Clustering Vessel Trajectories with Alignment Kernels under Trajectory Compression

Gerben de Vries and Maarten van Someren

Informatics Institute, University of Amsterdam, Sciencepark 107, 1098 XG Amsterdam, The Netherlands
{G.K.D.deVries,M.W.vanSomeren}@uva.nl

Abstract. In this paper we apply a selection of alignment measures, such as dynamic time warping and edit distance, to the problem of clustering vessel trajectories. Vessel trajectories are an example of moving object trajectories, which have recently become an important research topic. The alignment measures are defined as kernels to allow for use in the kernel k-means clustering method. Furthermore, we investigate the performance of these alignment kernels in combination with a trajectory compression method. Experiments on a gold standard dataset indicate that compression has a positive effect on clustering performance for a number of alignment measures. Also, soft-max kernels, based on summing all alignments, perform worse than classic kernels, based on taking the score of the best alignment.

1 Introduction

Largely due to the ubiquity of GPS receivers, moving object trajectories have become an important research topic. To gain insight into the behavior of moving objects, such as vessels, it can be useful to cluster their trajectories into groups of similar movement patterns. Essentially, moving object trajectories are a kind of multivariate time-series. Furthermore, they have the property that they are usually different in length, both in time, distance travelled and in the number of data points. Alignment methods, such as dynamic time warping or edit distance, are designed to handle the kinds of variations. Thus such methods can make a suitable similarity measure for clustering moving object trajectories.

These alignment based similarities are not necessarily proper metrics, i.e. the triangle inequality does not always hold. Since the standard, and very popular, k-means clustering method requires such a metric, we cannot use it. However, one can use more recent clustering methods that are more flexible in this sense, such as kernel k-means [16] and spectral clustering [14], between which there exists a close connection [4]. These methods also have the advantage that they can create clusters from objects that are non-linearly seperable in input space. Another adavantage of considering kernel k-means is that the defined kernels can potentially be used in other kernel based machine learning settings.

Computing these alignment kernels can be expensive, because of the number of elements in a trajectory. However, these trajectories have the property that they are very regular. Without losing much information they can be compressed very well with trajectory compression techniques such as [8]. Computing an alignment between two compressed trajectories can be a lot cheaper, because of the reduction in data points. But, the quality of the alignment does not necessarily have to be the same. Compression loses information and may have a bad effect on the alignment.

In this paper we define different alignment kernels for moving object trajectories. The kernels are based on a number of well-known alignment measures. We consider two kernel versions of these measures, a classic and a soft-max one. Not all of these kernels are proper kernels, since they are not positive semi-definite. However, for simplicity, we refer to all of them as kernels. Furthermore, all kernels can be used with the kernel k-means clustering algorithm that we use in our vessel trajectory clustering task. In this task the kernels are computed on uncompressed and compressed trajectories. A number of the alignment kernels show better performance on the task when the trajectories are compressed first.

The rest of the paper is organized as follows. In Sect. 2 we discuss related work in the fields of trajectory clustering and kernel methods for time-series. Section 3 contains the technical definitions: three kinds of alignments, two kinds of kernel definitions based on these alignments, kernel k-means clustering and trajectory compression. Our experiment and the results of clustering a hand-labelled set of vessel trajectories is described in Sect. 4. We end with some conclusions and directions for future work.

2 Related Work

We have not seen work on clustering moving object trajectories using an alignment kernel based approach. However, there are a number of papers researching the problem of clustering trajectories, mostly coming from the Moving Object Database community, such as: [6, 18, 13, 11, 15]. Like in this paper, the authors of [18] take an alignment based approach, but use a density based clustering algorithm. Also, [13] takes a density based approach but computes the distances between trajectories based on the area between them. In [11], compression based on the minimum description length principle is applied to trajectories first and then density based clustering is used to discover subtrajectories. Trajectories are first converted into a grid based representation and then clustered using fuzzy c-means in [15]. A very different approach is the somewhat older [6], in which the authors take a mixture of regression models based approach. Most approaches are based on a density based clustering algorithm, which assumes that clusters are not (densely) connected. However, we do not know whether this is the case with alignment based similarity, therefore, we prefer kernel k-means.

A lot of work exists on clustering time series data, see [12] for an overview. More specifically, there is also a body of work on using alignment kernels for time-series, for instance: [3, 9, 7, 10]. This work is mostly in the context of classification of hand-writing data using support vector machines. Both [9] and [10] use an alignment kernel based on dynamic time warping (DTW). However, [10] uses the soft-max version defined in [3], whereas [9] uses a more classic version of DTW. In the following we will consider both types.

3 Trajectory Alignment Kernels

In this section we first define a moving object trajectory. Then we look at alignments and define different kernels for them. We also briefly describe the kernel k-means clustering algorithm. Finally, we give the trajectory compression algorithm that we have used.

3.1 Trajectories

A moving object trajectory is defined in Definition 1 below. Note that trajectories include time.

