A Trajectory Segmentation Algorithm Based on Interpolation-based Change Detection Strategies

Mohammad Etemad
Institute for Big Data Analytics
Halifax, NS
etemad@dal.ca

Amílcar Soares
Institute for Big Data Analytics
Halifax, NS
amilcar.soares@dal.ca

Arazoo Hoseyni
Institute for Big Data Analytics
Halifax, NS
a.hoseyni@dal.ca

Jordan Rose
Institute for Big Data Analytics
Halifax, NS
jordanrose@dal.ca

Stan Matwin
Institute for Big Data Analytics
Polish Academy of Sciences
Halifax, NS
stan@cs.dal.ca

ABSTRACT

Trajectory mining is a research field which aims to provide fundamental insights into decision-making tasks related to moving objects. One of the fundamental pre-processing steps for trajectory mining is its segmentation, where a raw trajectory is divided into several meaningful consecutive sub-sequences. In this work, we propose an unsupervised trajectory segmentation algorithm named Octal Window Segmentation (OWS) that is based on the processing an error signal generated by measuring the deviation of a middle point of an octal window. The algorithm we propose is flexible and can be applied to different domains by selecting an appropriate interpolation kernel. We examined our algorithm on two datasets from different domains. The experiments show that the proposed algorithm achieved more than 93% of a cross-validated harmonic mean of purity and coverage for two different datasets. We also show that statistically significantly higher results were obtained by OWS when compared with a baseline for unsupervised trajectory segmentation.

1 INTRODUCTION

Processing traces of people, vehicles, vessels, and animals have been the focus of attention in the academic and industry sectors. These traces of moving objects are called trajectory data and can be informally defined as a consecutive sequence of the geolocations of a moving object. Transportation mode detection [6], fishing detection [4], tourism [7], environmental science [18], and traffic dynamics [2, 5, 20], are few examples of domains where trajectory mining methods can be applied.

One of the fundamental trajectory mining tasks is segmentation, i.e., split raw trajectories into sub-trajectories. Trajectory segmentation is a fundamental task since the method influences the features representing each trajectory. An accurate segmentation method may provide higher quality features that better represent the moving object behavior. The segmentation task is therefore based on methods capable of distinguishing the homogeneous or similar parts of a trajectory based on some criteria. Three cases can be distinguished: supervised, unsupervised, and semi-supervised trajectory segmentation. Unsupervised methods use only the raw trajectory as input, while supervised methods use labels available in a training data to extract some knowledge and use this knowledge as criteria to generate the sub-trajectories. Finally, semi-supervised methods use a combination of both labeled and unlabeled data as a criterion. Although efforts to create labeled trajectory datasets [8, 23] can be found in literature, the majority of them do not contain such information. Therefore, this work focuses on the development of unsupervised methods for trajectory segmentation.

Since trajectory data is usually large and has all the characteristics of Big Data, i.e., volume, velocity, variety, veracity, and value, presenting a fast and accurate segmentation method is of prime importance. In this research, we investigate the topic of trajectory segmentation and propose an unsupervised segmentation method that can generate high-quality segments. The intuition behind our approach is that when a moving object changes its behavior, this shift may be detected using only its geolocation over time. Unlike, previous methods that uses speed variations [14], direction variation [15], or a combination of many features [8, 9, 11, 13, 16], this work focuses on finding these changes in behavior only from the object's coordinates using interpolation methods to generate an error signal. This error signal is then used as a criterion to split the trajectories into sub-trajectories. Our method can be customized to a domain by using different kernel interpolation methods. The contributions of this paper are (i) the proposal of an unsupervised trajectory segmentation method named OWS, (ii) a comparison of OWS and a baseline regarding performance and execution time, and (iii) a comparison of different kernel interpolations for datasets of different domains.

The rest of this paper is organized as follows. We review the algorithms for trajectory segmentation and interpolation in Section 2. In Section 3, the definitions used through this paper, and the OWS algorithm are detailed. The experiments (e.g., metrics, dataset, hyperparameter tuning, and results) and its analysis are detailed in Section 4. Finally, Section 5 provides conclusions obtained from this work and future work that may be conducted.

