Kernel methods for vessel trajectories

de Vries, G.K.D.

Citation for published version (APA):
This chapter extends work by Willems et al. (2010); van Hage et al. (2011, 2009). The ferry example was first discussed by de Vries et al. (2008).

The volume of moving object trajectory data is huge and often quite regular, thus it makes sense to apply a trajectory compression technique to make the data more manageable. In doing analysis of moving objects, the concepts of stop and move are essential movement primitives. We show in this chapter that standard trajectory compression techniques, based on line-simplification, are not ideally suited to retain the stop and move information. Therefore we propose a simple extension to this method that looks at speed along a trajectory first. We evaluate standard trajectory compression and our technique on the task of stop retention using a dataset of vessel trajectories. The conclusion from this evaluation is that our method generally has better stop retention at the same compression rate. Furthermore, we show how we can use this method to create the semantic building blocks for higher level reasoning about vessel behavior.

3.1 Introduction

Moving object trajectories are often sampled every couple of seconds, which results in a large amount of data. However, in a number of cases, such as cars following a road, or ships in a sea-lane, the data is highly regular. In such cases it makes sense to make the data more manageable without losing much information by using a trajectory compression technique. For instance, large scale data reduction makes computation of similarity between trajectories much faster, as we see in Chapter 4, and it also allows for higher level reasoning, of which we give examples in this chapter.

As recognized by Spaccapietra et al. (2008), the concepts of stop and move are essential semantic movement primitives for the analysis of moving objects. Thus, when using a trajectory compression method we should not compress movement data in such a way that information about these concepts is lost. In other words, trajectory compression should retain stops and moves. We dub this problem stop retention.

For our vessel trajectory data we consider a standard trajectory compression technique based on line-simplification (Gudmundsson et al. (2009)), which we call Piecewise Linear Segmentation (PLS). We will show the performance of this method at stop retention for a number of different error measures and conclude that this stan-
standard method is not ideal. Therefore, we propose a modification of PLS-based trajectory compression that explicitly looks at the speed, i.e. the partial derivative to time of the trajectory, of a moving object first. This modification allows for a better retention of the stop and move concepts at the same compression rate. Stop retention also has the advantage that it allows us to define the concept of segments on compressed trajectory data. These segments are used as atomic movement events that allow for reasoning to derive more complex behavior patterns, which we illustrate with an example.

The goal of our method is not the task of stop detection on its own, for which there are methods such as SMoT (Alvares et al. (2007)), CB-SMoT (Palma et al. (2008)) and DB-SMoT (Rocha et al. (2010)). Our method aims at retaining stops when applying trajectory compression.

The rest of this chapter is structured as follows. In Section 3.2 we introduce the standard piecewise linear segmentation compression technique and introduce our own extension that looks at speed first. We evaluate the standard technique and our extension with respect to stop retention in Section 3.3. Section 3.4 introduces the segment representation that we define upon compressed trajectories and gives examples of reasoning towards detecting complex vessel behavior with these segments. We end with some conclusions in Section 3.5.

3.2 Trajectory Compression

In this section we first review some trajectory related definitions. Then we describe piecewise linear segmentation with a number of different error measures. We give a problematic example for this standard algorithm and propose a solution in terms of an extension. Finally, we say something about query error-bounds and computational complexity.

3.2.1 Trajectory Definitions

For a definition of a trajectory we refer to Definition 2.2.1 in Chapter 2. In this chapter we also explicitly consider the speed of a moving object \( v \). Therefore let \( T_v = \langle x_1, y_1, v_1, t_1 \rangle, \ldots, \langle x_n, y_n, v_n, t_n \rangle \) be a trajectory for which we also have the speed \( v_i \) for the moving object at time \( t_i \). This speed can either be reconstructed from \( T \), or derived from a special sensor.

Spaccapietra et al. (2008) argue that in analyzing the behavior of moving objects the concepts of stop and move are essential. What are stops and moves is application dependent. The part of a trajectory that constitutes a stop is the non-empty time interval for which the traveling object does not move from the application’s perspective. A
move is a part of a trajectory that is not a stop. We see in Section 3.3.1 what are considered stops and moves in our application.

