Inferring Waypoints Using Shortest Paths

Daniel A. Desmond, Kenneth N. Brown

Insight Centre for Data Analytics, Department of Computer Science, University College Cork, Cork, Ireland
{daniel.desmond, ken.brown}@insight-centre.org

Abstract. We present a method for reconstructing intermediate destinations from a GPS trace of a multi-part trip, without access to aggregated statistics or datasets of previous traces. The method uses repeated forwards and backwards shortest-path searches. We evaluate the algorithm empirically on multi-part trips on real route maps. We show that the algorithm can achieve up to 97% recall, and that the algorithm degrades gracefully as the GPS traces become sparse and irregular.

1 Introduction

Due to the increase in the use of smart phones and other navigation devices which can store and send GPS or location data, mobility mining has become an important field of research. One important aspect is the ability to reconstruct some form of intent from location traces. For example, in security and surveillance, there is a need to identify significant intermediate locations as a subject moves around an environment. Similarly, in a missing persons search, valuable information may be gained by identifying which previous locations were visited by the person intentionally. In retail and marketing analysis, it is important to know which retail or service locations are destinations in their own right, as opposed to those which are visited opportunistically. In some cases, e.g. retail, inference relies on mining large sets of traces in order to determine population statistics. In other cases, e.g. missing persons, the activity is anomalous, and the aim is to identify specific behaviour by that subject from a single trace.

In this paper, we focus on the anomalous case. Given a single GPS trace, the aim is to identify the intermediate destinations within the trace, which we call waypoints. We assume we have a model of the environment as a map with distances and travel times, but no other data on popularity of locations or trajectory frequencies. We make a default assumption that a person will attempt to choose the shortest path between any two successive waypoints. We present an algorithm based on repeated forward and backward searches for shortest paths to infer the sequence of intermediate waypoints from the trace. We evaluate the algorithm empirically using randomly generated multi-trip traces on a map of the city of Rome. We demonstrate that, for a given tolerance, the algorithm correctly infers up to 97% of the waypoints, and that the algorithm is robust to irregular sampling in the trace and to blocks of missing readings.

The remainder of the paper is organised as follows: Section 2 discusses related work. Our proposed approach to the problem is introduced in section 3. Section 4 describes the form of the experiments. The results of the experiments are reported in section 5 and section 6 concludes the paper.

2 Related work

For shortest path routing the standard solution is Dijkstra's algorithm [1]. Numerous other algorithms using goal-directed techniques such as A* [2] and ALT [3] focus the search towards the target, while hierarchical methods such as Highway Hierarchies [4] and Contraction Hierarchies [5] require preprocessing of the graph prior to implementing a modified bidirectional Dijkstra to find the shortest path. Bast et al [6] give a comprehensive overview of route planning. Except for Dijkstra's algorithm, all of the methods referenced above require the destination in order to calculate the shortest path whereas Dijkstra's algorithm is a one-to-all shortest path algorithm which computes shortest paths to multiple destinations in a single pass.

Prediction using GPS traces has centred on predicting destinations and looking for patterns. The methods require the use of pattern recognition [7], hidden Markov models [8] and other machine learning methods. Another predictive use is that of identifying popular paths using clustering [9]. All of the methods above require historical GPS information to build their models. Where sub-traces or waypoints are used it again relates to predicting the destination after decomposing traces into sub-traces [10]. This method again requires a training set. Also the methods referenced above make no use of a graph of the environment. Kafsi et al [11] tackle a similar problem to ours, trying to infer a set of waypoints from a GPS trace. They assume a history of traces and estimate waypoints using a method based on the computation of the entropy of conditional Markov trajectories while not using time information to segment a trajectory. To the best of our knowledge, we are the first to present a method for inferring waypoints using only shortest path computations and not requiring historical data.

There are many trip planners available on-line such as google maps [12], mapquest [13] and some built using openstreetmap(OSM) [16] data such as osrm [14] and graphhopper [15]. Graphhopper is an open source application in which it is possible enter multi-point trips and download the trace data of a multi-point trip, and offers numerous algorithms to calculate the shortest paths.

3 Approach

Our hypotheses are

- Given a multipart trip constructed via shortest path point-to-point trips, the individual destinations (waypoints) can be identified by a series of shortest path computations.

