate to be 15 s. Note that we have not experimented with different sampling rates since experimenting with trajectories of variable speed values has the same effect as tuning the sampling rate of the trajectories. Indeed, increasing the speed while the sampling rate is fixed or increasing the sampling rate while the speed is fixed, produces the same result as far as the location of the points is concerned.

Regarding TC, we have used the offline compression algorithm proposed in Meratnia and de By (2004) for offline data with Douglas–Peucker threshold varying from 0.1% to 2% of the trajectory length for removing farthest points, in order to enforce different compression rates. As far as online data is concerned, we used the OW algorithm proposed in Meratnia and de By (2004) with Douglas–Peucker threshold values varying from 15 to 1015 m in case of synthetic data and threshold values varying from 15 to 3015 in case of real data. In order to obtain different compression rates from the MMTC-App algorithm, we have tested our method with different p values varying from 0.1 to 4 (recall that p is a parameter for controlling the rate of compression; see Section 5.4). An important advantage of our method in comparison with the naïve methods is that it produces stable compression rate for the same p value irrespective of the trajectory length. For example, our method always results in 50–60% compression for $p = 2$, which is not the case for the naïve methods. In the case of online data, since the trajectory

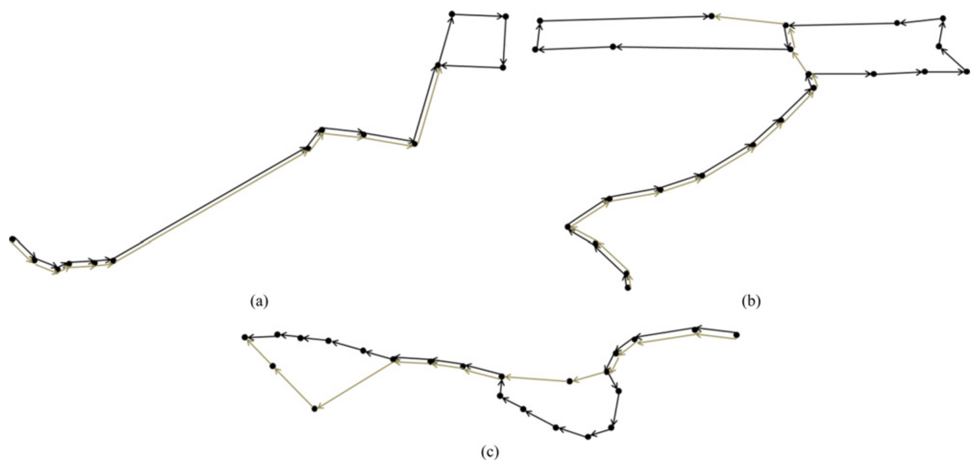


Fig. 11. Original and compressed trajectories offering similarity of (a) 60%, (b) 47%, and (c) 32%.

length is not known in advance, we have to experiment with different threshold values to obtain a wide range of compression rates (e.g. a threshold value of 50 m can result in different compression rates depending of the final length of the trajectory). This issue makes naïve methods less adaptive to online data applications.

Regarding MM, we have used the map-matching algorithm proposed in Brakatsoulas et al. (2005) with the parameters set to the default values, i.e., "look ahead" = 4, $\mu_d = 10$, $n_d = 1.4$, $\mu_a = 10$ and $a = 0.17$, with the single exception of n_a which was set to 3, i.e., an odd number was chosen as explained in Section 4.1. As far as Milano data is concerned, in order to have a map-matched counterpart to compare our results (since the original map-matched trajectory is not known a priori) we use (Brakatsoulas et al., 2005) with "look ahead" = 4. This is in line with the MMTC problem definition (see Definition 6) where it is declared that a state-of-the-art MM algorithm should be adopted in order to generate the map-matched counterpart of the original trajectory.

Regarding the total distance measure D_{total}, we set $W_{net} = W_{time} = 0.5$ in order to enforce both component distances to be considered as equally significant, although using other values does not affect the final result, and we set *div* to be twice the average length of network edges, i.e., 180 m for Oldenburg and 310 m for Milano (see Section 3 for arguments in favor of these settings). In order to illustrate the physical interpretation of the similarity measure used, Fig. 11 depicts three trajectories (black arrows), their compressed versions (gray arrows), and their similarity percentage (we do not take into account the time dimension).