3.2.2 Piecewise Linear Segmentation

In the field of moving object databases different techniques have been studied to compress trajectory data (Meratnia and de By (2004); Cao et al. (2006); Potamias et al. (2006); Frentzos and Theodoridis (2007); Gudmundsson et al. (2009); Ni and Ravishankar (2007)). The most common one is a line-simplification method, adapted to trajectory data, known as the Douglas-Peucker algorithm (Douglas and Peucker (1973)) in cartography, and Ramer’s algorithm (Ramer (1972)) in image processing. In the world of data-mining from time-series it yet again goes by other names (Keogh et al. (2001)), we shall refer to it, rather generically, as Piecewise Linear Segmentation (PLS). This method is intuitive, easy to implement, relatively fast (compared to optimal methods (Cao et al. (2006))) and gives good results.

The PLS-algorithm compresses a trajectory \( T \) into linear segments by recursively keeping the points that have maximum error higher than a fixed threshold \( \epsilon \). The pseudo code for this algorithm is given in Algorithm 3.2.1. It works in the following way. The first and last points, \( T(1) = \langle x_1, y_1, v_1, t_1 \rangle \) and \( T(n) = \langle x_n, y_n, v_n, t_n \rangle \), of a trajectory \( T \) of length \( n \) are selected, and for all intermediate points we compute the error, using the function \( \Phi \), with respect to these points. If the maximum of these errors is greater than a certain threshold, then we apply the procedure again, with the corresponding point \( T(i) \) both as start and end-point, i.e. we recursively apply the procedure to the trajectory from \( T(1) \) to \( T(i) \) and \( T(i) \) to \( T(n) \). When there is no error greater than the given threshold, then we just keep the points \( \langle x_1, y_1, v_1, t_1 \rangle \) and \( \langle x_n, y_n, v_n, t_n \rangle \). Thus, the goal of the algorithm is to reduce the number of points in a trajectory while keeping the maximum deviation, or error, from the original trajectory within the threshold \( \epsilon \).

For trajectories, \( \Phi \) can be a number of different error functions \( E \) (Algorithm 3.2.1, line 5), which we will introduce below. We define all these error functions in terms of the line-segment model, because the line model can lead to some counter-intuitive results (Gudmundsson et al. (2009)). The line-segment model considers the line-segment between two points when computing the error according to some \( E \), whereas the line model considers the line through two points, which is infinite in length. Thus, a point can be very far away from a line-segment between two points, but very close to, or even lie on, the infinite line through those points.

A number of authors have proposed error functions \( E \) to use in the above algorithm. Most recently, Gudmundsson et al. (2009) proposed
Algorithm 3.2.1 \( \text{pls}_\mu(T, \epsilon) \)

1. We use \( \text{end} \) to indicate the index of the last element of a trajectory.

2. \( d_{\max} = 0 \)
3. \( i_{\max} = 0 \)
4. for \( i = 2 \) to \( \text{end} - 1 \) do
   5. \( d = \Phi(T(i), T(1), T(\text{end})) \)
   6. if \( d > d_{\max} \) then
      7. \( i_{\max} = i \)
      8. \( d_{\max} = d \)
   end
5. end
6. if \( d_{\max} \geq \epsilon \) then
   7. \( A = \text{pls}(T(1, i_{\max}), \epsilon) \)
   8. \( B = \text{pls}(T(i_{\max}, \text{end}), \epsilon) \)
   9. \( T_C = A, B(2, \text{end}) \)
else
   10. \( T_C = T(1), T(\text{end}) \)
end
11. return \( T_C \)

an error function \( E_\mu \) that generalizes most of the earlier defined error functions, so we consider this one first.

Definition 3.2.1.

\[
E_\mu(\langle x_i, y_i, t_i \rangle, \langle x_1, y_1, t_1 \rangle, \langle x_n, y_n, t_n \rangle) = \| \langle x_i, y_i, \mu t_i \rangle - \langle x'_i, y'_i, \mu t'_i \rangle \| ,
\]

where \( \langle x'_i, y'_i, t'_i \rangle \) is the closest point to \( \langle x_i, y_i, t_i \rangle \) on the line-segment \( \langle x_1, y_1, t_1 \rangle, \langle x_n, y_n, t_n \rangle \).

The parameter \( \mu \) determines the ratio between the space and time dimensions. Different settings of \( \mu \) lead to different previously defined error functions, which we illustrate in Figure 5.

In terms of \( E_\mu \), the original error measure \( E_2 \) of the Douglas-Peucker algorithm is defined as:

Definition 3.2.2.

\[
E_2 = E_{\mu=0} .
\]

With this function, a trajectory is treated as a line in 2-dimensional space, ignoring the time dimension. \( E_2 \) is illustrated in Figure 5.

Time can also be treated as just another dimension, i.e. a trajectory is a line a 3-dimensional space. This leads to the standard error function \( E_3 \), also shown in Figure 5.

