Trajectory Compression under Network constraints

Georgios Kellaris
University of Piraeus, Greece
Phone: (+30) 6942659820
user83@tellas.gr

1. Introduction

The trajectory of a moving object can be described as a set of triplets which have the form <*x*, *y*, *t*>, where (*x*, *y*) is the geographic location of the object, at time *t*. The preservation of many trajectories for future reference has raised the need of their compression. Furthermore, the common way to retrieve trajectory data of an object is by using a GPS receiver. According to [8], data points received from a GPS receiver have an error that ranges from 2 to 8 meters. Thus, there arises the problem of matching these data points onto a road network, also known as the map-matching problem [1].

The existing work tries to solve the problem of trajectory compression for either online or offline data. Online algorithms compress the trajectory data while new points are received as opposed to offline algorithms, where data points are known in advance. Offline algorithms are presented in works [5] and [7]. In work [7], Meratnia and de By also suggest an algorithm for online data. Two algorithms for online data compression are also suggested in work [9]. For the map-matching problem, Brakatsoulas et al. [1] suggest an algorithm that works exclusively for offline data.

Although there are suggestions for the compression and the map matching problem, there are not any suggestions regarding the need of data compression and the preservation of the matching onto the network at the same time, as mentioned in work [6]. The existing compression algorithms do not take into account the network constraints and map matching algorithms do not offer the capability of compression.

In this research, we suggest methods for vehicle data compression under network constraints by using existing compression and map-matching algorithms serially. In addition, we propose a completely new algorithm regarding the compression of trajectories already matched onto a network, assessing at the same time the network constraints.

The paper is organized as follows. Section 2 reviews some of the current work in trajectory compression, trajectory map matching and additionally trajectory similarity search, since we need to compare the results of the proposed methods. Section 3 presents our suggestions on compression under network constraints and our proposed algorithm. Section 4 evaluates the results from the implementations of our methods. Section 5 concludes this work.

2. Background

In this section we represent the following methods of compression. The methods suggested by Douglas-Peckeur [5] and Meratnia and de By [7] concerning offline data and the methods Opening Window (OW) suggested by Meratnia and de By [7], Thresholds and STTrace suggested by Potamias et al. [9], that refers to online data. For the map-matching problem, we represent the algorithm of Brakatsoulas et al. [1], which works for offline data only. We also represent the algorithm of E. Tiakas et al. [10] for trajectory similarity search over the network.

2.1 Trajectory Compression

The first algorithm we represent for trajectory compression is the well known Douglas-Peckeur [5] algorithm as shown in Figure 1.

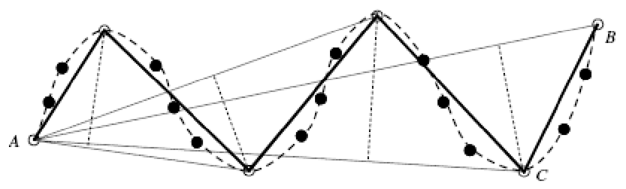


Figure 1 - Douglas-Peckeur algorithm [6]

This algorithm works as follows. First, it calculates the distance of every internal point from the line connecting the first and the last point of our trajectory (line AB in Figure 1). Then, it finds the point with the greatest distance from the line (point C in Figure 1) and creates the lines AC and CB. The algorithm then recursively checks the distance of each remaining point from the new lines following the same technique. The algorithm returns the set of these lines when no distance is greater than a predefined threshold. The new poly-line has fewer points than the original.

Meratnia and de By [7] uses the Douglas-Peckeur [5] algorithm to create a new compression technique that also takes into account the parameter of time. The suggested algorithm replaces the distance used in Douglas-Peckeur [5] algorithm with a new one, called Synchronous Euclidean Distance (SED), which takes into account the time dimension as shown in Figure 2.

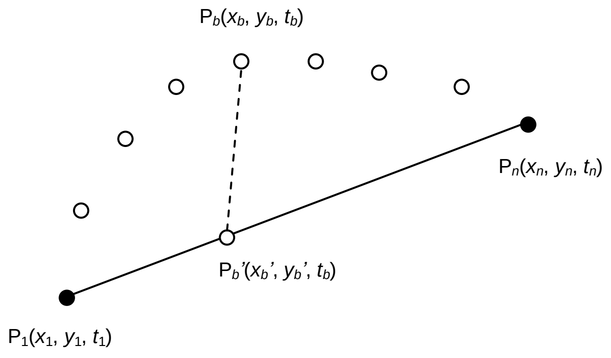


Figure 2 - SED

Let P_b be the currently examined point against the line P_1P_n as shown in Figure 2. The coordinates of point P_b' are calculated using SED as follows.

$$x_b' = x_1 + \frac{\Delta b}{\Delta n}(x_n - x_1)$$
$$y_b' = y_1 + \frac{\Delta b}{\Delta n}(y_n - y_1)$$

Where $\Delta n = t_n - t_1$ and $\Delta b = t_b - t_1$.

The described algorithm needs the a priori knowledge of the trajectory. Thus, in the same work [7], Meratnia and de By present a version of the algorithm called Opening Window (OW) that can be applied to online data. 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 segment's end point 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 current segment's end point and the start point of the new one. As long as new points arrive, the method continues as described.

Alternatively, Potamias et al. [9] suggest two algorithms, Thresholds and STTrace, 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. 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, Potamias et al. suggest a combination of the two alternatives. Both areas are calculated, but only their intersection is defined as the safe area.