2 RELATED WORK

Trajectory segmentation methods such as TRACLUS [10], W-KMeans [11], SMOT [1], CB-SMoT [14], GRASP-UTS [16], and RGRASP-SemTS [9] have been proposed to segment trajectory data. These methods are briefly reviewed in Section 2.1. Since the proposed method can be customized for a domain by selecting different interpolation methods as kernel, we reviewed some major interpolation methods in Section 2.2.

2.1 Trajectory segmentation

Trajectory segmentation methods can be divided into three categories regarding the input data of the algorithm: (i) unsupervised, (ii) supervised, and (iii) semi-supervised.

Unsupervised methods use only raw trajectory data as input and compute a set of features from it. This family of methods considers the similarity among features in the neighborhood of a sequence to create a set of sub-trajectories [11, 16, 21]. *Supervised* methods use labels available in training data to extract some knowledge and use it as criteria to generate the sub-trajectories [3, 14, 23]. Finally, *semi-supervised* methods use a combination of both labeled and unlabeled data as a criterion. The RGRASP-SemTS is an example of such method [9].

A trajectory segmentation can use a cost function or clustering methods to create sub-trajectories. GRASP-UTS, RGRASP-SemTS, and W-KMeans are examples of cost function based methods, while TRACLUS, SM0T, and CB-SMoT are examples of clustering based methods.

A quantitative comparison between some of the aforementioned methods is given in [16]. They reported higher performance for GRASP-UTS in comparison to W-KMeans, and CB-SMoT. The highest segmentation performance shown in [16] was a purity of 91.37% and coverage of 83.00% on the fishing vessels dataset, and purity of 90.57% and coverage of 83.47% on the hurricanes dataset (to be detailed in Section 4.1). In this work, we repeated the experiments in the same environment and showed that our proposed method obtained better results when compared to GRASP-UTS.

2.2 Interpolation

Sometimes it is necessary to resample the frequency of trajectory data due to signal loss. Calculating the geolocation for a time-stamp that the geolocation is missing called *interpolation*. Different methods such as linear, random walk, bézier curve, catmull-row, and kinematic path have been introduced to calculate the geo-location of these missing points. An interpolation method can be useful for one domain and useless for others. For example, *random walk* interpolation can be useful to interpolate wild animal behavior [17]. The *bézier curve* interpolation can be useful for moving objects in fluid environments [19]. The *hermite and spline* interpolation can be useful for AIS data(trajectories of vessels) [22] and *kinematic* interpolation is useful for transportation [12] or fast moving objects. *Linear* interpolation is the simplest, popular interpolation method. In this method, the missing location calculated so that it is sitting on a straight line between two available points. *Cubic* and *Kinematic* methods calculate the speed and acceleration of the moving object in each point of the octal window to interpolate the missing position. We implemented random walk, kinematic, cubic, and linear interpolation to utilize as a kernel for the proposed segmentation algorithm with the objective of exploring their results into different trajectory datasets.

3 THE TRAJECTORY SEGMENTATION METHOD

In this section, we detail our novel algorithm for unsupervised trajectory segmentation named Octal Window Segmentation (OWS). We first introduce the definitions used to describe the algorithm (Section 3.1). After, we detail OWS algorithm step by step in Section 3.2.

3.1 Definitions

A *trajectory point* (p_i) is defined as $p_i = (x_i, y_i, t_i)$, where x_i is longitude, y_i is latitude, and t_i ($t_i < t_{i+1}$) is the capturing time of the moving object. A *raw trajectory* (τ_n), is a sequence of trajectory points captured through time, $\tau = (p_i, p_{i+1}, .., p_n), p_i \in \tau_n$ and $i \leq n$.

A *segment* or *sub-trajectory* is a subsequence of a raw trajectory generated by splitting it into two or more sub-sequences. For example, if we have one split point, k, and τ_n is a raw trajectory then $s_1 = (p_i, p_{i+1}, ..., p_k)$ and $s_2 = (p_{k+1}, p_{k+2}, ..., p_n)$ are two sub-trajectories generated from τ_n. The process of generating sub-trajectories from a raw trajectory is called *segmentation*.