Definition 1. *A moving object trajectory in 2-dimensional space is represented by a sequence of vectors: $T = \langle x_1, y_1, t_1 \rangle, \ldots, \langle x_n, y_n, t_n \rangle$. Where x_i and y_i represent the position of the object at time t_i. The length of a trajectory, i.e. the number of vectors, is denoted as: $|T|$. Furthermore, let $T_i = \langle x_i, y_i, t_i \rangle$.*

The sample-rate of such trajectories is not fixed, thus the difference between consecutive values t_i, t_{i+1} is not the same. Also there are a lot more dimensions to trajectories that can be derived from the x, y, t information that we do not consider, such as speed and direction. In some tasks and applications these attributes might be more relevant than the absolute position x, y and time t. In principle these dimensions could just be added to the $\langle x, y, t \rangle$ vector, but we have not considered this for the clustering task that we will define in Sect. 4. In the following we refer to a vector $\langle x_i, y_i, t_i \rangle$ as trajectory element or point.

3.2 Alignments

We define three types of alignments between two trajectories. We start with the simplest one, which can be considered to be the baseline alignment. The goal of an alignment is to bring the elements of two sequences, e.g. trajectories, in a correspondence, traditionally with the goal to maximize (or minimize) a scoring function defined on this correspondence.

Assume in the following that there is a function $\mathbf{sub}(S_i, T_j)$ that gives a score to substituting the trajectory element S_i with T_j. Furthermore, to enhance readability, we will not make use of superscripts to indicate the specific type of alignment when this cannot lead to confusion.

We define different alignments for two trajectories below, we adapted the notation and definitions from [17].

Definition 2. *An alignment π, possibly with gaps, of $p \geq 0$ positions between two trajectories S, T is a pair of p-tuples:*

$$\pi = ((\pi_1(1), \ldots, \pi_1(p)), (\pi_2(1), \ldots, \pi_2(p))) \in \mathbb{N}^{2p} \ .$$

The idea behind this definition is that the alignment encodes p elements in each trajectory that are aligned to each other. That is, the $\pi_1(i)$th element in S is aligned to the $\pi_2(i)$th element in T. Multiple elements of S can be aligned to one element in T and vice versa. Furthermore, not all elements have to be aligned to an element in the other trajectory, in this case an element is a *gap*.

Shortest Sequence Alignment. Traditionally, one of the simplest alignments defined on two sequences with discrete elements is the Longest Common SubSequence (LCSS) measure. It tries to find the longest sequence of elements existing in both sequences, whereby gaps are allowed. Since we are dealing with sequences of continuous elements we define it slightly differently. Definition 3 differs from other definitions of LCSS on sequences with continuous elements, such as in [18, 7], because we will not use a threshold in our substitution function. Furthermore, the resulting measure, on first glance does not have much in common with LCSS, thus to avoid confusion we redub it: Shortest Sequence Alignment (SSA).

Definition 3. *An SSA alignment π^{SSA} is an alignment according to Definition 2, but furthermore has the constraints that:*

$$p = \min(|S|, |T|)$$

and

$$\begin{aligned} 1 \leq \pi_1(1) < \pi_1(2) < \ldots < \pi_1(p) \leq |S| \ , \\ 1 \leq \pi_2(1) < \pi_2(2) < \ldots < \pi_2(p) \leq |T| \ . \end{aligned}$$

Intuitively this means that all elements of the shortest sequence are aligned with different unique elements in the other sequence, hence Shortest Sequence Alignment.

We define the score for an SSA alignment below in Definition 4.

Definition 4. *The score for an SSA alignment π of two trajectories S, T is equal to:*

$$s^{\text{SSA}}(\pi) = \sum_{i=1}^{|\pi|} \mathbf{sub}(S_{\pi_1(i)}, T_{\pi_2(i)}) \ .$$

Dynamic Time Warping. A very popular alignment method more specifically designed for time-series is Dynamic Time Warping (DTW). In Definition 5 we follow [3].

Definition 5. *A DTW alignment π^{DTW} is an alignment according to Definition 2, but furthermore has the constraints that there are unitary increments and no simultaneous repetitions, thus $\forall 1 \leq i \leq p-1$,*

$$\pi_1(i+1) \leq \pi_1(i) + 1, \ \ \pi_2(j+1) \leq \pi_2(j) + 1$$

and

$$(\pi_1(i+1) - \pi_1(i)) + (\pi_2(i+1) - \pi_2(i)) \geq 1 \ .$$

Furthermore,

$$1 = \pi_1(1) \leq \pi_1(2) \leq \ldots \leq \pi_1(p) = |S| \ ,$$
$$1 = \pi_2(1) \leq \pi_2(2) \leq \ldots \leq \pi_2(p) = |T| \ .$$

This means that all elements in both trajectories are aligned, which might require repeating elements from a trajectory, but in the alignment we cannot simultaneously repeat an element in both trajectories. Furthermore, the start and end of trajectories are aligned by default.

The score for a DTW alignment is the same as for the SSA score (Definition 4), hence Definition 6 is short. Thus, these two alignment measures differ in the kinds of alignments that are allowed.

Definition 6.

$$s^{\mathrm{DTW}}(\pi) = s^{\mathrm{SSA}}(\pi) \ .$$

Edit Distance. Edit distances are a popular method for comparing similarity of strings. This similarity is computed in terms of the number of substitutions, deletions and insertions that are required to transform one string into another string. We define (Definition 7 and 8) a version for continuous elements similar to how [2] defines it for time-series. Howewer we chose to consider fixed gap penalties, e.g. deletion and insertion costs, because this seems more natural in the trajectory case[1].