The difference between Thresholds and STTrace algorithm lies to the fact that the latter offers the capability of removing previously chosen points. Let *M* be the maximum number of points that can be stored. For every point being chosen, the SED from that to the line segment formed by its previous and its following point is calculated. This calculation, concerning every chosen point, occurs when every next point is added. When a new point is chosen and the number of stored points is equal to *M*, an old point must be removed or the new one must be ignored. The choice is made as follows. Initially the stored point with the smallest SED is found. Then, the SED of the last two stored points and the new one is calculated. The point that returned the smallest SED value is removed / ignored. The removal of that point offers the least possible loss of information.

2.2 Map-Matching

Now, as far as the map-matching problem is concerned, Brakatsoulas et al. [1] suggest the following algorithm for offline trajectory data. Initially, the algorithm finds the nearest edge to the first point of the trajectory. Then, for every point P_i, given that the point P_{i-1} has already been matched, we evaluate the candidate edges to be matched to P_i as shown in Figure 3. P_{i-1} is matched to edge c_3 and, hence, c_1 and c_2 are the candidate edges for the point P_i.

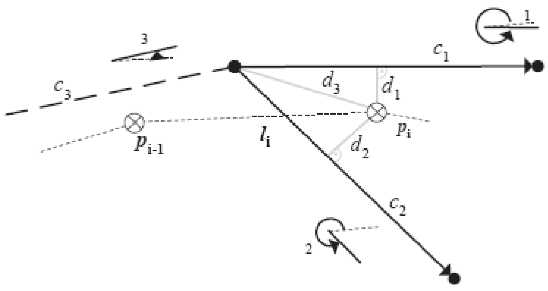


Figure 3 - Map-matching algorithm [1]

The following two measures are used for the edge evaluation:

$$s_d(p_i, c_j) = \mu_d - a \cdot d(p_i, c_j)^{nd}$$

And

$$s_a(p_i, c_j) = \mu_a \cdot \cos(a_i, a_j)^{na}$$

Where μ_d, μ_a, n_a, n_d and a are predefined scaling factors as described in work [1]. The measure s_d denotes the distance and the measure s_a denotes the orientation. The higher the sum s of these measures, the better the match to this edge is.

If the projection of the current point on the candidate edges is not 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 is evaluated.

In order to improve the functionality of the algorithm, Brakatsoulas et al. [1] suggest the use of a "look ahead" policy. That is, the total score of each candidate edge is calculated by adding the scores of a fixed number of edges, which are ahead of the current position, to the initial one.

Figure 4 shows an example of how the "look ahead" policy is used, regarding the point P_i. For candidate edge c_2 of this point, edge $c_{2,1}$ is the best candidate for the point p_{i+1}, while for candidate edge c_1, edge $c_{1,1}$ is the best candidate for point p_{i+1}. The final score for each of the edges c_1 and c_2 is the sum of the scores of each best sub-path.

The map-matching algorithm of Brakatsoulas et al. [1] does not take into account the time dimension and also it does not work for online data. However, if we set the "look ahead" edges equal to 0 (no "look ahead" check occurs), the algorithm could be used for online trajectory data.

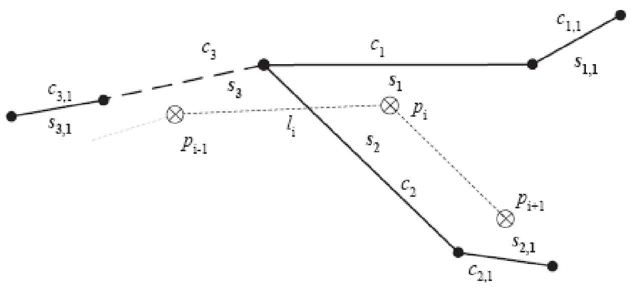


Figure 4 - Look Ahead [1]

2.3 Trajectory Similarity on Network

Traditional methods for trajectory similarity search [6] cannot be used for trajectories over the network. Their results would not be accurate. Thus, Tiakas et al. [10] present a method for trajectory similarity search under network constraints.

Let $c(u_i, u_j)$ be the cost to travel from node u_i to node u_j. The network distance $d(u_i, u_j)$ between these two nodes is defined as:

$$d(u_i,u_j) = \begin{cases} 0, c(u_i,u_j) = 0 \ \cap c(u_i,u_j) = 0 \\ \dfrac{\min\{c(u_i,u_j), c(u_j,u_i)\}}{\max\{c(u_i,u_j), c(u_j,u_i)\}}, otherwise \end{cases}$$

The network distance between two trajectories T_a and T_b is then defined as:

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

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

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

Where *T[i].t* is defined as the temporal data value of the *i*-th point of the trajectory.

Both measures satisfy the metric space properties as proven in [10]. By combining the two measures, the total similarity can be expressed as follows:

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

Where W_{net} and W_{time} is predefined weight factors.

In the above analysis, the length of each trajectory was considered to be equal to one another. In the general case, where the lengths of the two trajectories might not be the same, every trajectory is decomposed to sub-trajectories as follows. Given a trajectory T of length m and an integer $\mu<m$, m-μ+1 sub-trajectories are created by using a window of length μ, which is moved progressively by one node at a time. Figure 5 shows an example of de-composing a trajectory T of length m=6 into 4 sub-trajectories S_i, each one of them of length μ=3. In case m is less than μ for a given trajectory T, then the last point of T is repeated until the length of T becomes equal to μ.