An *octal window* (S_{ow}) is a sub-trajectory with seven trajectory points, in which new trajectory points are created using interpolation techniques. We define $S_{ow} = (p_1, p_2, p_3, p_4, p_5, p_6, p_7)$ so that p_i is time-ordered. The indexes are relative for each window so that it can slide over a raw trajectory and represents different windows.

The decision of using seven trajectory points on a window was motivated by the fact that it is necessary to use at least three points to interpolate (predict) a trajectory point preceding or following them. Calculating acceleration in kinematic interpolation requires at least three points. Since we use interpolation going both forward and backward, we have used three points in the beginning of the window to predict forward the position of the fourth point, and three points in the end of the window to predict backward another fourth point, expected to be very close to the result of the forward interpolation (see details in Section 3.2). We then use these two interpolated positions to create a midpoint and calculate its geographical distance from p_4. Since we have two sets of four points each, the minimum number of required trajectory points is seven points. Increasing the length of octal window can possibly improve the results; however, the objective of this work is to use minimal possible memory. We use the term *current octal window* to refer to the window being processed by our procedure at a given moment. After processing a window, we slide the trajectory by one point and process the next window.

3.2 Octal Window Segmentation Algorithm

The intuition behind our algorithm is that when a moving object changes from one behavior to another, this can be captured directly from its geolocation. To achieve an estimated position, where the moving object is supposed to be if its behavior does not change, we use interpolation methods. After, we compare the real position of the moving object with the estimated one, creating an error signal. By evaluating this error signal, it is possible to estimate if the moving object changed its behavior on a region and use this information to create sub-trajectories.

The first procedure that composes OWS unsupervised trajectory segmentation algorithm is detailed in Algorithm 1. This procedure creates an error signal by sliding the octal window over a raw trajectory τ_n.

The procedure starts with an array of Error signals (E) in line 1. In line 2, the empty signal set $[0, 0, 0]$ is added to the list and represents the error for the first three points from the raw trajectory. The algorithm explores all the octal windows from lines 3 to 10 as follows. First, the actual octal window is created (line 4). The *forward interpolation* is calculated in line 5. In this method, we assume that p_i in the current octal window is missing and will be interpolated using points p_1, p_2, p_3. The interpolated point at time $t^i = t_3 + \frac{t_5 - t_3}{2}$ is called p^F. After, the *backward*

Algorithm 1 Generate Error Signal

Require: τ_n - the raw trajectory

1: $E \leftarrow \{\}$
2: $E.append([0, 0, 0])$
3: **for** $(i = 3; i < n - 3; i++)$ **do**
4: Create octal window $S_{ow} = (p_{i-3}, ..., p_{i+3})$
5: $p^F \leftarrow$ interpolate forward S_{ow}
6: $p^B \leftarrow$ interpolate backward S_{ow}
7: $p^C \leftarrow$ extract midpoint from P^F and P^B
8: $\epsilon_i \leftarrow Haversine(p_i, p^C)$
9: $E.append(\epsilon_i)$
10: **end for**
11: $E.append([0, 0, 0])$
12: **return** E

interpolation method is calculated (line 6). In this method, it is also assumed that p_i in the current octal window is missing. However, we reverse the order of points so that points p_7, p_6, p_5 are used to interpolate the point p_i at time $t^i = t_5 - \frac{t_5 - t_3}{2}$ and the procedure calls it p^B. In line 7, we use p^F and p^B geolocations to calculate a midpoint (p^C). The error signal ϵ_i is finally computed in line 8, and it is obtained by calculating the haversine distance between p_i and p^C.

Figure 1: An example of an error signal calculation for an octal sliding window S_{ow}.