Definition 3.2.3.

\[
E_3 = E_{\mu=1} .
\]
Also very common is $E_u$ (Cao et al. (2006); Meratnia and de By (2004)), where the time dimension is treated differently, because we take the difference between the point and its linear interpolation based on time on the line-segment, see Figure 5.

Definition 3.2.4.

$$E_u = E_{\mu=\infty}.$$  

Cao et al. (2006) also define the error measure $E_t$, illustrated in Figure 5, which is more or less the dual of $E_u$ and not definable in terms of $E_\mu$. Instead of determining the spatial difference at the same time, we determine the temporal difference at the same place, or closest place. Compression using just this error measure is incomplete, because errors in the spatial dimension are ignored. Contrast this with $E_2$, which ignores the temporal dimension completely.

Definition 3.2.5.

$$E_t(\langle x_i, y_i, t_i \rangle, \langle x_1, y_1, t_1 \rangle, \langle x_n, y_n, t_n \rangle) = \sqrt{\left(\frac{1}{\mu} - t'_i\right)^2},$$

where $t'_i$ is the time of the point $\langle x'_i, y'_i \rangle$ on the 2-dimensional projection of $\langle x_1, y_1, t_1 \rangle, \langle x_n, y_n, t_n \rangle$ on the xy-plane that is closest to the 2-dimensional projection of $\langle x_i, y_i, t_i \rangle$ on the xy-plane.
Therefore, Cao et al. (2006) define an error measure that combines $E_u$ and $E_t$. This is done by simply taking the maximum of the two error functions. However, it is not entirely clear how the difference in scale between the spatial ($E_u$) and temporal ($E_t$) error should be treated. In our implementation we divide the error by the error threshold $\epsilon$ for $E_u$ and $E_t$ separately, and take the maximum of those two values.

**Definition 3.2.6.**

$$E_u \land E_t = \max(E_u, E_t)$$

We will use a subscript to indicate which error function is used with PLS, for example $\text{pls}_{E_u}$.

### 3.2.3 A Problematic Example

Consider the example of the application of PLS with $E_u$ in Figure 6 (a) and (b). This figure illustrates the compression of a trajectory consisting of 5 points, $p_1, \ldots, p_5$. Figure 6 (a) shows the initial state of the compression, where the trajectory is approximated by the line-segment between $p_1$ and $p_5$. In this situation $p_4$ has the greatest distance to this segment. This distance is over the threshold $\epsilon$, thus we retain point $p_4$, and we get Figure 6 (b). The stop in the trajectory between $p_2$ and $p_3$ is not retained in the compression, because the distances to the simplification $p_1, p_4$ are not large enough and hence not over $\epsilon$. The result is that in the compressed trajectory it seems as if the object has moved (slowly) between $p_1$ and $p_4$, instead of two faster moves with a stop in between. Should the stop between $p_2$ and $p_3$ be of a longer duration then eventually also the points $p_2$ and $p_3$ should be retained. However, if the distance between $p_1$ and $p_2$, and $p_3$ and $p_4$ is below $\epsilon$ then this will never happen, no matter how long...
the stop takes. Using $E_2$, $E_3$ or $E_u$ would not change things, since they are even less strict than $E_u$ (Cao et al. (2006)).

The stop in the example would be retained if we used $E_t$ or $E_u \wedge E_t$. However, we will see in the evaluation section that this comes at the cost of quite a bit lower compression rates.

3.2.4 Two Stage Piecewise Linear Segmentation

To deal with the problem presented above we extend the standard algorithm. The intuition behind this extension is that the stopping and moving behavior of moving objects is more apparent in the derivative, i.e. the speed time-series, of the trajectory than in the trajectory itself. Therefore we propose a simple extension to the earlier defined trajectory compression, which we give in Algorithm 3.2.2. The idea is that first piecewise linear segmentation is applied to the speed time-series of the trajectory, using essentially a one-dimensional variant of the $E_u$ error measure. We call this error measure $E_v$ and define it below for trajectories $T_v$, i.e. those that include speed.

**Definition 3.2.7.**

$$E_v((x_i, y_i, v_i, t_i), (x_1, y_1, v_1, t_1), (x_n, y_n, v_n, t_n)) = \sqrt{(v_i - v'_i)^2},$$

where $(x'_i, y'_i, v'_i, t_1)$ is the point on the line-segment $(x_1, y_1, v_1, t_1)$, $(x_n, y_n, v_n, t_n)$ with time $t_i$.

Essentially we take the difference between the actual speed $v_i$ and the linearly interpolated speed $v'_i$.