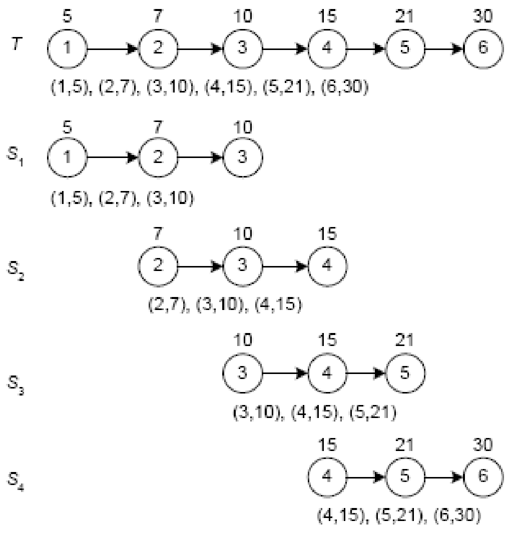


Figure 5 - Trajectory decomposition [10]

The total distance $D_{total}(T_a, T_b)$ is calculated by combining the D_{total} of every sub-trajectory as described in [10].

3. Suggested methodology for network trajectory compression

As stated in previous section, no solution exists to the problem of trajectory compression over the network. Map-matching algorithms can offer a form of compression, but they are not designed for data compression and they cannot compress trajectories that have already been matched onto a network. In this section we propose methods that possibly could offer trajectory compression under network constraints.

By combining the algorithms of Meratnia and de By [7] for online and offline data compression and the map-matching algorithm of Brakatsoulas et al. [1], we create two alternatives as follows.

a) We apply the compression algorithm (Comp) to the original un-matched trajectory data and then we apply the map-matching algorithm (MM) to the output. We call this method Comp + MM.
b) We apply the map-matching algorithm (MM) to the original un-matched data; then we apply the compression algorithm (Comp) to the output and finally we re-apply the map-matching algorithm (MM) to the new output. We call this method MM + Comp + MM.

In case of online data, we use the OW algorithm for compression and the map-matching algorithm of Brakatsoulas et al. [1] with no “look ahead” function.

3.1 Issues of the serial use of the algorithms

The map-matching algorithm of Brakatsoulas et al. [1] does not retain temporal data. This information is critical concerning the proper function of the compression algorithm (in order to calculate the SED), and it is also important to us, in order to maintain useful trajectory data.

The map-matching algorithm needs to be reformed in order to preserve the temporal information. For every edge that is being added to the final result by the algorithm, the time on the corresponding nodes needs to be approximated. In order to achieve that, we need to keep the information of the first and the last point projection on every edge, if any. Having this information, we can then proceed as described below.

We consider two point projections P_1 (the last point projection on the corresponding edge) and P_2 (the first point projection on the corresponding edge) on two neighboring edges with P_3 as their common node, as it is depicted in Figure 6.

We can calculate the time t_3 on node P_3, presuming the object was moving at constant speed, as following.

$$t_3 = \frac{d1 \cdot t_2 + d2 \cdot t_1}{d1 + d2}$$

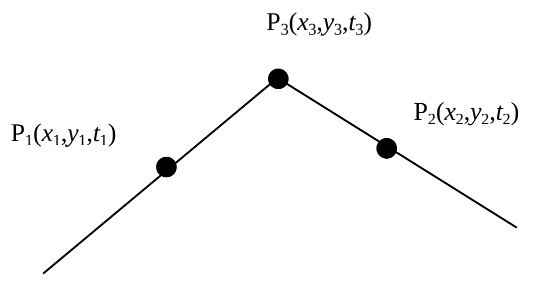


Figure 6 - Node time calculation

Where $d1 = \sqrt{(x_3 - x_1)^2 + (y_3 - y_1)^2}$ is the distance between P_1 and P_3 and $d2 = \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2}$ is the distance between P_2 and P_3.

In the general case where the points P_1 and P_3 might not be on neighboring edges, we define as distance d the network distance from the point to the given node. As network distance we define the distance form the point to the nearest node plus the lengths of all the edges needed to be travelled from the edge of the point to reach the given node. Let P_1 and P_n be the two points and P_r the node with the temporal info that needs to be calculated as shown in Figure 7.

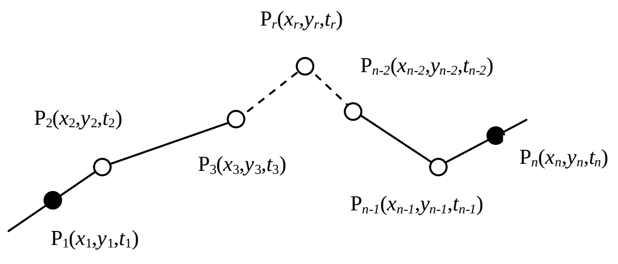


Figure 7 - Node time clalculation - general case

The time t_r on node P_r is calculated as:

$$t_r = \frac{d1 \cdot t_n + d2 \cdot t_1}{d1 + d2}$$

Where $d1 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} + \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2} + \cdots + \sqrt{(x_r - x_{r-1})^2 + (y_r - y_{r-1})^2}$

And $d2 = \sqrt{(x_{r+1} - x_r)^2 + (y_{r+1} - y_r)^2} + \cdots + \sqrt{(x_{n-2} - x_{n-1})^2 + (y_{n-2} - y_{n-1})^2} + \sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2}$.

3.2 A new approach to the problem of trajectory compression over the network

A new compression method could be introduced by altering some paths of the given trajectory with shorter ones. This could be achieved by executing a shortest path algorithm on specific points of the trajectory.

We can define the maximum compression this algorithm could achieve. Since it uses the shortest path algorithm on some sub-paths of our trajectory, the maximum achievable compression is the result of the shortest path algorithm executed on the first and the last point of the trajectory. By taking this into account, we present the suggested algorithm in the next paragraphs.