Definition 7. *An edit distance alignment π^{ED} is an alignment according to Definition 2, but also has the constraints:*

$$1 \leq \pi_1(1) < \pi_1(2) < \ldots < \pi_1(p) \leq |S| \ ,$$
$$1 \leq \pi_2(1) < \pi_2(2) < \ldots < \pi_2(p) \leq |T| \ .$$

This means that not all elements have to be aligned and there is no repetition of elements. Thus, gapping is allowed. In contrast to the SSA measure these gaps are not ignored, but have a fixed penalty g.

The score for an edit distance alignment is defined below in Definition 8. For all aligned elements we take the substitution score, and for all unaligned elements we take the gap penalty g.

[1] In [2] the gap penalties are essentially the value of the gapped element minus a fixed constant. For trajectories this would mean that something on the edge of the map would have a higher penalty then in the middle, which we think is counter-intuitive.

Definition 8. *The score for an Edit distance alignment π of two trajectories S, T is equal to:*

$$s^{\mathrm{ED}}(\pi) = \left(\sum_{i=1}^{|\pi|} \mathbf{sub}(S_{\pi_1(i)}, T_{\pi_2(i)}) \right) + g(|S| - |\pi|) + g(|T| - |\pi|) \ .$$

Considering the three types of alignments that are defined above, we note that the most important difference is in how they treat gaps. In the SSA case gaps get no penalty, in the DTW case gaps are treated by repeating a trajectory element, and thus get a score according to the substitution function **sub**, and in the edit distance case gaps have a fixed penalty.

3.3 Alignment Kernels

Below we will give two different ways to create a kernel using the alignments defined above. The first type of kernel is based on the classic way that these alignments are used. The second kernel is based on taking the soft-max of all alignments.

First, however, we define the substitution function to be the negative of the L^2 norm, i.e. the regular Euclidean distance. Other functions are possible here, but we have not experimented with this.

Definition 9.

$$\mathbf{sub}(\langle x_i, y_i, t_i \rangle, \langle x_j, y_j, t_j \rangle) = -\| \langle x_i - x_j, y_i - y_j, t_i - t_j \rangle \| \ .$$

Note that x and y are of the same dimension, but t is not directly comparable to x and y, hence in the experiments we will see that we apply a weight to t to make it comparable.

Above, in Sect. 3.2, we have defined a number of types of alignments between two trajectories and how to score them. However, this does not give a similarity between two trajectories yet, because there are a lot of possible alignments, and corresponding scores, for one type of alignment.

The classic similarity for the SSA, DTW and edit distance alignments is to take as similarity the score of the alignment that maximizes[2] the respective score function s. We will call this similarity $Sim_{\max}$ and define it below in Definition 10. In the edit distance case, gap penalties g should be negative.

Definition 10. *Given two trajectories S, T, let $\Pi^{\phi}_{S,T}$ be the set of all possible alignments between these trajectories under a certain alignment measure ϕ, then*

$$Sim^{\phi}_{\max}(S, T) = \max_{\pi \in \Pi^{\phi}_{S,T}} s^{\phi}(\pi) \ .$$

[2] Or minimizes, if the substitution score is positive.

However, it has been argued [3, 17] that taking the soft-max[3] of the scores of all possible alignments leads to better kernels, because all scores are taken into account. We will call this alignment score Sim_{sum}, because the sum over all possible alignments is taken. It is given in Definition 11 below.

Definition 11. *Given two trajectories S, T, let $\Pi^{\phi}_{S,T}$ be the set of all possible alignments between these trajectories under a certain alignment measure ϕ, then*

$$Sim^{\phi}_{\mathrm{sum}}(S,T) = \sum_{\pi \in \Pi^{\phi}_{S,T}} \exp(s^{\phi}(\pi)) \enspace .$$

As an example, for DTW alignments the above definition amounts to:

$$Sim^{\mathrm{DTW}}_{\mathrm{sum}}(S,T) = \sum_{\pi \in \Pi_{S,T}} \prod_{i=1}^{|\pi|} \exp(\mathbf{sub}(S_{\pi_1(i)}, T_{\pi_2(i)})) \enspace . \qquad (1)$$

These similarities are not kernels yet. We define a kernel based on Sim_{max} in Definition 12.

Definition 12. *For all T^i and T^j in a set of trajectories Θ, we compute the K^{ϕ}_{max}-kernel for an alignment measure ϕ as:*

$$K^{\phi}_{\mathrm{max}}(i,j) = Sim^{\phi}_{\mathrm{max}}(T^i, T^j) \enspace ,$$

furthermore we normalize K^{ϕ}_{max}:

$$K^{\phi}_{\mathrm{max}} = 1 - \frac{K^{\phi}_{\mathrm{max}}}{\min(K^{\phi}_{\mathrm{max}})} \enspace .$$

The kernel based on Sim_{sum} is given in Definition 13.