To the resulting subtrajectories, which are created in this speed-only segmentation step, we apply regular PLS with an error measure of our choosing: $E_2$, $E_3$, $E_u$ or $E_\mu$. Should we skip the second step, or alternatively just set $\epsilon_p = \infty$, then we have trajectory compression purely based on speed. This is incomplete in the same way as compression with just $E_t$ is. As we did for PLS we indicate with a subscript which error measure is used in the second step, e.g. $2\text{stage-pls}_{E_2}$.

Figure 6 (c) is the speed time-series corresponding to the trajectory from (a) and (b). The graph illustrates that when using compression on the speed time-series, the point with the greatest distance to the line $p1, p5$ is now $p2$ and not $p4$. In other words, if we look at the speed, the stop is now the most important point to keep.

Since speed is derived from the trajectory, the compression in terms of speed that we do here is not entirely accurate. Leaving out a point from a trajectory (because of compression) will change the total displacement of a moving object. This means that, to be more precise, we should compare the speed at a certain point and time to the estimation of the speed at the same time, but with the point left out of the trajectory. This is different from how it is done above, because
we compare the speed at a certain time to a linear interpolation of the speed at that time. However, this linear interpolation is based on start and end speeds that are derived with the point that we are comparing to still part of the trajectory.¹

For higher level reasoning that we will illustrate in Section 3.4, we prefer to combine 2stage-pls with E₂, because this allows us to attach the most interesting semantics to the trajectory. When doing compression with E₃, E₄ and E₅ using regular PLS or using 2stage-pls we have to assume constant speed between points. However, with 2stage-plsₑ₂ we can assume constant acceleration between points. This means that apart from separating stops and moves we can also classify moves in a trajectory as being accelerating or decelerating. Furthermore, there is a clear separation between the compression in the temporal dimension (using speed) and the spatial dimension (using E₂). Thus, we know whether points are kept because they are (temporal) speed change events or because they are (spatial) course change events. However, if we use E₂, then the interpolation of a position for a timestamp between two points that are kept is slightly more complex, since we do not assume constant speed, but constant acceleration.

3.2.5 Error bounds and Complexity

An important point in research into trajectory compression techniques are the error bounds that different error measures give on a number of standard moving object database queries. Regarding these bounds, Gudmundsson et al. (2009) generalize the work of Cao et al. (2006). They show the error bounds for their E₅ measure for the Where-At and When-At queries and how these are dependent on µ. It is easy to see that if we use 2stage-plsₑ₅, we have the same error bounds,

¹We have also experimented with reestimating the speed, but found this not to make any significant differences in our current experiments. However, it does increase computation time significantly.
because for every subtrajectory created by the speed compression, these error bounds hold since these subtrajectories are compressed using $E_\mu$.

The error bounds for $E_\mu$ vanish when $\mu = 0$ and consequently the bounds for $2\text{stage-pls}_{E_2}$ are not good. The error for the first speed step is bounded. However, an error in speed leads to a quadratic error in position. Position is not further bounded, since $E_2 = E_{\mu=0}$.

The worst case running time of standard piecewise linear segmentation is $O(n^2)$. The $2\text{stage-pls}$ algorithm is in the same order of complexity, because it is essentially two applications of PLS. The first (speed) step is just a regular full recursion of PLS and we can consider the second step as if we are already at a certain level of the recursion of PLS.

### 3.3 Evaluation

Typically the performance of a trajectory compression algorithm is judged by the reconstruction error introduced when recreating the original trajectory from the compressed trajectory. One way to do this is to prove query error bounds (Cao et al. (2006); Gudmundsson et al. (2009)). An empirical option is to compute the distance between the compressed trajectory and the original trajectory for each point in time (Meratnia and de By (2004)). As we mentioned in the previous section, our method has the same error bounds as regular PLS with $E_\mu$, so we do not consider reconstruction error in our evaluation.

The goal of our two-stage method is to be better at retaining stop information than the traditional error measures while still providing good compression rates. Or to put it differently, our method should provide better stop retention at the same compression rate. This is what our evaluation below focuses on. To show this, we use a dataset from our set of AIS vessel trajectories, described in more detail in Chapter 2. AIS contains speed information about the vessel. However, this is sometimes based on a different sensor, and furthermore is not always available in other applications. To show the more general applicability of the above method we have also reconstructed speed, based on the position information alone. To reduce the influence of noise in the AIS data, caused by for instance GPS, we apply some standard moving average smoothing\(^2\) to the trajectory and also to the recomputed speed.