The algorithm works as shown in Figure 8. Given a trajectory over the network consisted of seven points $\{P_1, P_2, P_3, P_4, P_5, P_6, P_7\}$, we can calculate the maximum achievable compression which is $\{P_1, a_3, P_7\}$ as shown in Figure 8(c).

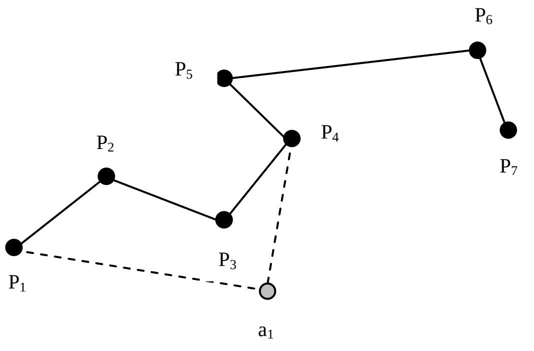
During the first pass of the algorithm the first point of the trajectory is checked against its subsequent points, until a shortest path between a point and the initial one is found as shown in Figure 8(a). There is no shortest path between P_1 and P_2, since they are contiguous points. There would be a shortest path between P_1 and P_3 if they were connected directly. We suppose that they are not directly connected and there is a shortest path connecting P_1 and P_4 through the node $a1$. Then we can change the sub-path $\{P_1, P_2, P_3, P_4\}$ of the trajectory to $\{P_1, a_1, P_4\}$. Thus, our trajectory has become shorter by one point out of seven.

We also need to calculate the time at which the object would have been at node a_1. This can be estimated by using the temporal data on nodes P_1 and P_4 and by considering that the object was moving at constant speed. This method is similar to the one described in section 3.1.

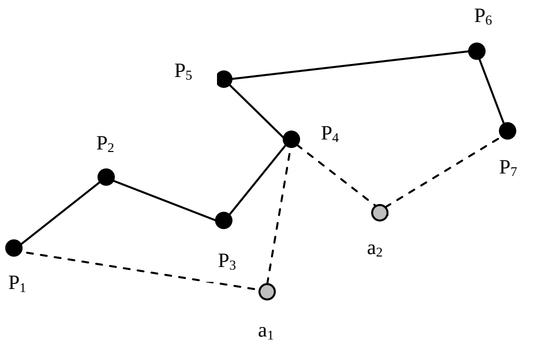
When a shortest path is found, the algorithm stores this result and continues the pass by defining the last node of the found shortest path as first point and by checking the contiguous points to this node. In our case, we check P_4 against its next points.

Finally, a shortest path is found connecting P_4 and P_7 through a_2 as shown in Figure 8(b). Since P_7 is the last point of the trajectory, the first pass of the algorithm ends and it returns the compressed trajectory $\{P_1, a_1, P_4, a_2, P_7\}$.

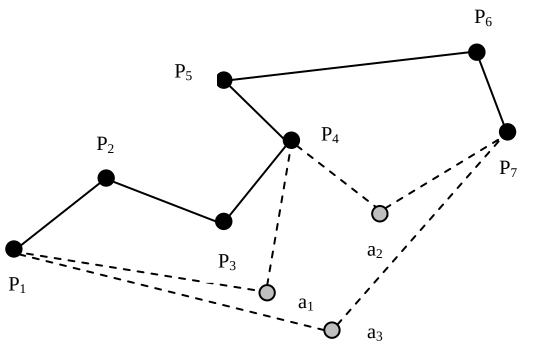
In case we need a more compressed result, the algorithm continues to the second pass, where the same technique we saw above is applied to the remaining points of our original trajectory. In our case these points are P_1, P_4 and P_7. There is a shortest path connecting P_1 and P_7 through a_3 as shown in Figure 8(c). So, the new result is $\{P_1, a_3, P_7\}$, which is also the maximum achievable compression and, hence, the algorithm stops.



(a)



(b)



(c)

Figure 8- Execution example of the algorithm: (a) first pass, (b) continue of the first pass and (c) second pass

Since the algorithm only needs the shortest paths from the original nodes, we can pre-calculate these shortest paths in order to avoid duplicate calculations. The algorithm is shown in Figure 9. It works for offline data.

```
Algorithm MapComp (T, max)
Input
T: the given trajectory
max: the maximum compression we want to/can achieve

1. Pre-calculate all the shortest paths from the nodes of T
2. Set compression achieved c=0
3. While c<max
   a) Set i=1
   b) Set the i-th node of T as the beginning node N
   c) While i<T.length
      i)  i=i+1
      ii) If there is a shortest path between N and the i-th
          node of T
          (1) Calculate the time values of the new nodes
          (2) Replace the sub-path from N to the i-th node
              with their shortest path
          (3) Set the i-th node of T as the beginning node
              N
   d) Calculate the achieved compression c
```

Figure 9 - MapComp algorithm

4. Evaluation

In order to evaluate the methods we proposed, we need trajectories of objects onto which the algorithms will be applied. Given a network map, we need an algorithm that creates random trajectories. We also need an algorithm that adds noise to these trajectories in order to have a more realistic result, as well as an algorithm, like the one proposed by Tiakas et al. [10], for comparing the outputs.

4.1 Methodology

We use the following methodology in order to produce our results.

1. A trajectory T is created on the network
2. Random noise is added to the trajectory T in order to produce the trajectory T'
3. We generate the compressed trajectory T'' on the network using the proposed methods on the trajectory T':
   i) Comp + MM
   ii) MM+ Comp + MM and
   iii) MM for the comparison of the results
4. We compare the resultant trajectory T'' with the trajectory T against the compression achieved and the time needed to be executed. We also compare

T'' with the map-matched version of *T'* against their resemblance.