Definition 13. *For all T^i and T^j in a set of trajectories Θ, we compute the K^{ϕ}_{sum}-kernel for an alignment measure ϕ as:*

$$K^{\phi}_{\mathrm{sum}}(i,j) = Sim^{\phi}_{\mathrm{sum}}(T^i, T^j) \enspace ,$$

furthermore we normalize K^{ϕ}_{sum}:

$$K^{\phi}_{\mathrm{sum}}(i,j) = \frac{K^{\phi}_{\mathrm{sum}}(i,j)}{\sqrt{K^{\phi}_{\mathrm{sum}}(i,i) K^{\phi}_{\mathrm{sum}}(j,j)}} \enspace .$$

The K_{max} kernels are not proper kernels in the sense that they are not Positive Semi-Definite (PSD). However it has been observed that non-PSD kernels can still work well in practice (for DTW for instance in [9]). We notice this in our experiment in Sect. 4 aswell. The soft-max version of the DTW alignment kernel was proven to be PSD in [3] given a proper substitution function (e.g. the one

[3] Given a set of positive scalars $Z = z_1, \ldots, z_n$, the soft-max of Z is $\log \sum e^{z_i}$.

we have defined). Furthermore, we conjecture that similar proofs are possible for the other two soft-max kernels we define, because they are very similar to either DTW or Smith-Waterman, for which there is a proof in [17].

The authors of [3, 17] also notice that the soft-max type of kernel often suffers from the diagonal dominance issue and they remedy this by taking the logarithm of this kernel. We have experimented with taking the logarithm, but this did not improve results.

The classic alignment similarities, $\mathit{Sim}_{\max}$, are efficiently computable via a dynamic programming approach. The same holds for the $\mathit{Sim}_{\text{sum}}$ version of DTW, as shown in [3]. This is done by replacing the max-sum algebra with a sum-product algebra. Insight that this works can be gained by looking at equation 1. In [17] it is shown how the same approach can be applied to the Smith-Waterman alignment measure. Because of the similarity of the DTW and Smith-Waterman alignments with the SSA and edit distance alignments it is clear that the same approach is valid for the SSA and edit distance alignments.

To sum it up, we have now defined a number of kernel functions. Two for the SSA measure: $K^{\text{SSA}}_{\max}$ and $K^{\text{SSA}}_{\text{sum}}$. Two for the DTW measure: $K^{\text{DTW}}_{\max}$ and $K^{\text{DTW}}_{\text{sum}}$. And a family of kernels for the edit distance measure, since these are parameterized by the value of the gapping penalty g: $K^{\text{ED}}_{\max}$ and $K^{\text{ED}}_{\text{sum}}$.

3.4 Kernel K-means

Because the focus of the paper is on the alignments, we use the relatively simple kernel k-means algorithm [16], though more advanced kernel based algorithms exist. The kernel version of k-means works in exactly the same way as regular k-means. This means that we start out with a random initialization of k clusters and at each iteration try to minimize the distance between the cluster centers and objects in the clusters until we arrive at a stable clustering (Definition 14).

Definition 14. *A clustering C of a set of objects Θ into k partitions is:*

$$C = \{c_1, \ldots, c_k\} \ ,$$

such that for all $i \in \Theta$, i is exactly in one c.

The difference lies in how the distance from an object to a cluster center is calculated. This is done using the assumption that a kernel represents a dot-product in a higher dimensional feature space. This means that it can be used to calculate the euclidean distance in that space, as in Definition 15.

Definition 15. *Let K be a kernel computed for a set of objects Θ. The distance from an object $i \in \Theta$ to the center of a cluster c in kernel k-means is computed by:*

$$K(i,i) - 2\left(\sum_{j \in c} K(i,j)\right) + \sum_{j,k \in c} K(j,k) \ .$$

Because different random initializations can lead to different stable partitions C, the kernel k-means clustering is run a number of times. The partitioning with the lowest inter cluster spread is kept as the final clustering.

3.5 Trajectory Compression

For trajectory compression we use the method proposed in [8], which is an advanced version of a classic line simplification algorithm [5]. The algorithm, given in Algorithm 1, compresses a trajectory T by finding the point $\langle x_i, y_i, t_i \rangle$ that has the largest error $\mathbf{E}_\mu$, as defined in Definition 16. If this error is larger than a given threshold ϵ then we keep this point and recursively apply the same procedure on two subtrajectories created by splitting T at that point. Thus the result of applying trajectory compression to a trajectory T is that the compressed trajectory T^C is a subset of the points that are in T, always including the start and end points. This subset contains the most salient points of the trajectory. Unlike in for instance smoothing, no new points are created or altered.

Algorithm 1 $\mathbf{pls}(T, \epsilon)$

```
1  We use $end$ to indicate the index of the last element of a trajectory.
2  $d_{max} = 0$, $i_{max} = 0$
3  for $i = 2$ to $end - 1$ do
4    $d = \mathbf{E}_\mu(T_i, T_1, T_{end})$
5    if $d > d_{max}$ then
6      $i_{max} = i$, $d_{max} = d$
7    end
8  end
9  if $d_{max} \geq \epsilon$ then
10   $A = \mathbf{pls}(T_1, \ldots, T_{i_{max}}, \epsilon)$, $B = \mathbf{pls}(T_{i_{max}}, \ldots, T_{end}, \epsilon)$
11   $T^C = A, B_2, \ldots, B_{end}$
12 else
13   $T^C = T_1, T_{end}$
14 end
15 return $T^C$
```

In Definition 16, the error $\mathbf{E}_\mu$ is defined as the Euclidean distance between a point $\langle x_i, y_i, t_i \rangle$ and the closest point on the line $\langle x_1, y_1, t_1 \rangle, \langle x_n, y_n, t_n \rangle$, where the influence of the temporal dimension is weighed using the parameter μ.