#### 3.3.1 Stop Retention

As was mentioned earlier, the definition of a stop is application dependent. In this application we consider a ship to be stopped whenever its movement speed is (close to) zero for some (short) amount of time.

\(^2\) Simple moving average, such as the standard MatLab `smooth` function.
of time. This means that a ship can stop in port but also at sea or in an anchoring area. With this in mind, we selected trajectories from our dataset that appeared to have (a lot of) stops in them. We plotted these trajectories on a map and hand-labeled the stops in the trajectory, indicating them by a start and end time. This stop dataset, for instance, contains a ferry between two ports and a cargo ship docking in a harbor.

Next, we need to define when a compressed trajectory has successfully retained a stop, which we do as follows. If a compressed trajectory $T_C$ contains a segment $T_C(i, i + 1)$, such that

$$\frac{\text{dist}(\langle x_i, y_i \rangle, \langle x_{i+1}, y_{i+1} \rangle)}{(t_{i+1} - t_i)} \leq \theta,$$

then that segment is a stop. In other words, a vessel is stopped if the average speed in a segment is below a certain threshold. If the temporal interval $t_i, t_{i+1}$ overlaps with the temporal interval of a hand-labeled stop (i.e. they have at least one time-point in common), then we say that the compression has retained that stop. The stop-threshold $\theta$ should at least be as high as the maximum speed among the hand-labeled stops. Note that a stop can only be retained once, e.g. if there are two segments below the threshold that overlap with one labeled stop, then this is counted as one. This also holds the other way around, if there are two hand-labeled stops that overlap with one segment, then we also count this as one. Furthermore, we define the stop retention ratio as the total number of retained stops divided by the total number of hand-labeled stops. Apart from stops that are not retained, the opposite can also occur. The average speed in a segment can be below the stop threshold, but it is not labeled a stop. We do not consider this option in the stop retention ratio definition, since it did not occur in our experiments.

What a good stop threshold is in our application is not immediately obvious, therefore we consider stop retention at three different stop thresholds. Because the data is noisy and hand-labeling is far from perfect, the number of hand-labeled stops that fall below these thresholds differs slightly. At stop threshold $\theta = 0.1$ knots we have 291 stops, at $\theta = 0.05$ knots we have 288 and at $\theta = 0.025$ knots there are 281 stops. Thus, for each threshold the total number of stops that should maximally be retained differs.

3.3.2 Experimental Set-Up

We are interested in increasing stop retention while maintaining compression rate. Thus, for the two algorithms, the different error measures and a range of parameter settings, we compute the stop retention for the hand-labeled stop dataset. Clearly, this set is biased towards trajectories with stops. To get a more general compression
result for our data we compute the compression rate for the different settings on a regular unbiased dataset containing 600 randomly selected trajectories.

As baseline methods we will look at the regular pls algorithm with the error measures: $E_2, E_3, E_u, E_{\mu=\frac{1}{2}}, E_{\mu=2}, E_t$ and $E_u \wedge E_t$, essentially what is defined in Section 3.2.2. We recall that the settings $E_{\mu=0,1,\infty}$ are equal to $E_2, E_3$ and $E_u$, respectively. $E_{\mu=\frac{1}{2}}$ is in between $E_2$ and $E_3$, and $E_{\mu=2}$ is in between $E_3$ and $E_u$.

Before any of these measures can be applied, the time dimension needs to be scaled to the spatial dimensions.\(^3\) The intuition between the measure $E_3 / E_{\mu=1}$ is that differences in the temporal and spatial dimensions are equal. Thus, we have done the scaling of the temporal dimension such that the average temporal difference between samples, in the original scale of the temporal dimension, equals the average difference in position, under the $E_3 / E_{\mu=1}$ setting.

All of the above baselines are compared to our two-stage method $2$stage-pls. We combine $2$stage-pls with the error measure $E_2 / E_{\mu=0}$, which we like for semantic reasons, and also with the $E_\mu$ setting that scores the best among our baselines. Furthermore, we consider the situation where we take the speed information directly from AIS and the scenario where we recompute the speed.

Finally, we consider other intuitive variants that use two error measures, like $2$stage-pls. The first variant is using the $E_t$ error measure as the first step in $2$stage-pls, because $E_t$, like $E_v$, is a candidate for being a good stop detector. We test this variant with $E_2$ and the same $E_\mu$ setting as above. We also create two variations on pls with $E_u \wedge E_t$: $E_2 \wedge E_v$ and $E_\mu \wedge E_v$. As the last alternative we change $2$stage-pls so that we reverse the two compression steps, i.e. we do compression with $E_2$, respectively $E_\mu$, first, and do the compression with $E_v$ in the second step.