The same data sets have been used for the experiments with online data assuming that the trajectory points are not known a priori, but they are gathered one by one at real time. Regarding the MMTC-App method for online data, the threshold value *r* (i.e., the number of shortest paths to be checked before storing the result) is set to 5, 10, and 15.

All the proposed algorithms have been implemented in Java. The experiments were run on a PC with Intel Pentium 4 640 at 3.2 GHz, 2 GB RAM and 120 GB hard disk.

6.2. *Evaluating the offline case*

The first set of experiments concentrates on the effectiveness of the proposed offline methods, i.e., as high compression (according to Eq. (7)) and overall similarity (according to Eq. (8)) as possible. We note here that MMTC-App has a lower and upper bound regarding the achieved compression rates, while naïve methods only have an upper bound. The upper bound is the maximum possible compression (i.e. the shortest path between the first and the last point of the map-matched trajectory), while the lower bound of the MMTC-App is the result of the MMT-App algorithm when only choosing between candidate shortest paths of a sub-route and not between the shortest paths and the original sub-route.

Fig. 12 compares the naïve methods with MMTC-App-offline for the real data case. Clearly, the naïve methods cannot present better similarity than MMTC-App for the same compression rates.

A similar behavior appears when we experiment with synthetic datasets. Fig. 13 shows the effectiveness of the proposed techniques for low speed data with different agility values. It is clear again that MMTC-App-offline offers better similarity for the same compression rates. Moreover, higher agility has resulted in higher compression rates.

When dealing with high speed data, Fig. 14 shows that MMTC-App-offline again outperforms the naïve methods. Once more, higher agility results in higher compression rate.

Having synthetic datasets in our hands, we can experiment with the complexity of the techniques with respect to the number of points that compose a trajectory, which is the key factor that affects their cost (recall the discussion about complexity in Section 5.2). Toward this goal, we varied the number of points in each trajectory and performed MMTC in the datasets produced. Fig. 15 illustrates the response time of each method as the number of points grows. The naïve MM + TC + MM results in the highest execution time because of the double use of the MM algorithm. On the contrary, TC + MM turns out to be inexpensive, since the MM component of this method is applied on fewer points. MMTC-App-offline performs in between the two naïve approaches because its MM component is applied on the original set of points (which is obviously more populated than the compressed one that applies for MM + TC). With respect to the theoretical complexity discussed

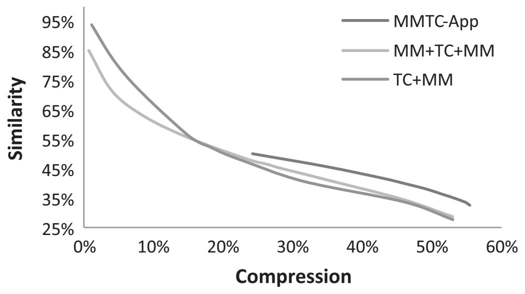


Fig. 12. Compression versus similarity – real offline data (Milano).

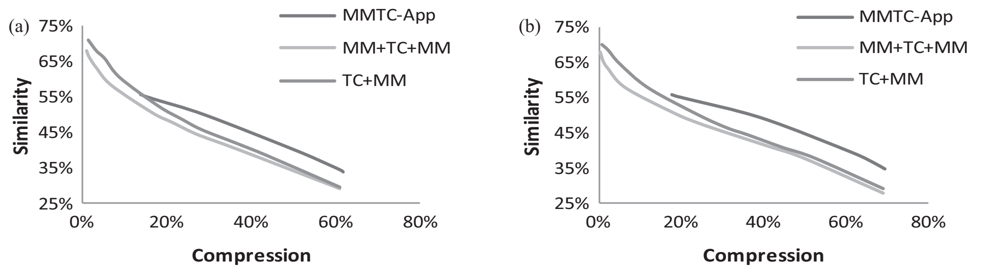


Fig. 13. Compression versus similarity – synthetic offline data (a) $S_{\text{low\_low}}$, (b) $S_{\text{low\_high}}$.