Definition 16.

$$\mathbf{E}_\mu(\langle x_i, y_i, t_i \rangle, \langle x_1, y_1, t_1 \rangle, \langle x_n, y_n, t_n \rangle) = \| \langle x_i - x_j, y_i - y_j, \mu(t_i - t_j) \rangle \| \ ,$$

where $\langle x_i', y_i', t_i' \rangle$ *is the closest point on the line-segment* $\langle x_1, y_1, t_1 \rangle, \langle x_n, y_n, t_n \rangle$.

4 Evaluation

In this section we will present a number of experiments to test the performance of the different combinations of alignment measures and kernels on the task of clustering a set of vessel trajectories. We especially investigate the effect on performance of applying a compression algorithm to these trajectories as a preprocessing step.

4.1 Dataset

Our experimental dataset consist of 716 vessel trajectories originally gathered using the Automatic Identification System (AIS). A domain expert has partitioned these trajectories into 8 different clusters, creating a gold standard $G = g_1, \ldots, g_8$. The clusters are very different in size, ranging from 8 to 348. The average length of a trajectory is a sequence of 300 $\langle x, y, t \rangle$ vectors. Trajectories are delimited either by the vessel leaving the area of observation or the vessel stopping. For all trajectories t_1 is set to 0. We illustrate the clustering in Fig. 1. The general direction of movement for each of the 8 clusters is indicated with an arrow. Note that we only plotted the x and y dimensions and not t. The position data of the vessels was originally collected in a latitude, longitude format but has been converted using a suitable map projection to allow for normal euclidean geometry.

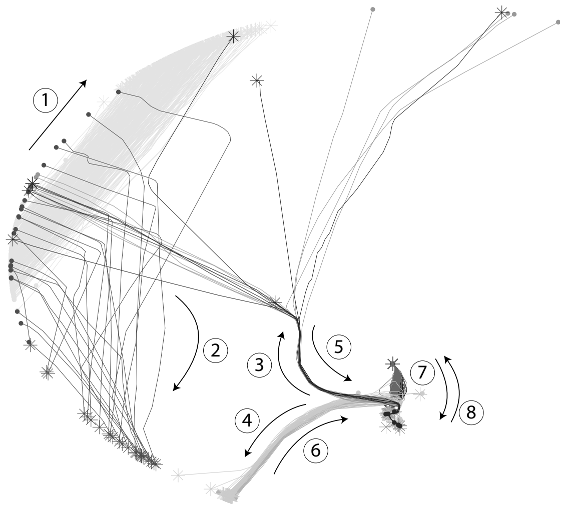

Fig. 1. An overview of the vessel trajectory dataset with the 8 different clusters of the gold standard indicated with shades of gray and an arrow giving the general direction. A solid dot indicates the start of a trajectory and an asterix indicates the end.

4.2 Experimental Setup

As test datasets we constructed 4 random subsets from the 716 vessels set. From each cluster in the gold standard we randomly extracted up to 20 trajectories. Some clusters contained fewer elements in which case we took them all. This resulted in 4 datasets of 138 trajectories. We did this to allow for reasonable computation times while doing our experiments.

We defined a number of settings to weigh the time dimension against the space dimensions. In the first setting we treated the trajectories as if they were sampled with a fixed samplerate, which means that sometimes positions had to be linearly interpolated between two positions from the original data. In this setting a trajectory is a sequence of $\langle x, y\rangle$ points and time is implicitly represented by the fixed time between these points. In the other settings we tested 4 weights w for the time dimension. So that a trajectory essentially becomes: $T = \langle x_1, y_1, wt_1\rangle, \ldots, \langle x_n, y_n, wt_n\rangle$. As weights we took $w = 0, \frac{1}{3}, \frac{2}{3}, 1$. With $w = 0$ we ignore the time dimension and with the other weights we increasingly weigh the time dimension more heavily. We made it so that the $w = 1$ setting means that the average difference between two points in the space dimension is roughly equal to the average difference in the time dimension[4].

To investigate the influence of trajectory compression on the clustering performance we used 4 compression settings. The first setting being $\epsilon = 0$, thus, we apply no compression. In the other settings we applied algorithm 1 to each trajectory with $\epsilon = 25, 50, 100\,\mathrm{m}$, respectively. We fixed the μ parameter with a value that we determined during previous experiments to be reasonable. In the fixed samplerate setting we set the samplerate to be (very) roughly equal to the compression rate achieved under the different compression settings, see Table 1. We set the samplerate for the no compression case to the average distance between two consecutive trajectory samples in the uncompressed dataset, again see Table 1.

Table 1. Average compression rate and corresponding samplerate for different ϵ settings.

	No compression	$\epsilon = 25\,\mathrm{m}$	$\epsilon = 50\,\mathrm{m}$	$\epsilon = 100\,\mathrm{m}$
Compression rate (%)	0	96	97	98
Samplerate (Hz)	0.1	0.01	0.002	0.001

For each of the different combinations of settings, i.e. for each combination of subset, time setting and compression setting, we computed a number of kernels. Two for the SSA measure: $K^{\mathrm{SSA}}_{\max}$ and $K^{\mathrm{SSA}}_{\mathrm{sum}}$. Two for the DTW measure: $K^{\mathrm{DTW}}_{\max}$ and $K^{\mathrm{DTW}}_{\mathrm{sum}}$. And 10 for the Edit measure; five times $K^{\mathrm{ED}}_{\max}$ and five times $K^{\mathrm{ED}}_{\mathrm{sum}}$, with $g = -0.01, -0.025, -0.05, -0.075, -0.1$.