For all experiments, the $\epsilon$ settings that we range over are the same. For the error measures $E_2, E_3, E_u$ and $E_{\mu,\nu}$, $\epsilon$ ranges from 0.01 to 0.08 kilometers with increments of 0.01. For $E_t$, $\epsilon$ ranges from 90 to 720 seconds, with increments of 90. Finally, for $E_v$, $\epsilon_v$ ranges from 0.5 to 4 knots, with increments of 0.5.

### 3.3.3 Results

Because of the large number of different settings that we tested, we only present the most important results of the experiments in the following section. For a complete overview of all the results we refer the reader to Appendix A.

From the baselines we only show the results for the $E_{\mu=2}$ setting, the best performing setting in terms of stop retention versus compres-

\(^3\) Typically the time is in seconds, and the spatial dimension in arc lengths or kilometers.
Figure 7: Results of $\text{zstage-pls}_{E_2}$ under different experimental settings. Results for $\text{pl}_{E_{\mu-2}}$ are included for reference.
Figure 8: Results of $\text{2stage-pls}_{E_{\mu=2}}$ under different experimental settings. Results for $\text{plS}_{E_{\mu=2}}$ are included for reference.
sion rate. Furthermore we give the results for 2stage-pls with $E_2$ and with $E_{\mu=2}$. The other variants using two error measures are outperformed by these two 2stage-pls variants and we therefore do not give the results, but instead refer to Appendix A.

Figures 7 and 8 shows results for 2stage-pls with $E_2$, and 2stage-pls with $E_{\mu=2}$ respectively, for a number of settings. Each figure also contains the result for regular pls with $E_{\mu=2}$ for reference. The graphs plot stop retention on the vertical axis versus compression rate on the horizontal axis. Each graph shows results for one setting of the stop threshold $\theta$ (vertically) and recomputed speed (left) or AIS speed (right). Ideally, we have good stop retention with high compression rate, which we find in the top right corner of the graphs. Each line in the graph is created by running the algorithm and error measure combination for the range of $\epsilon$ that we defined above. However, 2stage-pls has two parameters, thus there are 8 lines, representing the different settings of $\epsilon_v$. For each figure it holds that the highest line in the graph is for $\epsilon_v = 0.5$, the lowest line for $\epsilon_v = 4$ and the rest of the settings are in between.

3.3.4 Discussion

We first consider the performance of 2stage-pls$_{E_{\mu=2}}$. From Figure 8 we see that this variant of 2stage-pls outperforms regular pls$_{E_{\mu=2}}$ under almost all parameter settings. Thus, what we do with two stage PLS, adding an initial speed compression step to the best performing baseline, is a good idea. If we do this, we can retain more stops at the same compression rate.

The results in Figure 7 for our semantically motivated method 2stage-pls$_{E_2}$ show that we do not outperform our best baseline for all parameter settings. However, because 2stage-pls has two parameters we can tune our method as to how well we want to detect stops and how much spatial error we allow.

What is also interesting is that the performance of both 2stage-pls$_{E_{\mu=2}}$ and 2stage-pls$_{E_2}$ does not degrade much when we lower the stop threshold. On the other hand our baseline $E_{\mu=2}$ does perform quite a bit worse with a stricter stop threshold. This suggest that the added speed compression step of 2stage-pls adds robustness when it comes to stop retention.

Furthermore, stop retention is better using the speed coming from the AIS sensor. One reason for this seems to be that recomputing the speed from the trajectory is not trivial business, since the GPS used by AIS creates quite some noise, which is amplified in the speed recomputation. This is something to consider for applications where the speed of the moving object is not directly available.

Though we have introduced a rather general definition of a stop there are also other possibilities. Another definition of a stop might
make a difference in performance. For instance, one could define a stop as a point, instead of a segment of a trajectory, for which the speed has gone below the stop threshold. Such a definition might be more favorable to our approach. However, we have not investigated this.

Finally, we note that using piecewise linear segmentation leads to a large data reduction with compression rates over 90% for the realistic settings that we used.

3.4 DERIVING COMPLEX BEHAVIOR

In this section we show how we create atomic movement events that we call segments using the compression method described above. These segments are represented in the Simple Event Model (SEM) (van Hage et al. (2011)). Additional concepts are added from external sources and via reasoning, and are also represented in SEM. Using these extra concepts we can derive more abstract, and complex vessel behavior, which we will illustrate with an example. In this section we first briefly describe the semantic web and then the simple event model; for more details on this we refer to the papers (Willems et al. (2010); van Hage et al. (2011)). Secondly we discuss how the simple event model is combined with the segments.