– If the trace is irregularly sampled or has missing data, waypoints can still be reliably inferred using shortest path computations.

Let $G = \{V, E, f\}$ be a strongly connected, weighted, directed graph embedded in a two-dimensional (2D) space. V is the set of vertices where each vertex is a location in the space. E is the set of directed edges (v_i, v_j) where $v_i, v_j \in V$ and so each edge represents a line in the space. f is a function $f : E \longrightarrow \mathbb{N}^+$ representing the cost of traversing an edge. We restrict the the set of feasible points to be any vertex, or any point on an edge line. A trip s is a sequence of points and $\bar{s}$ is the last point in s. A multitrip M is a sequence of trips $\langle s_1, s_2, ..., s_j \rangle$ such that $\bar{s_i}$ is the first point in s_{i+1}. A trace $T = \langle t_1, t_2, ..., t_k \rangle$ is a sequence of points sampled in order from the trips within a multitrip. Given a trace our aim is to reconstruct the individual trips i.e. the endpoints $\langle \bar{s_1}, \bar{s_2}, ..., \bar{s_{j-1}} \rangle$ from the multitrip. We allow a relaxation in which the output is a list of intervals $\langle [a_1, b_1], [a_2, b_2], ..., [a_j, b_j] \rangle$ where $\bar{s_i}$ is contained within $[a_i, b_i]$.

Since each point is a location in 2D space, each successive pair of points has a direction between them. In order to recognise abrupt reversals of direction, we define an α-heading change as follows

Definition 1. *α-heading change:* Difference between heading of travel from t_{i-1} to t_i and heading of travel from t_i to t_{i+1} is $180° \pm \alpha°$.

Since our underlying model represents a route map, we cannot assume complete accuracy on travel times and distance, so we define an ε-shortest path as follows.

Definition 2. *ε-shortest path(time):* Path P from A to B is an ε-shortest path from A to B if there is no other A, B path with time $\leq$ time(P) - ε, where ε is measured in seconds.

Definition 3. *ε-shortest path(percentage):* Path P from A to B is an ε-shortest path from A to B if there is no other A, B path with time $\leq (\frac{100-\varepsilon}{100}) * \text{time(P)}$, where ε is a percentage.

The pseudocode for the Waypoint Estimation algorithm incorporating the definitions for α-heading change and ε-shortest path is shown in Algorithm 1. The inputs into the algorithm are the trace, allowable tolerance and heading tolerance. Initialize two lists, K to hold the estimations and ST to hold sub-traces (lines 1-2). First we search for α-heading changes, extract these as waypoints and split the trace into sub-traces using these extracted waypoints. Initally $subTraceStart$ is set to the first point on the trace (line 3). Iterate through the trace looking for abrupt heading changes which are detected at line 10. Any estimates found are added to K, a sub-trace is created and added to ST and $subTraceStart$ is set to the end of the interval (lines 11-13). Secondly for each sub-trace, we use shortest path search to find further waypoints. For each sub-trace initially *source* is set to the first point of the sub-trace (line 17) and while we have not reached the end of the sub-trace we search forward from *source* until we find a point on the sub-trace which is not a ε-shortest path (line 19). This

point is marked as Y. Search backwards from Y until we find the first point on the reverse search which is not a ε-shortest path (line 20). This point is marked as X. Add interval $[X, Y]$ to list K (line 21). Set source to Y (line 22). When we have reached the end of all the sub-traces return the list K of estimations found (line 25).

When setting *source* after finding an estimation we have many points it could be set to, these points are between the two ends of the previous estimation shown as X and Y in Fig. 2. As our assumption is that the trace is made up of shortest path point-to-point trips, if we select a point in the estimate that occurs prior to the actual waypoint then this assumption would not hold as we would have a shortest path from the source to the waypoint we have just estimated and a second shortest path from this waypoint to the next waypoint. To remove this uncertainty point Y is selected as the source for the next search as it should occur after the waypoint.