We used each of these kernels as input for the kernel k-means clustering algorithm. The kernel k-means algorithm is run 100 times with random initalizations and we keep the clustering with minimal inter cluster spread. This process is repeated 10 times. We set $k = 8$, i.e. the same amount of clusters as in the gold standard. Thus for each kernel we get 10 clusterings that we evaluate against the gold standard G according to the scoring function in Definition 17.

[4] To enhance readability we do not give the actual values for the weights, because they depend on the units that the dimensions are in.

Definition 17. *The score of a clustering C of size k with respect to a gold standard G of size k is:*

$$\mathbf{score}(C, G) = \frac{1}{k} \sum_{i=1}^{k} \max_{1 \leq j \leq k} \frac{2|g_i \cap c_j|}{|g_i| + |c_j|} \quad .$$

4.3 Results

In the results below the mean of a kernel is computed over the scores for the 4 subsets with 10 repeats each, thus for each kernel $N = 40$. Per kernel we statistically compare the scores for the high compression/low samplerates settings to the scores for the no compression/high samplerate setting using a two-tailed student t-test with $p < 0.05$.

We do not give all the mean scores because of space constraints. Also, discussion of the presented results is postponed to Sect. 5. First we will look at the performance of the different kernels under trajectory compression. Therefore we look at the time setting for which the highest scores are achieved. In Table 2 we give the results for the 14 kernels under the time setting $w = 0$. Scores in **bold** indicate a significant difference between that clustering score, according to the above defined test, and the clustering score for the no compression case, with the bold score being higher. Scores in *italic* indicate a significant difference between that score and the score for the no compression case, with the score in italic being lower. For completeness we give the mean score of 40 random clusterings, which is 0.24.

We can see that the best performing kernels for our clustering task are the $K^{\text{ED}}_{\text{max}}$ kernels. With the right g setting they can achieve a perfect clustering, e.g. a score of 1.0. These kernels perform better on the compressed trajectories than on the uncompressed trajectories. However, the $K^{\text{ED}}_{\text{sum}}$ kernels perform worse on the compressed trajectories. The $K^{\text{DTW}}_{\text{max}}$ kernel does not perform so much better on the compressed data as the $K^{\text{ED}}_{\text{max}}$ kernels. Both shortest sequence alignment kernels perform a lot worse than the other kernels. The $K^{\text{ED}}_{\text{sum}}$ and $K^{\text{DTW}}_{\text{sum}}$ kernels perform almost identical in the uncompressed case.
Next we will look at the effect of the different time settings. In Table 3 we give a selection of kernel results for the fixed samplerate time setting. We have included the two edit distance kernels with the best performance and also added the best performing kernel ($K^{\text{ED}}_{\text{max},g=-0.05}$) for the $w = 0$ setting. Bold and italic have a similar meaning as in the previous table. Furthermore, $^+$ indicates a significant difference with the corresponding value from Table 2 if this difference is positive. This is similar for $^-$ but the difference is negative. We can see that there are better performing kernels in the $w = 0$ setting. Table 3 also shows that low frequency fixed samplerates ($\leq 0.01\,\text{Hz}$) do not degrade the performance of a number of kernels and in one case even improves it.
In Table 4 we give the results for a selection of kernels for the setting $w = \frac{1}{3}$. We have taken the same edit distance kernels as in Table 3, which also includes the best performing one for this setting. Again, $^+$ and $^-$ indicate statistical

Table 2. Value of **score** for 14 different kernels under time setting $w = 0$.

Kernel	No compression	$\epsilon = 25\,\mathrm{m}$	$\epsilon = 50\,\mathrm{m}$	$\epsilon = 100\,\mathrm{m}$
max, SSA	0.52	*0.34*	*0.35*	*0.42*
max, DTW	0.87	**0.90**	**0.91**	**0.91**
max, ED, $g = -0.01$	0.91	0.92	0.93	**0.94**
max, ED, $g = -0.025$	0.88	**1.0**	**1.0**	**1.0**
max, ED, $g = -0.05$	0.88	**1.0**	**1.0**	**1.0**
max, ED, $g = -0.075$	0.88	**0.93**	**0.97**	**0.99**
max, ED, $g = -0.1$	0.87	**0.90**	**0.89**	**0.96**
sum, SSA	0.52	*0.34*	*0.35*	*0.33*
sum, DTW	0.87	*0.82*	*0.85*	0.87
sum, ED, $g = -0.01$	0.87	*0.62*	*0.63*	*0.65*
sum, ED, $g = -0.025$	0.87	*0.64*	*0.64*	*0.63*
sum, ED, $g = -0.05$	0.87	*0.63*	*0.63*	*0.66*
sum, ED, $g = -0.075$	0.87	*0.64*	*0.63*	*0.66*
sum, ED, $g = -0.1$	0.87	*0.64*	*0.63*	*0.67*

Table 3. Value of **score** for a selection of 7 different kernels under the fixed samplerate setting.