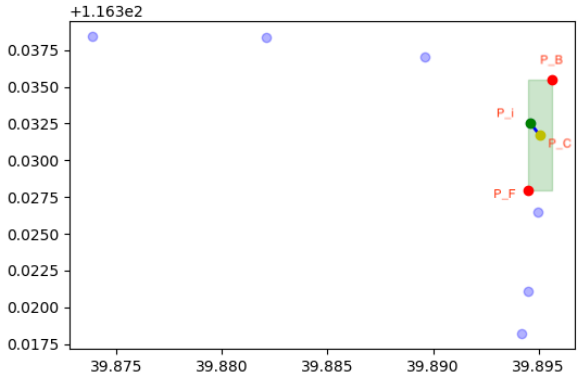
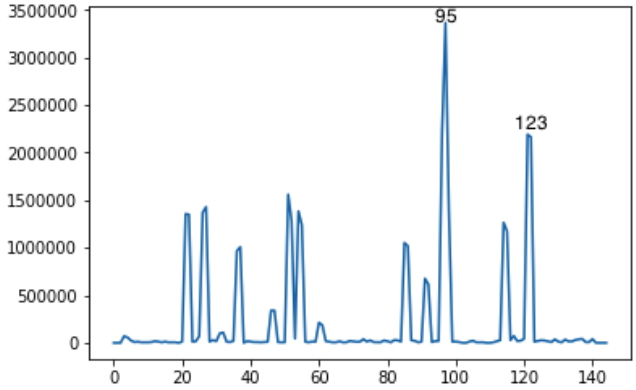


Figure 1 shows the p^B and p^F interpolated positions as red points, p_i as a green point and p^C as a yellow point. In the example of Figure 1, the haversine distance from the estimated position p^C to the real position p_i is visible. This may indicate that the moving object behavior has changed at position p_i.

In Figure 2, an example of an error signal generated by Algorithm 1 is shown. A raw trajectory with around 150 trajectory points was used in this example. As can be seen in Figure 2, there are several trajectory points (e.g., around trajectory point 95, or around trajectory point 123) along the raw trajectory where the estimated positions were far from the actual reported positions by the moving object.

The OWS algorithm is detailed in Algorithm 2 which receives as input a single ϵ value. The intuition of how this algorithm works is that segments are created in partitioning positions where the error values from E are higher than the ϵ value and these partitioning positions are created as a list of tuples with the indexes of where segments start and end. Algorithm 2 starts

Figure 2: An example of an error signal calculation for an octal sliding window S_{ow}.



Algorithm 2 Octal Window Segmentation

Require: ϵ min. error value to split a trajectory

1: $E \leftarrow$ Generate Error Signal (τ_n)
2: $first \leftarrow 0$
3: $q \leftarrow [(first, n)]$
4: $p \leftarrow \emptyset$
5: **while** $q \neq \emptyset$ **do**
6: $t \leftarrow q.pop()$
7: $curr \leftarrow E[t[0] : t[1]]$
8: $m \leftarrow max(curr)$
9: **if** $m > \epsilon$ **then**
10: $idx \leftarrow index(curr == m)$
11: **if** $len(idx) == 1$ **then**
12: $q.append((t[0], t[0] + idx[0]))$
13: $q.append((t[0] + idx[0] + 1, t[1]))$
14: **else**
15: $ixx \leftarrow group\ index(idx)$
16: **for all** $g \in ixx$ **do**
17: $q.append((first, ixx[g]))$
18: $first = ixx[g]$
19: **end for**
20: $q.append((first, t[1]))$
21: **end if**
22: **else**
23: $p.append(t)$
24: **end if**
25: **end while**
26: **return** p

creating the error signal E that is the output of the procedure in Algorithm 1. In lines 2 and 3, the algorithm initializes the $first$ variable with a 0 value which represents the starting index of the trajectory and creating the first tuple $(first, n)$ that represents the entire trajectory and adding it to a list q. In line 4, the final variable p with all the partitioning positioning tuples is declared as an empty. While the list q is not empty, lines 6 to 24 are executed. First, this algorithms get the first element of the list t (line 6), which in the first run is the full trajectory, creates a list $curr$ with all the error values from E (line 7), and gets its maximal error value m (line 8). If this maximal error value is greater than the threshold, the index of m is retrieved, and two new tuples are created if there is a single position with value m (lines 11 to

13). The new tuples are stored in q and are analyzed in the next iteration of the algorithm, which will look for other error values higher than the ϵ threshold. If there is more than one partitioning position with a value equal to m (lines 14 to 21), tuples are created in every single position that satisfies this criterion. This procedure will run until all the tuples with partitioning positions are created where error values are greater than the error threshold ϵ. In the last step, i.e. if $m \leq \epsilon$, tuple t is appended to the final list p.