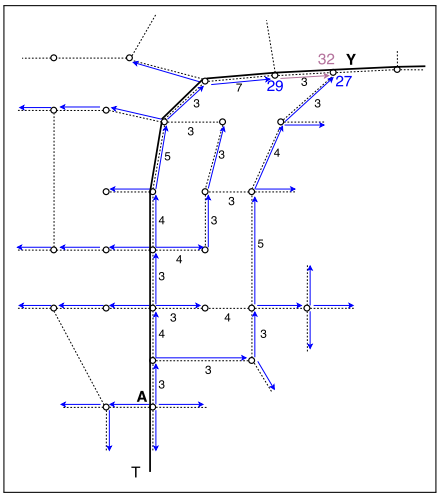


Fig. 1: Outward search from source

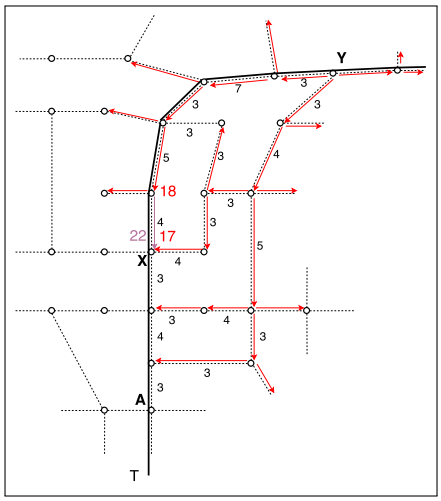


Fig. 2: Reverse search from point Y

4 Experiments

In this section we describe the creation of the graph, the test data and the values for the allowable tolerance. For these experiments we use the city of Rome as a test bed. Fig. 3 shows a map of Rome with one of the test routes. The algorithm was implemented in Java 1.8 using the eclipse IDE and run on a machine using Windows 10, an i7 CPU at 2.1 GHz and 7GB of RAM dedicated to the JVM.

The graph of the road network was created from OSM data. The only modification made was that extra nodes were added to ensure that nodes were not seperated by more than 20m.

Algorithm 1: Waypoint Estimation

input : Allowable Tolerance ε
input : Heading Tolerance α
input : Trace T
output: List K of estimates

List K
List ST `// will hold the calculated sub-traces`
subTraceStart $\longleftarrow T[0]$
for $i \leftarrow 2$ **to** *last point in* T **do**
    previous $\longleftarrow T[i-2]$
    current $\longleftarrow T[i-1]$
    next $\longleftarrow T[i]$
    *heading*1 $\longleftarrow$ heading traveled from *previous* to *current*
    *heading*2 $\longleftarrow$ heading traveled from *current* to *next*
    if *difference between heading1 and heading2 is an* α*-heading change* **then**
        add interval [*previous*, *next*] to K
        add sub-trace from *subTraceStart* to *previous* to ST
        subTraceStart $\longleftarrow$ *next*
    end
end
for $st \in ST$ **do**
    source $\longleftarrow$ *st*[0]
    while *not at end of st* **do**
        Searching from source find first point Y on trace which is not a ε-shortest path (Fig. 1)
        Searching back from Y find first point X on reverse search which is not a ε-shortest path (Fig. 2)
        add interval [X,Y] to K
        source $\longleftarrow Y$
    end
end
return K

Twelve test routes were created. The waypoints were randomly selected from the original graph data and the routes then created as shortest path point-to-point routes using graphhopper. The number of waypoints pre route varied from 17 to 22 and the duration of the routes varied between 6.61 and 9.86 hours. Graphhopper was chosen to create the routes because the latitude, longitude and timestamp of points along the trip could be exported. These routes were then sampled so that the points occured at regular intervals so as to simulate a GPS trace. A byproduct of the sampling was that except for a few cases the actual waypoint would not appear on the trace. Edge costs in graphhopper are hidden and are not necessarily the same edge costs used in our calculations.

The routes were created with the following parameters

- Mean times between readings ranging from 20 seconds to 70 seconds at 10 second intervals
- Standard deviation in the times between readings being equal to 0.0, 2.5 and 5.0 seconds (simulates time variation between readings)