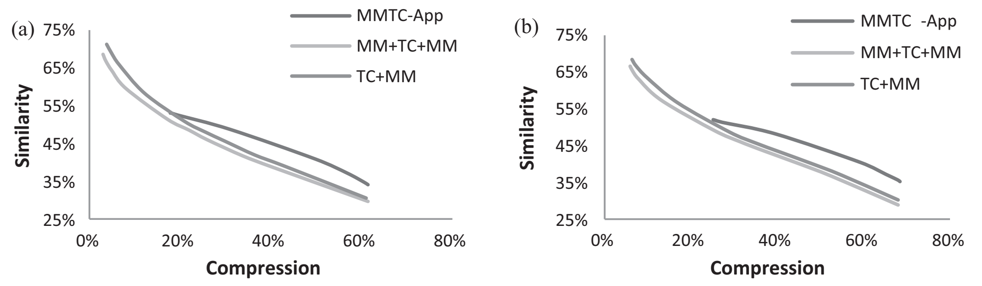


Fig. 14. Compression versus similarity – synthetic offline data (a) $S_{\text{high\_low}}$, (b) $S_{\text{high\_high}}$.

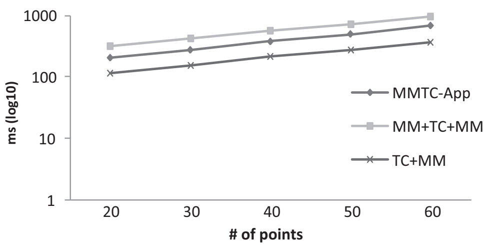


Fig. 15. Execution time – synthetic offline data.

in Section 5.2, in fact the running time of the proposed MMTC-App method turns out to be lower than quadratic.

Fig. 16 illustrates the behavior of MMTC-App-offline for various p values. It is obvious that as p increases, the method reaches the maximum possible compression rate. We can also observe that MMTC-App also has a minimum compression rate as explained earlier. It is clear that we can achieve a great variety of compression rates by using p values that vary between 0.5 and 2.

6.3. Evaluating the online case

As far as online data is concerned, MMTC-App-online is tested with different values of the threshold r (r = 5, 10, 15) that corresponds to the number of paths shorter than the original to be gathered before a decision is made. MMTC-App-online again has a lower compression bound as explained in Section 6.2.

Fig. 17 depicts the results of MMTC-App-online compared to the two naïve online methods. Same as the offline case, MMTC-App-online achieves better similarity for the same compression rates. While r does not affect the quality of the results, it restrains the maximum possible compression. Higher r values result in higher possible compression rates.

Figs. 18 and 19 illustrate the quality of MMTC results for low and high agility trajectories, respectively. Again, for low speed data, MMTC-App-online achieves better similarity than the naïve methods for equivalent compression rates (Fig. 18). In the same figure, we can observe the behavior of MMTC-App-online for different r values: while setting r = 5 offers good similarity results, it restrains the maximum possible compression to 40%. As a result, values of r greater than 10 turn out to be more flexible. For high agility data (Fig. 19), MMTC-App-online again outperforms the two naïve methods regardless the value of parameter r.

When dealing with online data, each method has to wait for a number of points before returning a compressed result. Fig. 20 depicts the average number of points each method has to receive before it is able to return its output. Note that when a method is waiting for incoming points the user still gets a result, the original trajectory, a result to be updated as soon as enough points are

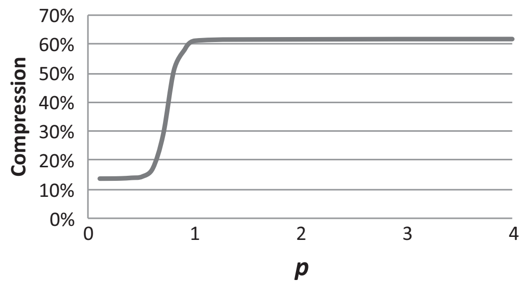


Fig. 16. Compression rates for different p values.