4 EXPERIMENTS

This section details the metrics and datasets (Section 4.1) and algorithms parameter selection (Section 4.2) procedure. Finally, the interpolation methods analysis obtained with the OWS algorithms detailed in Section 4.3 and Section 4.4 shows a comparison between our OWS strategy and a baseline segmentation algorithm.

4.1 Metrics and datasets

Since our method is classified as an unsupervised method, clustering metrics such as purity, coverage, and the harmonic mean of purity and coverage are proper evaluation metrics. In this work, we have used the metrics named *average purity* and *average coverage*. They were first introduced in the context of trajectory segmentation in the work of [16]. These two metrics were designed to be orthogonal, i.e., when one tends to increase, the other tends to decrease. Therefore, we defined the *harmonic mean* of average purity (P) and average coverage (C), *harmonic mean (H)*, $H = \frac{2*\overline{P}*\overline{C}}{\overline{P}+\overline{C}}$, as the primary metric for our analysis and to simplify the plots and comparison of the segmentation algorithms.

The segment $purity$ in a segment is defined as follows. Assuming the set of all target labels in a segment is L with k trajectory points. The majority label, $p^d \in L$, is the label of majority trajectory points in the segment and the number of occurrence of p^d is p. Therefore, the purity of a segment is $\frac{p}{k}$. The average of purity values for all segments generated by a trajectory segmentation algorithm is called *average purity*, $\overline{P}$. The *coverage* of a segment can be calculated using a segment identifier (s_{id}) from the segments found by the segmentation algorithm. Assuming σ_m is a segment that was supposed to be found by a segmentation algorithm, it is possible to verify for every segment found by the algorithm the most frequent s_{id} by $\frac{s_{id}}{m}$. The average for coverage of all generated segments is then called $\overline{C}$. The more over segmented a trajectory is, higher values of purity are expected to be found. However, lower values for coverage will be computed in the case of a large number of segments found by the segmentation algorithm. The same conflicting result occurs if the trajectory is under segmented, i.e., the purity values tend to decrease, but the coverage tends to have a higher value.

Two datasets were selected for evaluation of OWS and the baseline named GRASP-UTS: (i) fishing (5190 points, 153 segments) and (ii) hurricane datasets (1990 points, 182 segments). The dataset was processed using the same conditions and features adopted in the experiments of [16]. The objective was to achieve the best result reported by GRASP-UTS for the unsupervised trajectory segmentation problem.

4.2 Algorithms parameter selection

In the experiments conducted in this work, ten different trajectory subsets were created aiming to properly evaluate the performance of the segmentation algorithms. We have used one subset for estimating the input parameters values of both algorithms, and the remaining nine to verify the algorithm's performance in terms of the harmonic mean of average purity and coverage. The same process was repeated for every single subset as the set for input parameters value estimation, and validation in the remaining subsets. As a result, ten different values of the harmonic mean of average purity and coverage were found in our experiments.

The input parameter values estimation for GRASP-UTS was done by a grid search with all combinations of values reported in [16]. The decision of the best input parameters combination was guided by the best cost function value achieved by an algorithm configuration, in the same way, reported in [16].

For the OWS segmentation algorithm, the ϵ value was found using the following steps. First, the total error signal E was generated for the one subset for parameters estimation. After, the harmonic mean of the purity and coverage was calculated by running OWS and using values of percentiles (P) from E. We tested the percentiles values for every $\epsilon \in E$ from 99 to 90 ($P = [P_{99}, P_{98}, P_{97}, P_{96}, P_{95}, P_{94}, P_{93}, P_{92}, P_{91}, P_{90}]$. The percentile that produced the highest harmonic mean was chosen to be used as the ϵ value and was used to estimate the harmonic mean in the remaining nine subsets.

4.3 OWS interpolation methods evaluation

In the first experiment, we tested the kinematic, linear, random walk, and cubic interpolation methods in OWS for the hurricanes and fishing datasets.