3.4.1 Semantic Web

In 2001 the concept of the Semantic Web was introduced (Berners-Lee et al. (2001)). The aim of the semantic web is to create a network of interlinked data that is readable for machines, as opposed to the old world wide web, which is understandable for humans. This development has brought new standards such as the Resource Description Framework (RDF) and the Web Ontology Language (OWL).

RDF is the foundation for knowledge representation, i.e. ontologies, on the semantic web and is based on the idea of making statements about resources in a subject-predicate-object form. Such expressions are dubbed triples. The RDF specification defines a number of classes, for the subjects and objects, and properties, for the predicates. Moreover, users can, and should add their own classes and objects. For example, suppose that we have an ontology about fruits, called fruit. Then the RDF triple fruit:Pear:rdfs:isSubClassOf:fruit:Fruit expresses the fact the the class of Pear is a sub-class of the class of Fruit. And the triple fruit:elstar: rdf:type:fruit:Apple expresses the fact that an elstar is an instance of the class Apple. The colon notation is used to indicate from which ontology a class or property is used, i.e. the class Pear comes from the example fruit ontology that we created ourselves, but the property type is defined by the RDF standard. These notations are shorthands for full-fledged Universal Resource Identifiers (URI) that
40 Trajectory Compression

uniquely identify the specific ontology used and where to find it, thereby forming the backbone of the semantic web. The triple format of RDF is essentially a labeled directed multi-graph and pieces of RDF are often represented as graphs in figures for better readability. A lot more on RDF and the semantic web can be found online.4

3.4.2 Simple Event Model

The simple event model (SEM) (van Hage et al. (2011)) was designed to represent events, in the broad sense of the word, derived from various sources, such as, the web, sensor data, etc. SEM has four core classes: sem:Event (what happens), sem:Place (where did it happen), sem:Actor (who or what participated), and sem:Time (when did it happen). Each core class has an associated sem:Type that is used to indicate the type of a core individual. For instance, the core class sem:Place has the sem:PlaceType class associated to it.

SEM has three kinds of properties: sem:eventProperties, sem:Type properties, and a few miscellaneous properties like the sem:hasTimeStamp’s subproperties. The sem:eventProperties relate sem:Events to other individuals. They also contain the sem:hasSubEvent property to aggregate events. A sem:Type relates individuals of the core classes to individuals of sem:Type. There are a number of sem:hasTimeStamp properties, among them: sem:hasBeginTimeStamp and sem:hasEndTimeStamp for time intervals.

3.4.3 Segments

We use 2stage-pls to create segments that are represented using SEM. Let $T_{vc}$ be a trajectory as defined earlier, which next to speed $v$ also includes course $c$. We compress this trajectory into

$$T_C = \text{2stage-pls}_{E_2}(T_{vc}, \epsilon_v, \epsilon_p),$$

where $\epsilon_v = 2.5$ knots and $\epsilon_p = 50$ m are chosen such that we have high compression, but also good stop retention, according to Figure 7. For all $i$ such that $T_C(i) = (x_i, y_i, v_i, c_i, t_i)$ and $T_C(i + 1) = (x_{i+1}, y_{i+1}, v_{i+1}, c_{i+1}, t_{i+1})$ we create a segment. A segment is an instance of sem:Event. It furthermore has the additional sem:eventType etype_stopped or etype_moving, indicating whether the segment is a stop or a move, determined as described in Section 3.3.1 with a stop threshold $\theta = 0.1$. Event information about the speed along the segment, i.e. whether it is constant, increasing or decreasing can also similarly be added as an event type. The coordinates $x_i, y_i$ and $x_{i+1}, y_{i+1}$, converted back to latitude and longitude, are represented as instances of sem:Place which are connected to the segment event by the prop-

4 http://www.w3.org/standards/semanticweb/
3.4 Deriving Complex Behavior

Properties `seg:hasBeginPlace` and `seg:hasEndPlace`, subproperties of `sem:hasPlace`. The segment also has the properties `sem:hasBeginTimestamp` and `sem:hasEndTimestamp`, `sem:hasBeginSpeedOverGround` and `sem:hasEndSpeedOverGround`, and `sem:hasBeginCourseOverGround` and `sem:hasEndCourseOverGround`, which take the values $t_i$ and $t_{i+1}$, $v_i$ and $v_{i+1}$, and $c_i$ and $c_{i+1}$, respectively. In Figure 9 the RDF representation of a segment is illustrated by the white boxes.