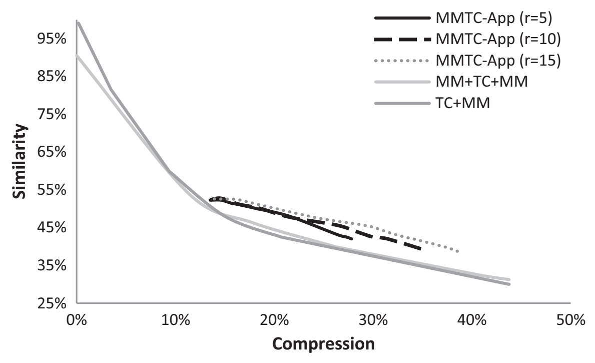


Fig. 17. Compression versus similarity – real online data.

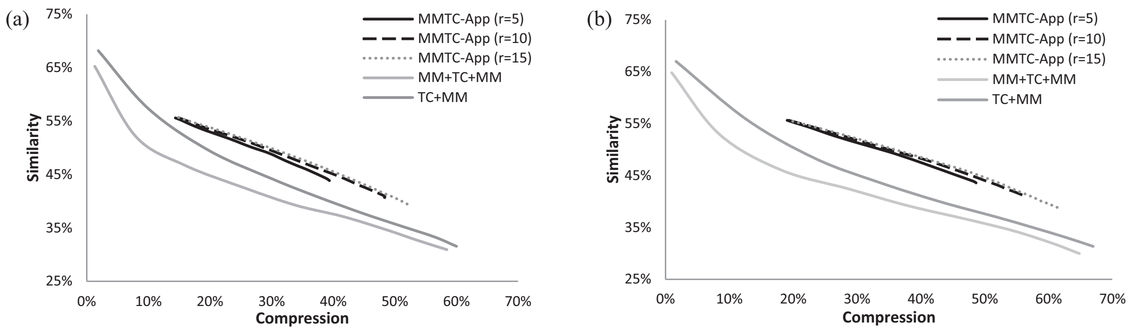


Fig. 18. Compression versus similarity – synthetic online data (a) $S_{\text{low\_low}}$, (b) $S_{\text{low\_high}}$.

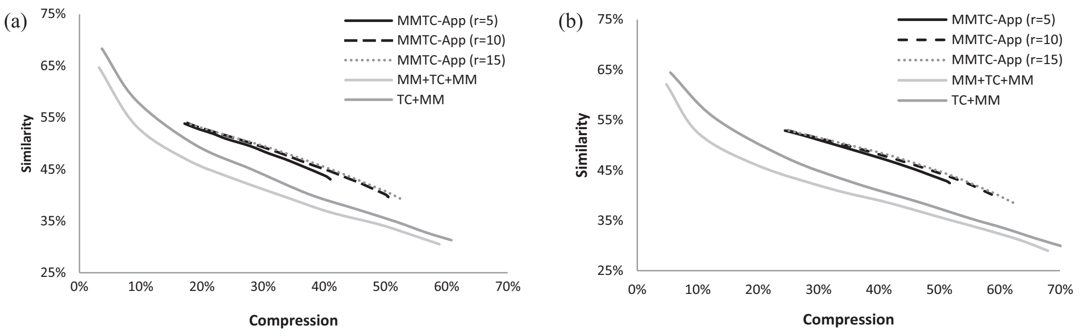


Fig. 19. Compression versus similarity – synthetic online data (a) $S_{\text{high\_low}}$, (b) $S_{\text{high\_high}}$.