Kernel	0.1 Hz	0.01 Hz	0.002 Hz	0.001 Hz
max, SSA	0.47^-	0.46^+	0.47^+	0.45^+
max, DTW	0.87	0.87^-	0.87^-	0.87^-
max, ED, $g = -0.01$	0.95^+	0.94	$\mathbf{0.97}^+$	0.94
max, ED, $g = -0.05$	0.87^-	$\mathbf{0.87}^-$	$\mathbf{0.87}^-$	0.87^-
sum, SSA	0.48^-	*0.39*$^-$	*0.29*$^-$	*0.26*$^-$
sum, DTW	0.86^-	*0.71*$^-$	*0.70*$^-$	*0.75*$^-$
sum, ED, $g = -0.01$	0.86^-	*0.62*	*0.61*$^-$	*0.59*$^-$

significance compared to the same kernel for the $w = 0$ setting. We see that there are better performing kernels in the $w = 0$ setting.

Table 4. Value of **score** for a selection of 7 different kernels under time setting $w = \frac{1}{3}$.

Kernel	No compression	$\epsilon = 25\,\mathrm{m}$	$\epsilon = 50\,\mathrm{m}$	$\epsilon = 100\,\mathrm{m}$
max, SSA	0.52	*0.36*$^+$	*0.42*$^+$	0.52^+
max, DTW	0.87^-	$\mathbf{0.89}^-$	$\mathbf{0.89}^-$	**0.90**
max, ED, $g = -0.01$	0.89^-	*0.82*$^-$	*0.80*$^-$	*0.80*$^-$
max, ED, $g = -0.05$	0.88	$\mathbf{0.95}^-$	$\mathbf{0.98}^-$	$\mathbf{0.99}^-$
sum, SSA	0.51	*0.33*	*0.35*	*0.33*
sum, DTW	0.83^-	*0.82*	0.84	**0.86**
sum, ED, $g = -0.01$	0.83^-	*0.63*	*0.64*	*0.65*

The performance for the time settings $w = \frac{2}{3}, 1$ is worse than the results for $w = \frac{1}{3}$. Thus, we considered it not necessary to include them. We furthermore remark that the difference in mean score between the best performing kernel in the $w = 0$ setting ($K^{\text{ED}}_{\max,g=-0.05}, \epsilon = 50\,\text{m}, \textbf{score} = 1.0$) and the best performing kernel in the fixed samplerate setting ($K^{\text{ED}}_{\max,g=-0.01}, 10\,\text{Hz}, \textbf{score} = 0.95$) is significant.

We also give some results on computation time in Table 5. These computation times are for running our MatLab implementation of the different alignment kernels on an AMD Phenom II X4 955 (3.2 Ghz) cpu system with 4 GB of ram. The computation time for a kernel on the full dataset would be around a factor 40 larger. This does not make it intractable, but currently did not allow us enough flexibility in performing experiments.

Table 5. Running time for computing DTW kernels. In the compression cases, the computation time needed for compression is included.

Kernel type	No compression	$\epsilon = 25$m	$\epsilon = 50$m	$\epsilon = 100$m
max	330s	2.4s	1.9s	1.7s
sum	356s	2.6s	2.1s	1.8s

4.4 Performance on the Full Dataset

We have also computed a set of kernels on the full dataset. Because of long computation time we did not do this for the no compression/high samplerate setting. We used the same kernel k-means settings as before, which results in 10 clusterings per kernel. In general we see the same performance differences as above, but the absolute numbers are lower. The highest performance, 0.86, is achieved by $K^{\text{ED}}_{\max}$ with $g = -0.075$ under the $w = \frac{2}{3}$ and $\epsilon = 50\,\text{m}$ settings. However, this score does not differ significantly from the best score, 0.85, in the $w = 0$ setting, which is for $K^{\text{ED}}_{\max}$ with $g = -0.025$ and $\epsilon = 25\,\text{m}$.

5 Conclusions & Future Work

The main result is that the $K^{\text{ED}}_{\max}$ and $K^{\text{DTW}}_{\max}$ kernels perform better on the compressed trajectories, even though we see from Tables 1 and 5 that the dataset is reduced by 96% or more, and kernel computation time is at least 100 times faster. The trajectory compression algorithm works by recursively retaining the most salient elements of a trajectory, reducing noise. Removing so many points from the trajectories can have a negative influence on the alignments. However, in our dataset, similar trajectories have similar salient elements, vessels stop and turn in the same places due to geographical and legislative constraints. This means that similar compressed trajectories can be aligned well. In such a setting applying trajectory compression as a preprocessing step can reduce computational costs significantly and improve performance.

That the $K^{\mathrm{ED}}_{\mathrm{max}}$ kernels perform better than the $K^{\mathrm{DTW}}_{\mathrm{max}}$ kernels on the compressed trajectories can be explained by how gaps are treated. In the edit distance case only a relatively small gap penalty is added for unaligned points. However, in the DTW case, all points have to be aligned. Because of compression, consecutive points in a trajectory can be very far apart, and repeating an element can lead to a high penalty. For the $K^{\mathrm{ED}}_{\mathrm{sum}}$ kernels the relatively small gap penalty actually seems to work against them, since bad alignments still have a relatively high score: the worst score is a summing of just gap penalties. And these high scores are a relatively big, but meaningless, part of the sum. This does not happen in the $K^{\mathrm{DTW}}_{\mathrm{sum}}$ case, bad alignments do not get such a relatively high score. We see that the $K^{\mathrm{DTW}}_{\mathrm{sum}}$ kernels perform almost as wel as $K^{\mathrm{DTW}}_{\mathrm{max}}$ versions. The almost identical performance of the $K^{\mathrm{ED}}_{\mathrm{sum}}$ and $K^{\mathrm{DTW}}_{\mathrm{sum}}$ kernels in the uncompressed case is because the amount of points in a trajectory is so high that there are very little gaps compared to aligned elements, and thus the influence of how the DTW and edit kernels treat gaps is very small.