In order to produce trajectories over the network, we use the Brinkhoff's Generator [3] as described in work [2]. The result needs to be realistic, so we add Gaussian noise to the coordinates of every point with mean 0 and deviation 4, as stated to be the typical noise values for a GPS receiver by Pfoser and Jensen [8]. Figure 10 shows our suggested methodology.

The above methodology is applied to both online and offline data in order to achieve a more thorough perspective of the results. We also apply our suggested algorithm to the map-matched version of trajectory *T'*.

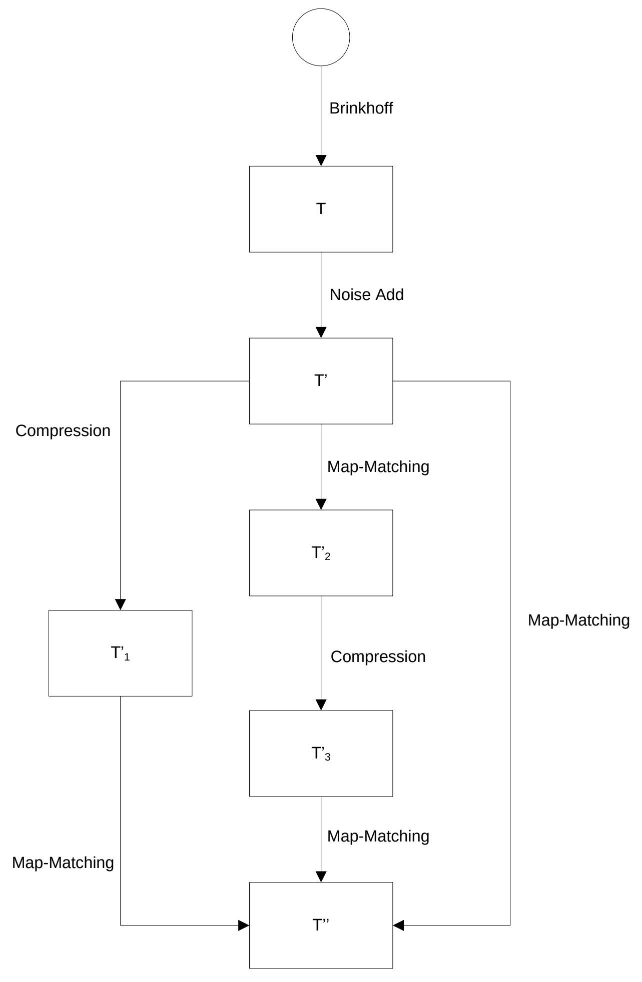


Figure 10 - Evaluation Methodology

4.2 Map-matching algorithm issues

Concerning the map-matching algorithm, in order to calculate the orientation score for every edge, Brakatsoulas et al. [1] use the cosine of the angle formed by the line segment of the last two points and the candidate edge. In particul, the formula is as follows.

$$s_a(p_i, c_j) = \mu_a \cdot \cos(a_i, a_j)^{na}$$

In their work the value of the parameter n_a is set to 4. However, if the cosine is powered by an even number, we get a positive result regardless the sign of the angle's cosine. This could cause the algorithm to malfunction in a case such as the one illustrated in Figure 11.

In this case, the algorithm chooses edge s_1 for the point P_1. When it proceeds to the 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, orientation score will determine which edge will be selected. Normally, edge s_3 should achieve better score than s_2, since $\theta_2<\theta_1$ and hence $cos\theta_2>cos\theta_1$. However, by using absolute values, we get $|cos\theta_2|<|cos\theta_1|$ and the edge s_2 is selected instead of s_3, provided that no "look ahead" check is performed. In order to avoid this, we use the odd number 3 for the parameter n_a in our implementation, instead of the even number 4.

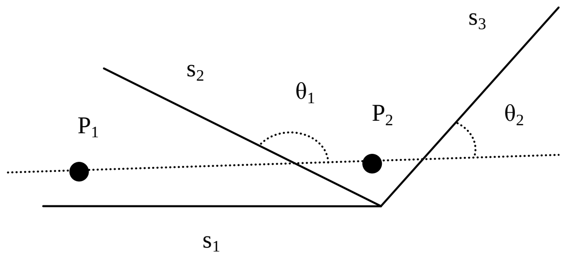


Figure 11 - Case scenario of wrong edge selection

4.3 Similarity search algorithm issues

The algorithm that Tiakas et al. [10] suggested for trajectory similarity search cannot be used in our case scenario. In case the graph of the network map is not directed, the cost to travel from node u_i to node u_j is the same with the cost to travel from node u_j to node u_i. Thus, if $u_i=u_j$, the proposed distance $d(u_i, u_j)$ returns the value 0, otherwise it returns the value 1. This measure is not accurate for comparing trajectories, especially in the case where one of the two trajectories, which are under comparison, is the shifted result of the other.

Therefore, we propose the following solution. The cost $c(u_i, u_j)$ is defined as the distance to travel from u_i to u_j using their shortest path as proposed in [10]. In order to calculate this cost we use Dijkstra's shortest path algorithm [4]. Then, we need to define an alternative to the distance d between two nodes. In order to get results scaled from 0 to 1, we need a function with co-domain values $0\leq d(u_i, u_j)<1$ for domain values $0\leq c(u_i, u_j)<\infty$.

For this purpose, we can use the function *arctan*(*x*) which is shown in Figure 12.

We notice that for domain values less than 2, there is a highly increasing rate of co-domain values as opposed to domain values greater than 2. This is a useful attribute of the function, because we need more accurate results for trajectories that are closer to one another, than for long-distant ones.