Under the current compression settings we have a compression of around 98%, thus a data reduction by a factor of 50. This data reduction is important to make the reasoning that we will see in the rest of the section feasible.

3.4.4 Additional Semantics

In order to derive more complex behavior patterns extra semantics for the segments are needed, such as the type of place a vessel goes to, and the type of the vessel itself.

**Places** By using spatial proximity, the latitude and longitude coordinates of the `sem:Place`s are connected to concepts in the GeoNames ontology. The GeoNames ontology associates instances of places with geo-coordinates to GeoNames feature codes like `geo:H.HBR` (harbor), and `geo:H.ANCH` (anchorage). If such a GeoNames concept is close, we add a `sem:Place` instance for this concept and relate it to a segment using a `seg:hasBeginPlace` or `seg:hasEndPlace` property. This allows us, for instance, to add the additional `sem:eventType` `etypeStoppedInHarbor` for segments that are of type `etypeStopped` and for which the related place has the type `geo:H.HBR`. The addition of this type of information to the segments is illustrated in Figure 9 by the red (C) boxes.

**Actors** Each segment belongs to a single vessel. This vessel is identified as the `sem:Actor` of the segment, related by the `sem:hasActor` property. The vessel has an `ais:mmsi` property to the value of its Maritime Mobile Service Identity (MMSI) number. Using this MMSI number additional information about the vessel is pulled from different web sources and new properties, like `ais:length` and `ais:callsign`, of the vessel are introduced. Also type information is added, which is aligned to WordNet. For instance, the type “passenger vessel” would be translated to `atypePassengerVessel`, which is aligned to `wordnet:synset-passenger_ship-noun-1`. Extra actor information is shown in Figure 9 by the yellow (D) boxes.

**Deriving Complex Events** With the extra semantic information in place we can derive more complex behavior than the simple

---

5 http://www.geonames.org
6 http://www.wordnet.org
are concepts inferred using rules.

basic segments. The red (C) boxes illustrate added place semantics, the yellow (D) boxes are added actor semantics, and the blue (E) boxes spatially inferred by proximity.

Figure 9: An RDF-graph example of part of a ferry trip. Green (A) and purple (B) boxes are SEM core classes. The white boxes are concepts from the

TRAJECOTRY COMPRESSION
etyp_stopped and etyp_moving events. This is done using rules. For instance, to derive the complex behavior “trip” we use a rule that is based on the assumption that if we do not know that an explicit stop between consecutive moving events exists, then it does not exist. This closed world assumption allows us to deal with missing ship observations, which occur frequently. We conclude with this rule, that if we do not know about a stop at a harbor between two stops at harbors a and b, then there was a trip between harbors a and b. This trip is added as a new event, which has sem:hasSubEvent property relations to the segments that compose the trip. The harbors of departure and arrival become sem:hasBeginPlace and sem:hasEndPlace properties of the trip. Subsequently we construct a rule that defines a “ferry trip” as a trip back and forth between two different harbors, allowing us to recognize ferries. All the information inferred using rules is illustrated in Figure 9 by the blue (E) boxes.

An important advantage of having different levels of abstraction of events is that for the higher level events it does not matter where the subevents were derived from. For instance, one subevent stopped_at_harbor of the trip event could be derived as described above, whereas another could be derived from another source, like the ferry schedule on the web. This would still allow the derivation of the ferry trip event.

3.5 CONCLUSIONS & FUTURE WORK

We presented a simple extension to standard trajectory compression based on Piecewise Linear Segmentation (PLS). The idea of our compression method is to add an initial compression step based on the speed of the moving object. Afterwards we apply compression with a spatial error measure of our choosing. The motivation for introducing this method is that standard error measures and PLS are not ideally suited for stop retention.

For most parameter settings, this two-stage method allows us to retain more stops at the same (high) compression rate than regular piecewise linear segmentation using any of the error measures. Furthermore, the two-stage method allows for better tuning of the retained stops versus compression rate trade-off. The method is also more robust in terms of stop retention, using stricter stop thresholds decreases stop retention only slightly whereas this is not the case with regular PLS.

Furthermore, we saw examples of how we can use compression to create atomic movement events in terms of the Simple Event Model (SEM). The representation in terms of SEM allows us to incorporate more domain knowledge and do higher level reasoning about vessel behavior, leading to rules recognizing, for example, ferry movements.
In future work we would like to compare to the generic segmentation framework presented by Buchin et al. (2010), which we unfortunately only became aware of after research on this chapter was complete.