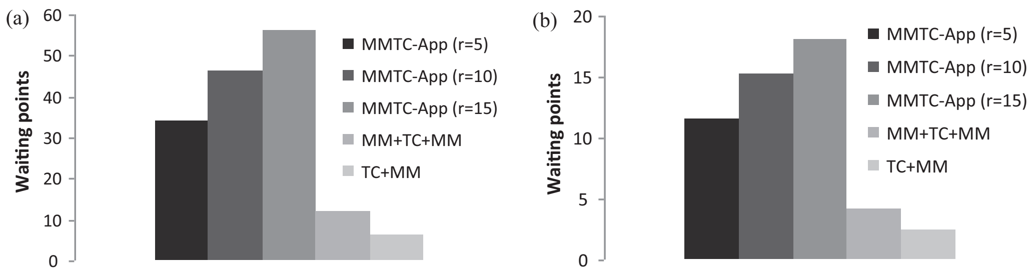


Fig. 20. Waiting points – synthetic online data (a) low speed, (b) high speed.

collected (in a realistic scenario, consider a software in a GPS navigation device that visualizes the user's current trajectory on the screen, and online compresses it).

In order to understand the effect of parameter r to the maximum achievable compression, Fig. 21 illustrates the compression rates achieved by MMTC-App-online for different p values. Obviously, the compression rate increases with p and a variety of compression rates is achieved using p between 0.5 and 2 (as it happens in the offline case – see Fig. 16 – though less smoothly than in the online case).

As far as the execution time is concerned, in the online case there is no a priori knowledge of the number of points of each trajectory. Hence, Fig. 22 illustrates the average execution time per point for each method. Higher r values do not increase the execution time noticeably. The most important conclusion drawn from this figure is the low execution time per point for all methods (a few ms), which renders them suitable for online data applications since hundreds of points can be online processed per second.

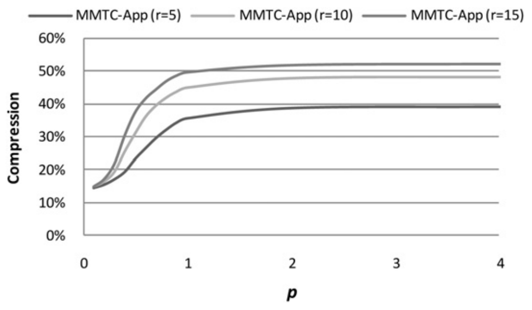


Fig. 21. Maximum possible compression for r = 5, 10, 15.

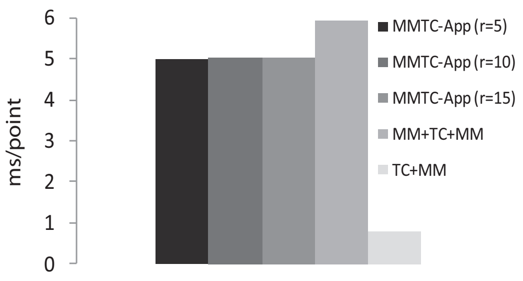


Fig. 22. Execution time ms/point for each method – synthetic online data.

7. Conclusion

The MOD bibliography offers solutions to the problems of trajectory compression and map-matching separately, but none of them satisfies the combined requirement for map-matched trajectory compression.

In this paper, apart from straightforward solutions to the MMTC problem based on existing TC and MM solutions, we have proposed a novel MMTC method, viewing MMTC as a cost-optimization problem and exploiting on the MDL principle and the shortest path concept.

Clearly, the naïve methods cannot offer high quality solutions to the MMTC problem. This is an expected outcome since the two sub-problems seem to be independent at a first sight. On the one hand, an MM solution does not guarantee compression while a TC solution is a simplified route that is not network-constrained.

In contrast, our novel MMTC-App approach (whether offline or online) considers the network parameters in order to offer a compressed result that also resembles the initial object trajectory. Our experimental study shows that our methods can offer nice solutions to both versions (offline and online) of the MMTC problem. Moreover, our methods turn out to be robust irrespective of the variations in the length, agility, and speed of trajectories.

With respect to running time, MMTC-App has shown a very good behavior, practically being lower than quadratic with respect to the number of points that compose the input trajectory.

As a future work, we intend to further evaluate the quality of MMTC-App results in trajectory data mining tasks, such as discovery of hot paths (Sacharidis et al., 2008) and clustering (Lee et al., 2007; Pelekis et al., 2009). Moreover, we will extend our method to take into account possible points of interest of each trajectory. The *Semantic Trajectory Compression* problem would ask to compress a trajectory while keeping as many important points (e.g., landmarks) as possible.