The results on fishing dataset show that random walk interpolation produces the highest harmonic mean. Since we do not have enough samples (10 harmonic mean values for every segmentation algorithm) to verify if the outcomes are normally distributed, we have used the Mann Whitney U test to verify if the difference in the results is significantly different. If P was lower than 0.05, we rejected the hypothesis that the observed median values came from the same distribution, so there are statistical differences. A Mann Whitney U test indicated that the random walk interpolation kernel produces statistically significant higher median (M = 93.68) harmonic mean for trajectory segmentation comparing to kinematic (S = 11.0, P = 0.0018, M = 86.98), linear (S = 22.0, P = 0.0188, M = 91.57), and cubic (S = 13.0, P = 0.0028, M = 91.61) interpolation. We think that this result shows that the human factor (i.e., the vessel's captain) plays an essential role in detecting fishing activities and this is reflected in random movement behaviors changes.

Table 1: Comparing OWS interpolation methods on the fishing dataset.

	RW	LIN	KIN	CUB
M	93.68	91.57	86.98	91.61
σ	1.85	2.68	4.88	1.56

The results on the hurricanes dataset are detailed in Table 2. The results show that kinematic interpolation produces the highest harmonic mean. A Mann Whitney U test indicated that the kinematic interpolation (M = 93.11) kernel produces statistically significant higher median harmonic mean for octal window segmentation comparing to random walk(S = 18.0, P = 0.0086, M = 92.45), linear (S = 10.0, P = 0.0014, M = 90.71), and cubic (S = 9.0, P = 0.0011, M = 87.91) interpolation. We think that the kinematic interpolation worked better in this dataset because a high-speed

moving objects tend to follow this strategy and also the sampling rate for this dataset is constant (e.g., every 6 hours).

Table 2: Comparing OWS interpolation methods on the hurricanes dataset.

	RW	LIN	KIN	CUB
M	92.45	90.71	93.11	87.91
σ	1.24	1.12	2.53	4.22

4.4 Comparison with a baseline

In this section, we compare the algorithms for unsupervised trajectory segmentation named OWS and GRASP-UTS. Figure 3 (a) shows a violin chart for the results of OWS segmentation on the fishing dataset in blue (left) and the GRASP-UTS in green (right) for all subsets and interpolation methods. The random walk interpolation shows visible improvements against the GRASP-UTS, as well as all the other interpolation kernels. Furthermore, a Mann Whitney U test between the GRASP-UTS and the random walk interpolation kernel on fishing dataset shows that OWS produces a statistically significant higher median for harmonic mean than the GRASP-UTS.

The Figure 3 (b) shows the results of the OWS segmentation on hurricane dataset with blue (left) and the GRASP-UTS with green (right). Even though the GRASP-UTS took advantage of using wind speed as a feature, the kinematic interpolation shows a considerable improvement against the GRASP-UTS. The other interpolation methods also show competitive results with GRASP-UTS. Another Mann Whitney U test done between the GRASP-UTS and the kinematic interpolation kernel on hurricane dataset shows that OWS produces a statistically significant higher median for harmonic mean than the GRASP-UTS.

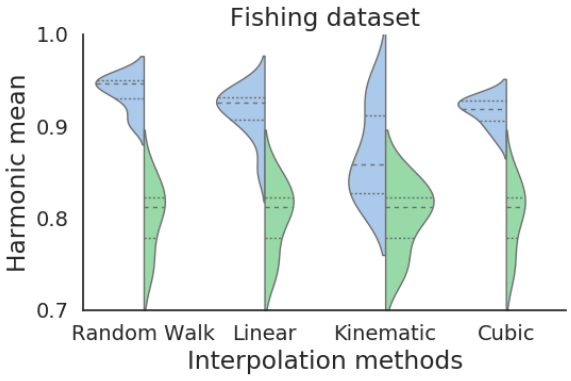


Figure 3: Comparing results of OWS against GRASP-UTS on Fishing dataset

5 CONCLUSION

In this work, we proposed an unsupervised trajectory segmentation algorithm named Octal Window Segmentation (OWS) that segments trajectory data using interpolation methods to generate a geolocation error signal from where it was supposed to be. This error signal represents possible partitioning positions where a moving object changed its behavior, and it is used to segment