The main reason for shortest sequence alignment kernels to perform so much worse is that a trajectory of a small number of elements can still be very similar to a trajectory with a lot more elements, because we do not penalize gaps. In most cases, such trajectories are actually not very similar.

Inspection of the dataset shows that the best performance being in the $w = 0$ setting is not so strange. Within one cluster most trajectories actually have similar speeds, and time is thus not so important, the sequence of the $\langle x, y \rangle$ points is enough. We have not investigated what happens if we do not set each start time of the trajectory to 0, but, for instance, temporally align them differently, this is something for future work.

In future work we would also like to look at other types of movement data to test the influence of trajectory compression. Especially data with less structure than vessel trajectories and a larger role of the temporal component. Furthermore, it would be interesting to look at other types of clustering methods with these alignments, for instance density based. Finally, we would like to look at similarity measures from computational geometry, as for instance in [1, 13].

Acknowledgments

This work has been carried out as a part of the Poseidon project in collaboration with Thales Nederland, under the responsibilities of the Embedded Systems Institute (ESI). This project is partially supported by the Dutch Ministry of Economic Affairs under the BSIK program.

References

1. Buchin, K., Buchin, M., van Kreveld, M.J., Luo, J.: Finding long and similar parts of trajectories. In: Agrawal, D., Aref, W.G., Lu, C.T., Mokbel, M.F., Scheuermann, P., Shahabi, C., Wolfson, O. (eds.) GIS. pp. 296–305. ACM (2009)

2. Chen, L., Ng, R.: On the marriage of lp-norms and edit distance. In: VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases. pp. 792–803. VLDB Endowment (2004)
3. Cuturi, M., Vert, J.P., Birkenes, O., Matsui, T.: A kernel for time series based on global alignments. In: Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on. vol. 2, pp. II–413 –II–416 (april 2007)
4. Dhillon, I.S., Guan, Y., Kulis, B.: Kernel k-means: spectral clustering and normalized cuts. In: KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 551–556. ACM, New York, NY, USA (2004)
5. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica 10(2), 112–122 (October 1973)
6. Gaffney, S., Smyth, P.: Trajectory clustering with mixtures of regression models. In: Knowledge Discovery and Data Mining. pp. 63–72 (1999)
7. Gruber, C., Gruber, T., Sick, B.: Online signature verification with new time series kernels for support vector machines. In: Zhang, D., Jain, A.K. (eds.) ICB. Lecture Notes in Computer Science, vol. 3832, pp. 500–508. Springer (2006)
8. Gudmundsson, J., Katajainen, J., Merrick, D., Ong, C., Wolle, T.: Compressing spatio-temporal trajectories. Computational geometry 42(9), 825–841 (2009)
9. Gudmundsson, S., Runarsson, T.P., Sigurdsson, S.: Support vector machines and dynamic time warping for time series. In: IJCNN. pp. 2772–2776. IEEE (2008)
10. Joder, C., Essid, S., Richard, G.: Alignment kernels for audio classification with application to music instrument recognition. In: European Signal Processing Conference (EUSIPCO). Lausanne, Suisse (Aug 2008)
11. Lee, J.G., Han, J., Whang, K.Y.: Trajectory clustering: a partition-and-group framework. In: SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data. pp. 593–604. ACM, New York, NY, USA (2007)
12. Liao, T.W.: Clustering of time series data–a survey. Pattern Recognition 38(11), 1857 – 1874 (2005)
13. Nanni, M., Pedreschi, D.: Time-focused clustering of trajectories of moving objects. J. Intell. Inf. Syst. 27(3), 267–289 (2006)
14. Ng, A.Y., Jordan, M.I., Weiss, Y.: On spectral clustering: Analysis and an algorithm. In: Dietterich, T.G., Becker, S., Ghahramani, Z. (eds.) NIPS. pp. 849–856. MIT Press (2001)
15. Pelekis, N., Kopanakis, I., Kotsifakos, E., Frentzos, E., Theodoridis, Y.: Clustering trajectories of moving objects in an uncertain world. In: ICDM '09: Proceedings of the 2009 Ninth IEEE International Conference on Data Mining. pp. 417–427. IEEE Computer Society, Washington, DC, USA (2009)
16. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, New York, NY, USA (2004)
17. Vert, J.P., Saigo, H., Akutsu, T.: Local alignment kernels for biological sequences. In: Scholkopf, B., Tsuda, K., Vert, J.P. (eds.) Kernel Methods in Computational Biology, pp. 131–154. The MIT Press (2004)
18. Vlachos, M., Gunopoulos, D., Kollios, G.: Discovering similar multidimensional trajectories. In: ICDE '02: Proceedings of the 18th International Conference on Data Engineering. p. 673. IEEE Computer Society, Washington, DC, USA (2002)