Acknowledgements

A preliminary version of this paper appears in Kellaris et al. (2009). Research partially supported by the European Union under the FP7/ICT project MODAP (http://www.modap.org), the FP7/ICT project DATASIM (http://datasim-fp7.eu), and the ESF/COST Action MOVE (http://www.move-cost.info).

References

Agenzia Milanese Mobilitá Ambiente. URL: http://www.ama-mi.it/ (accessed 17.11.11).
Barbehenn, M., 1998. A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices. IEEE Transactions on Computers 47, 263.
Bernstein, D., Kornhauser, A., 1996. An introduction to map matching for personal navigation assistants. Technical report, New Jersey TIDE Center Technical Report.
Brakatsoulas, S., Pfoser, D., Salas, R., Wenk, C., 2005. On map-matching vehicle tracking data. In: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB).
Brinkhoff, T., Network-based Generator of Moving Objects. URL: http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/ (accessed 17.11.09).
Cao, H., Wolfson, O., 2005. Nonmaterialized motion information in transport networks. In: Proceedings of the 10th International Conference on Database Theory (ICDT), pp. 173–188.
Chen, L., Tamer Özsu, M., Oria, V., 2005. Robust and fast similarity search for moving object trajectories. In: Proceedings of ACM SIGMOD.
Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numerische Mathematik 1 (1), 269–271.
Douglas, D., Peucker, T., 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. The Canadian Cartographer 10 (2), 112–122.
Frentzos, E., Gratsias, K., Theodoridis, Y., 2007. Index-based most similar trajectory search. In: Proceedings of the 23rd International Conference on Data Engineering (ICDE).
Greenfeld, J., 2002. Matching GPS observations to locations on a digital map. In: Proceedings of the 81st Annual Meeting of the Transportation Research Board.
Grünwald, P., Myung, I.J., Pitt, M., 2005. Advances in Minimum Description Length: Theory and Applications. MIT Press.
Johnson, D.B., 1977. Efficient algorithms for shortest paths in sparse networks. Journal of the ACM 24 (1), 1–13.
Kellaris, G., Pelekis, N., Theodoridis, Y., 2009. Trajectory compression under network constraints. In: Proceedings of the 11th International Symposium on Spatial and Temporal Databases (SSTD).
Korf, R.E., 1985. Depth-first iterative-deepening: an optimal admissible tree search. Artificial Intelligence 27 (1), 97–109.
Lee, J.-G., Han, J., Whang, K.-Y., 2007. Trajectory clustering: a partition-and-group framework. In: Proceedings of ACM SIGMOD.
Lee, T.C.M., 2001. An introduction to coding theory and the two-part minimum description length principle. International Statistical Review 69 (2), 169–184.
Meratnia, N., de By, R.A., 2004. Spatiotemporal compression techniques for moving point objects. In: Proceedings of the Extending Database Technology (EDBT).
Migliore, M., Martorana, V., Sciortino, F., 1990. An algorithm to find all paths between two nodes in a graph. Journal of Computational Physics 87, 231–236.
Pelekis, N., Kopanakis, I., Kotsifakos, E.E., Frentzos, E., Theodoridis, Y., 2009. Clustering trajectories of moving objects in an uncertain world. In: Proceedings of IEEE International Conference on Data Mining (ICDM).
Pfoser, D., Jensen, C.S., 1999. Capturing the uncertainty of moving-object representations. In: Proceedings of the 6th International Symposium on Spatial Databases (SSD).
Potamias, M., Patroumpas, K., Sellis, T., 2006. Sampling trajectory streams with spatiotemporal criteria. In: Proceedings of the Scientific and Statistical Database Management (SSDBM).
Quddus, M., Ochieng, W., Zhao, L., Noland, R., 2003. A general map matching algorithm for transport telematics applications. GPS Solutions Journal 7 (3), 157–167.
Sacharidis, D., Patroumpas, K., Terrovitis, M., Kantere, V., Potamias, M., Mouratidis, K., Sellis, T., 2008. Online discovery of hot motion paths. In: Proceedings of the Extending Database Technology (EDBT).
Tiakas, E., Papadopoulos, A.N., Nanopoulos, A., Manolopoulos, Y., Stojanovic, D., Djordjevic-Kajan, S., 2009. Searching for similar trajectories in spatial networks. Journal of Systems and Software 82 (5), 772–788.
Vlachos, M., Kollios, G., Gunopulos, D., 2002. Discovering similar multidimensional trajectories. In: Proceedings of the 18th International Conference on Data Engineering (ICDE).
Xu, Y., Lee, W., 2007. Compressing moving object trajectory in wireless sensor networks. International Journal of Distributed Sensor Networks 3 (2), 151–174.
Yin, H., Wolfson, O., 2004. A weight-based map matching method in moving objects databases. In: Proceedings of the 16th Scientific and Statistical Database Management (SSDBM).