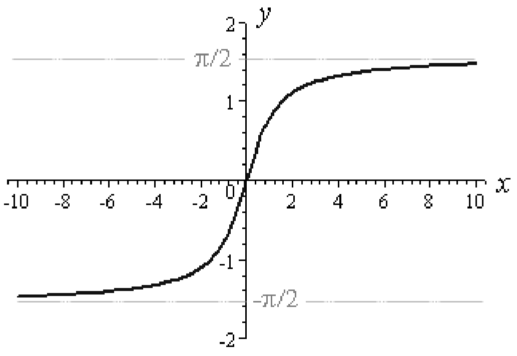


Figure 12 - Inverse Tangent Function's Graph

Additionally, the value of 1 is needed instead of $\pi/2$ as an upper bound, so we can define the following measure as distance between the nodes u_i and u_j:

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

The network distance between two trajectories T_a and T_b of length *m* can then be defined as in work [10]:

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

Respectively, the time distance between the trajectories T_a and T_b can be defined as:

$$D_{time}(T_a, T_b) = \frac{1}{m-1} \cdot \frac{2}{\pi} \cdot \sum_{i=1}^{m-1} \arctan\left(\left|(T_a[i+1].t - T_a[i].t) - (T_b[i+1].t - T_b[i].t)\right|\right)$$

Where *T[i].t* is defined as the temporal data value of the *i*-th point of the trajectory.

The total similarity can be expressed as in work [10]:

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

Since the noise-add method does not affect time values, we can assume that the time distance depends on the network distance. Thus, we can set W_{net}=W_{time}=0.5 since both distances are considered to be equally significant.

In case the two trajectories have not the same length, we use the same technique mentioned in [10] and we set the length μ of the sub-trajectories equal to the length of the 'shortest' initial trajectory. All the sub-trajectories are evaluated and we use the average score of these evaluations as total similarity measure.

4.4 Implementation

All algorithms have been implemented in Java. The Brinkhoff's Generator [3] has been executed on the default map *oldenburgGen*. The values used for the parameters of the map-matching algorithm are μ_d=10, n_d=1.4, μ_a=10 and *a*=0.17 as used in work [1] except for the value of the parameter n_a, since the latter is set to 3 as explained in section 4.2.

We have produced three trajectories using the Brinkhoff's Generator for different *max. speed div.* values. This parameter defines the speed of the moving object. Since the sampling rate is constant, *max. speed div.* actually defines the density of trajectory points. We evaluate our results against the parameter *look* that defines the number of the look-ahead edges concerning the map-matching algorithm, against the parameter *tol* that defines the threshold for the compression algorithm and finally against the parameter *max. speed div.*. The values of the parameters are shown in Table 1, where the default ones are underlined.

In case of online data, we use the OW algorithm and the map-matching algorithm with *look*=0. We consider the points of every trajectory as not known in advance in order to simulate the online data generation.

Table 1 – Parameters' values

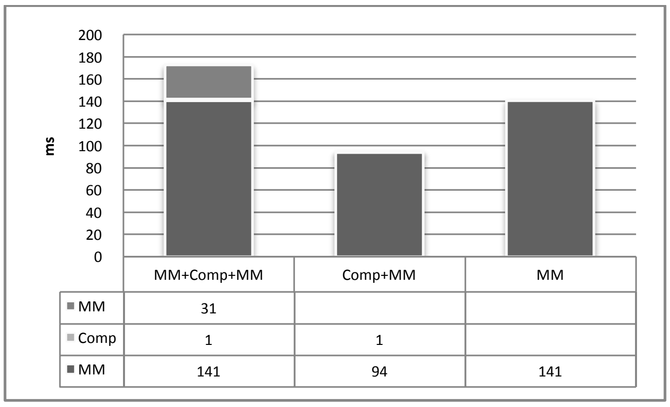
Parameter	Value		
Max. speed div.	75	150	<u>250</u>
look	2	<u>4</u>	6
tol	<u>10</u>	50	90

Regarding our proposed solution, the algorithm has been tested with the trajectory produced by Brinkhoff's Generator [3] with *max. speed div.* =150. In order the result to be on the network, we use the map-matching algorithm[1] with *look*=4. Dijkstra's algorithm [4] is used for the discovery of the shortest paths. Eventually, we compare the results to the map-matched version of the original trajectory.

4.5 Results

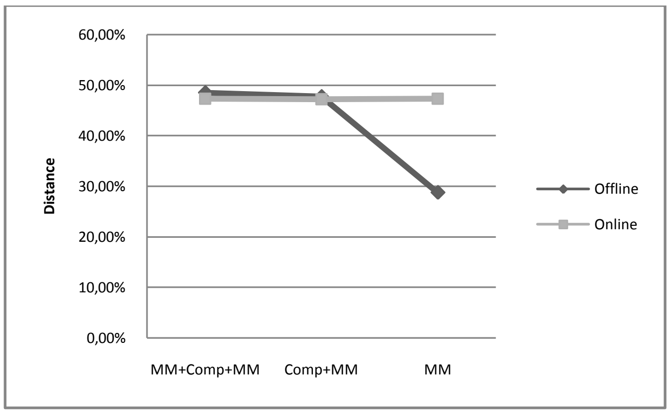
From the analysis of the results, it is obvious that method Comp + MM offers similar compression and distance results compared to method MM + Comp + MM. The double execution of the time-consuming map-matching algorithm in method MM + Comp + MM has resulted in two times slower execution speed compared to method Comp + MM as shown in Figure 13. Comp + MM executes faster than MM, since map-matching algorithm is applied to fewer points because of the pre-executed compression algorithm. However, as far as quality is concerned, methods MM + Comp + MM and Comp + MM have produced worst results than MM regarding offline data and similar results regarding online data, as shown in Figure 14(a). The compression, as compared to the original trajectory before the map-matching, is almost the same for all the methods as shown in Figure 14(b).