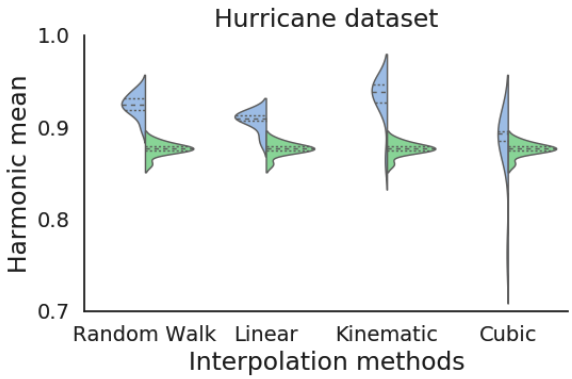


Figure 4: comparing results of OWS against GRASP-UTS on Hurricanes dataset

the trajectory data into sub-trajectories. The proposed model is flexible to different domains by adjusting the interpolation methods. The experimental results show that the kinematic interpolation is more suitable for the hurricane dataset, while the random walk interpolation was the best choice for segmenting the fishing dataset. OWS produces higher quality segmentation than the state-of-the-art segmentation algorithm, GRASP-UTS. We compare our proposed model against GRASP-UTS, and the results show that our algorithm achieved a statistically significant higher harmonic mean of purity and coverage for the hurricane and fishing datasets. Furthermore, OWS does not need any extra knowledge than the raw trajectory, while GRASP-UTS needs trajectory features such as speed, direction variation, etc.

This work can be extended in several directions. First, we intend to expand the quantitative comparison of OWS with other methods like WK-Means and CB-SMOT and use other trajectory datasets like Geolife [24]. We also intend to evaluate the possibility of other interpolation methods and the effects of increasing the window size used to create the error signal.

REFERENCES

[1] Luis Otavio Alvares, Vania Bogorny, Bart Kuijpers, Jose Antonio Fernandes de Macedo, Bart Moelans, and Alejandro Vaisman. 2007. A Model for Enriching Trajectories with Semantic Geographical Information. In *Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems (GIS '07)*. ACM, New York, NY, USA, Article 22, 8 pages. https://doi.org/10.1145/1341012.1341041

[2] Pablo Samuel Castro, Daqing Zhang, Chao Chen, Shijian Li, and Gang Pan. 2013. From taxi GPS traces to social and community dynamics: A survey. *ACM Computing Surveys (CSUR)* 46, 2 (2013), 17.

[3] Sina Dabiri and Kevin Heaslip. 2018. Inferring transportation modes from GPS trajectories using a convolutional neural network. *Transportation Research Part C: Emerging Technologies* 86 (2018), 360–371.

[4] Erico N de Souza, Kristina Boerder, Stan Matwin, and Boris Worm. 2016. Improving fishing pattern detection from satellite AIS using data mining and machine learning. *PloS one* 11, 7 (2016), e0158248.

[5] Renata Dividino, Amilcar Soares, Stan Matwin, Anthony W Isenor, Sean Webb, and Matthew Brousseau. 2018. Semantic Integration of Real-Time Heterogeneous Data Streams for Ocean-related Decision Making. In *Big Data and Artificial Intelligence for Military Decision Making*. STO. https://doi.org/10.14339/STO-MP-IST-160-S1-3-PDF

[6] Mohammad Etemad, Amílcar Soares Júnior, and Stan Matwin. 2018. Predicting Transportation Modes of GPS Trajectories using Feature Engineering and Noise Removal. In *Advances in Artificial Intelligence: 31st Canadian Conference on Artificial Intelligence, Canadian AI 2018, Toronto, ON, Canada, May 8–11, 2018, Proceedings 31*. Springer, 259–264.

[7] Shanshan Feng, Gao Cong, Bo An, and Yeow Meng Chee. 2017. POI2Vec: Geographical Latent Representation for Predicting Future Visitors.. In *AAAI*. 102–108.

[8] Amílcar Soares Júnior, Chiara Renso, and Stan Matwin. 2017. ANALYTiC: An Active Learning System for Trajectory Classification. *IEEE Computer*

Graphics and Applications 37, 5 (2017), 28–39. https://doi.org/10.1109/MCG.2017.3621221

[9] Amílcar Soares Júnior, Valéria Times, Chiara Renso, Stan Matwin, and Lucıdio A. F. Cabral. 2018. A semi-supervised approach for the semantic segmentation of trajectories. In *19th IEEE International Conference on Mobile Data Management*.