Georgios Kellaris received his BSc degree in Informatics and Telecommunications from the University of Athens, and his MSc degree in Digital Systems from the University of Piraeus. From September 2008 until November 2009, he was a Research Associate at the Information Systems Laboratory, University of Piraeus, and on November 2009 until July 2010, he worked as a Research Fellow at the School of Information Systems, Singapore Management University (SMU). Currently, he is a PhD candidate at the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His research interests include spatio-temporal data management and privacy preserving data mining.

Nikos Pelekis is a Lecturer at the Department of Statistics and Insurance Science, University of Piraeus, Greece. Born in 1975, he received his BSc degree from the Computer Science Department of the University of Crete (1998). He has subsequently joined the Department of Computation in the University of Manchester Institute of Science and Technology (UMIST) to pursue his MSc in Information Systems Engineering (1999) and his PhD in Moving Object Databases (2002). His research interests include data mining, spatiotemporal databases, management of location-based services, machine learning and geographical information systems, whereas he teaches respective courses at under- and post- graduate level. He has been particularly working for almost ten years in the field of Mobility Data Management and Mining, being the architect of the widely cited “Hermes” Moving Object Database (MOD) engine.He has offered several invited lectures in Greece and abroad (including PhD/MSc/summer courses at Rhodes, Milano, KAUST, Aalborg and Trento) on Mobility Data Management and Data Mining topics. He has co-authored more than 50 research papers and book chapters, while he is a reviewer in many international journals and conferences. He has served as co-Organizer of a EURO Stream on data mining and knowledge discovery (DMKD@EURO'09). He has been member of the Organizing Committee for ECML/PKDD 2011. He has been actively involved in more than 10 European and National R&D projects. Among them he is or was principal researcher in GeoPKDD (FP6/IST, 2005-09), MODAP (FP7/ICT, 2009-12), MOVE (COST, 2009-13; substitute member of the mgmt committee), DATASIM (FP7/ICT, 2011-14) and SEEK (FP7/PEOPLE, 2012-15). For more information: http://infolab.cs.unipi.gr/people/npelekis.

Prof. Yannis Theodoridis is faculty member at the Department of Informatics, University of Piraeus, where he currently leads the Information Management Lab. Born in 1967, he received his Diploma (1990) and PhD (1996) in Electrical and Computer Engineering, both from the National Technical University of Athens, Greece. His research interests include Data Science (management, analysis, mining) for mobility data, whereas he teaches databases, data mining and GIS at under- and post- graduate level. He is or was principal investigator for a number of EU-funded research projects (with FP7/MODAP, COST/MOVE, FP7/DATASIM and FP7/SEEK being the most recent). He has served as general co-chair for SSTD'03, ECML/PKDD'11 and PCI'12, vice PC chair for IEEE ICDM'08, member of the editorial board of the Int'l Journal on Data Warehousing and Mining ñ IJDWM (since 2005), and member of the SSTD endowment (since 2010). He has delivered invited lectures in Greece and abroad (including PhD/MSc- level seminars at Venice, Milano, KAUST, Aalborg, Trento, Ghent and Cyprus) on the topic of Mobility Data Management and Exploration. He has co-authored three monographs and more than 100 refereed articles in scientific journals and conferences, receiving more than 800 citations. For more information: http://www.unipi.gr/faculty/ytheod and http://infolab.cs.unipi.gr.