Regarding execution speed performance, the execution time of the compression algorithm is insignificant when compared to the execution time of the map-matching algorithm. Furthermore, the execution time of the map-matching algorithm time seems to be dependant on the number of points, presuming we use the "look ahead" function as shown in Figure 15(a). In case of online data, the number of points does not affect the execution time of the map-matching algorithm as shown in Figure 15(b). The execution time is also proportional to the number of edges that "look ahead" checks as shown in Figure 16. Regarding the compression algorithm in methods MM + Comp + MM and Comp + MM, greater *tol* values offer better execution speed and compression but worse quality, as it was expected.

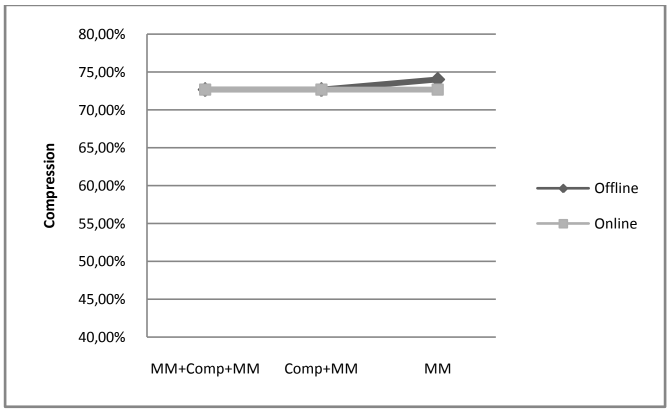


	MM+Comp+MM	Comp+MM	MM
■ MM	31		
■ Comp	1	1	
■ MM	141	94	141

Figure 13 - Execution time of each method for offline data



(a)

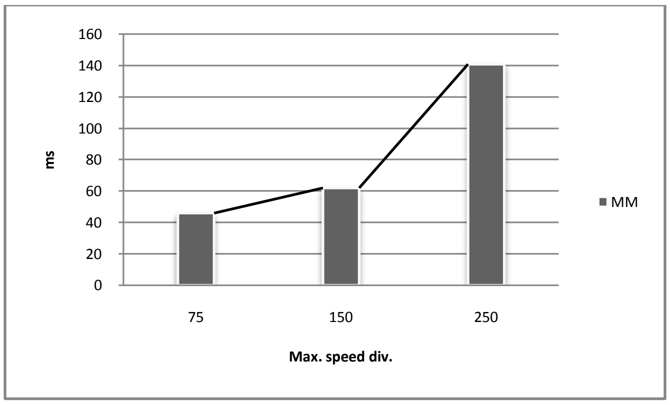


(b)

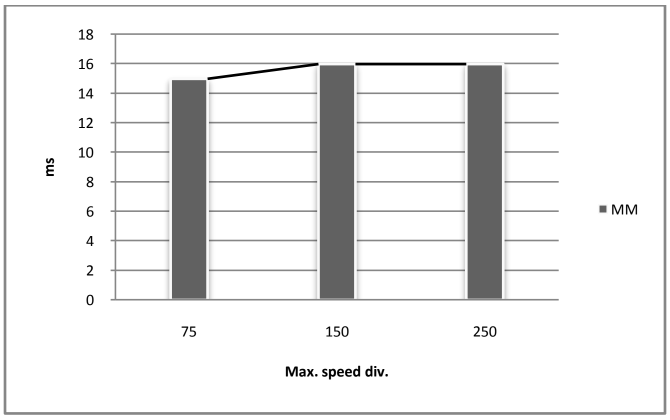
Figure 14 – (a) Distance and (b) compression of each method for offline and online data

According to the results, greater *look* values do not offer better quality for methods Comp + MM and MM as shown in Figure 17(a). They do, however, affect the quality of method MM + Comp + MM, because of the double execution of the map-matching algorithm. In addition, the value of the *look* parameter does not affect significantly the compression as shown in Figure 17(b).

Regarding our proposed method, the first pass has offered a compression which is 14.7% better than the compression achieved by the simple execution of the map-matching algorithm. The second pass has achieved an even better compression by 20.5%. Concerning quality, the first pass has resulted in a distance of 53% compared to the original map-matched trajectory while the second pass has resulted in a distance of 53.6%.



(a)



(b)

Figure 15 - Execution time of map-matching for different values of *max. speed div.* in case of (a) offline and (b) online data