[10] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. 2007. Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 593–604.

[11] Luis A. Leiva and Enrique Vidal. 2013. Warped K-Means: An algorithm to cluster sequentially-distributed data. *Information Sciences* 237 (2013), 196 – 210. https://doi.org/10.1016/j.ins.2013.02.042 Prediction, Control and Diagnosis using Advanced Neural Computations.

[12] Jed A Long. 2016. Kinematic interpolation of movement data. *International Journal of Geographical Information Science* 30, 5 (2016), 854–868.

[13] B. N. Moreno, A. Soares Júnior, V. C. Times, P. Tedesco, and Stan Matwin. 2014. Weka-SAT: A Hierarchical Context-Based Inference Engine to Enrich Trajectories with Semantics. In *Advances in Artificial Intelligence*. Springer International Publishing, Cham, 333–338. https://doi.org/10.1007/978-3-319-06483-3_34

[14] Andrey Tietbohl Palma, Vania Bogorny, Bart Kuijpers, and Luis Otavio Alvares. 2008. A Clustering-based Approach for Discovering Interesting Places in Trajectories. In *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC '08)*. ACM, New York, NY, USA, 863–868. https://doi.org/10.1145/1363686.1363886

[15] Jose Antonio MR Rocha, Valéria C Times, Gabriel Oliveira, Luis O Alvares, and Vania Bogorny. 2010. DB-SMoT: A direction-based spatio-temporal clustering method. In *Intelligent systems (IS), 2010 5th IEEE international conference*. IEEE, 114–119.

[16] A. Soares Júnior, B. N. Moreno, V. C. Times, S. Matwin, and L. A. F. Cabral. 2015. GRASP-UTS: an algorithm for unsupervised trajectory segmentation. *International Journal of Geographical Information Science* 29, 1 (2015), 46–68.

[17] Georgios Technitis, Walied Othman, Kamran Safi, and Robert Weibel. 2015. From A to B, randomly: a point-to-point random trajectory generator for animal movement. *International Journal of Geographical Information Science* 29, 6 (2015), 912–934.

[18] Tammy M Thompson, Sebastian Rausch, Rebecca K Saari, and Noelle E Selin. 2014. A systems approach to evaluating the air quality co-benefits of US carbon policies. *Nature Climate Change* 4, 10 (2014), 917.

[19] Yann Tremblay, Scott A Shaffer, Shannon L Fowler, Carey E Kuhn, Birgitte I McDonald, Michael J Weise, Charle-André Bost, Henri Weimerskirch, Daniel E Crocker, Michael E Goebel, et al. 2006. Interpolation of animal tracking data in a fluid environment. *Journal of Experimental Biology* 209, 1 (2006), 128–140.

[20] I. Varlamis, K. Tserpes, and C. Sardianos. 2018. Detecting Search and Rescue Missions from AIS Data. In *2018 IEEE 34th International Conference on Data Engineering Workshops (ICDEW)*. 60–65. https://doi.org/10.1109/ICDEW.2018.00017

[21] Zhixian Yan, Nikos Giatrakos, Vangelis Katsikaros, Nikos Pelekis, and Yannis Theodoridis. 2011. SeTraStream: semantic-aware trajectory construction over streaming movement data. In *International Symposium on Spatial and Temporal Databases*. Springer, 367–385.

[22] Daiyong Zhang, Jia Li, Qing Wu, Xinglong Liu, Xiumin Chu, and Wei He. 2017. Enhance the AIS data availability by screening and interpolation. In *Transportation Information and Safety (ICTIS), 2017 4th International Conference on*. IEEE, 981–986.

[23] Yu Zheng, Hao Fu, X Xie, WY Ma, and Q Li. 2011. Geolife GPS trajectory dataset-User Guide. (2011).

[24] Yu Zheng, Hao Fu, Xing Xie, Wei-Ying Ma, and Quannan Li. 2011. *Geolife GPS trajectory dataset - User Guide*. https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/