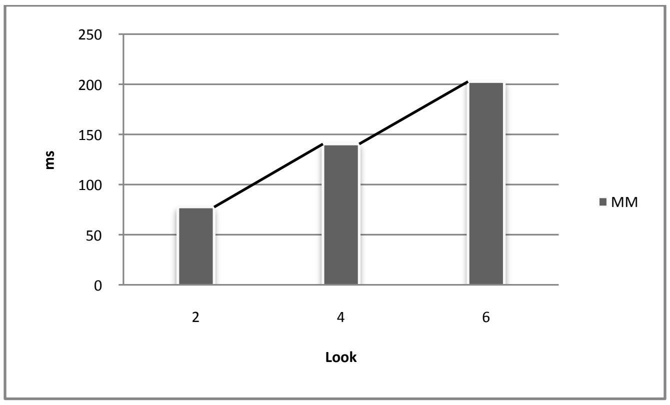
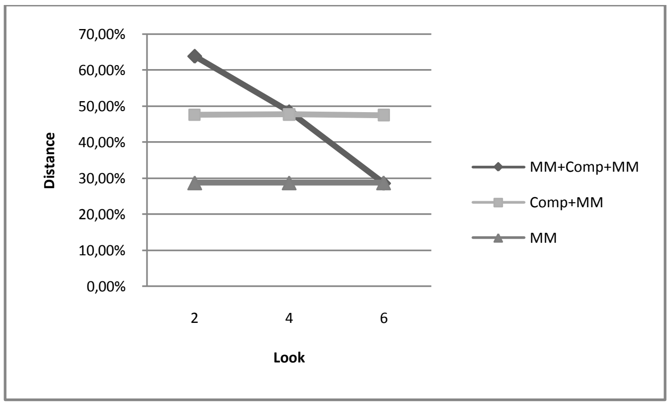
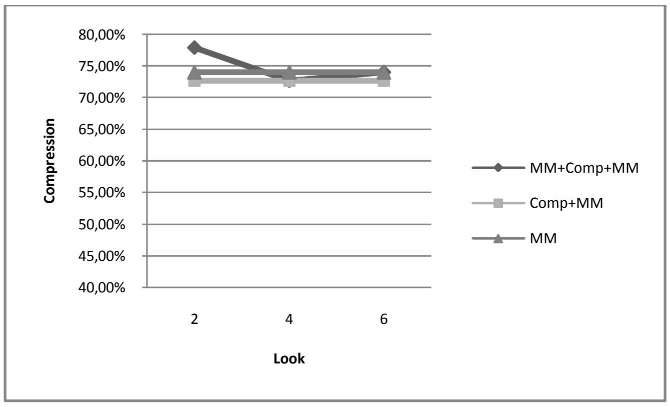


Figure 16 - Execution time of map-matching for different values of *look* parameter

The algorithm is time-consuming, since it needs to pre-run the shortest path algorithm as many times as the points of the map-matched trajectory are. In addition, the total amount of the map nodes has great impact to the execution time of the shortest path algorithm. In order to use the algorithm for online data, large sampling rate would be needed, since the shortest path algorithm would need to be executed for every new arriving node.



(a)



(b)

Figure 17 – (a) Distance and (b) compression of each method for different *look* values

In our case, for a map of 7443 nodes and 34 trajectory points, the algorithm needed 12 seconds to run. The minimum frequency of incoming points for online data would be in this case $12sec/34points=0.35sec$, presuming that every new point is matched to a new edge.

5. Concluding Remarks and Future Work

Current research offers solutions to the problems of trajectory compression and map-matching succinctly, but none satisfies the need of trajectory compression under network constrains. A map-matching algorithm can offer a certain amount of compression. Moreover, this can be achieved within a small period of time by combining the map-matching algorithm with a compression one. However, the simpler trajectory resulting from this algorithm is not always a compressed one or the best compressed result that can be achieved. Our experimental results support this claim. Thus, the need of a new method arises.

In this work we have proposed a completely new method for trajectory compression under network constraints accompanied with satisfying results regarding compression. However, it is time consuming due to the use of the shortest path algorithm. In future work, other shortest path algorithms could be used alternatively to Dijkstra's algorithm [4] for better execution performance. Better results regarding execution time could also be achieved by reducing map points. This would be possible by removing map points that have a great distance from the given trajectory.

Alternatively, a completely new approach can be adopted. We can compress our data by removing every edge of the given trajectory that can be easily predicted. To be more specific, an edge can be removed if it can be discovered from its precedent, using an algorithm like Thresholds or STTrace [9] or if it is the only neighboring edge to its previous. We can also remove contiguous edges of a sub-path which is the unique shortest path between two points of the trajectory. The result of this algorithm would not be a trajectory over a network, but just a compressed file. A decompressing algorithm would be needed in order to retrieve the original trajectory data. This solution could offer lossless compression regarding the position of the object. However, the temporal data values of the removed nodes would not be able to be retained, but it would be possible to be approximated.

References

[1] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On Map-Matching Vehicle Tracking Data. Proceedings of the 31st international conference on Very Large Data Bases (VLDB), 2005.

[2] T. Brinkhoff. Generating Network-Based Moving Objects. Proceedings of the 12th International Conference on Scientific and Statistical Database Management (SSDBM), 2000.

[3] T. Brinkhoff. Road network generator url: http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, Vol 1(1), pp. 269-271, 1959.

[5] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. The Canadian Cartographer Vol. 10(2), pp. 112-122, 1973.

[6] E. Frentzos, N. Pelekis, I. Ntoutsi, and Y. Theodoridis. 6th Chapter in "Mobility, Data Mining and Privacy", pp. 151-187, 2008.

[7] N. Meratnia and R. A. de By. Spatiotemporal Compression Techniques for Moving Point Objects. Proceedings of the Extending Database Technology (EDBT), 2004.

[8] D Pfoser and CS Jensen. Capturing the Uncertainty of Moving-Object Representations. Proceedings of the 6th International Symposium on Spatial Databases (SSD), 1999.

[9] M. Potamias, K. Patroumpas, and T. Sellis. Sampling Trajectory Streams with Spatiotemporal Criteria. Proceedings of the Scientific and Statistical Database Management (SSDBM), 2006.

[10] E. Tiakas, A. N. Papadopoulos, A. Nanopoulos, and Y. Manolopoulos. Trajectory Similarity Search in Spatial Networks. Proceedings of the 10th International Database Engineering and Applications Symposium (IDEAS), 2006.