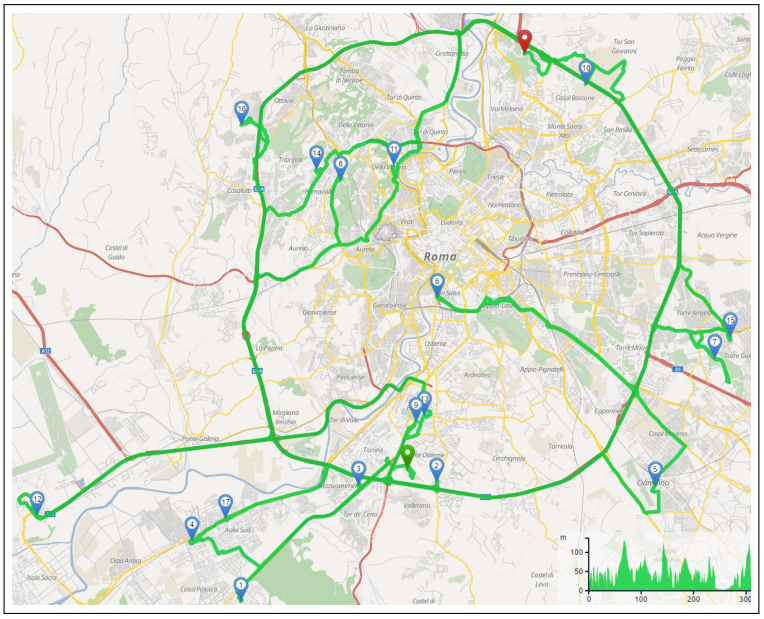


Fig. 3: Map of Rome containing one of the test routes

To simulate when a trace is missing readings due to the signal being dropped blocks of readings were removed from the traces. Table 1 shows the parameters used to remove points from the traces

Mean time between readings (secs)	precentage of readings removed	size of blocks to be removed
20, 30	7	3 - 7
40, 50	6	3 - 6
60, 70	5	3 - 5

Table 1: Parameters used to remove blocks of readings

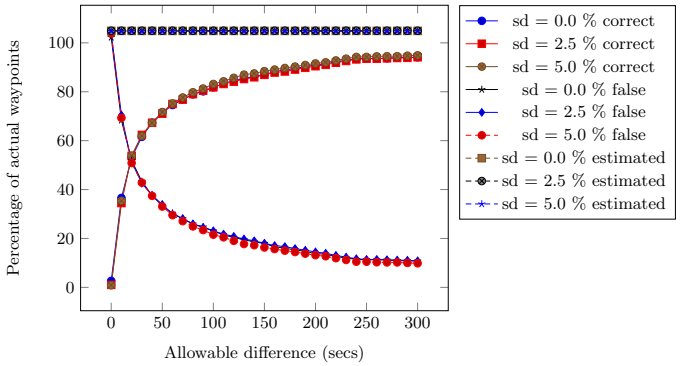
For these experiments different combinations of times and precentages were used as allowable tolerance. Four of each were chosen and combined to make up 16 different combinations. The times selected were 5, 10, 15 and 20 seconds. The percentages were 2.5, 5, 7.5 and 10 %. The heading tolerance was set to 5°.

5 Results

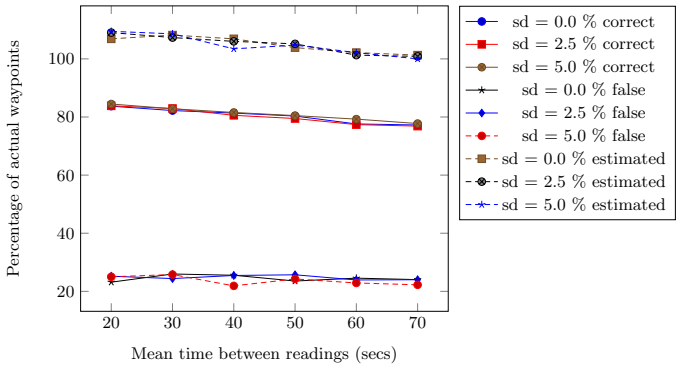
To evaluate the algorithm we use a number of measures. These are: performance as the allowable difference between the midpoint of the estimation and the actual time of the waypoint varies from 0 to 300 seconds and the performance of estimating waypoints as the mean time between readings increases. For each of these the following were measured: number of estimations returned as a percentage of actual waypoints, percentage of waypoints correctly estimated, precentage of incorrect estimations.

Due to number of combinations of tests carried out not all can documented here. Fig. 4 show the trends in the measures detailed as the mean time between readings and the allowable difference are varied when ε is set to 5% and 15 seconds with no points missing on the trace

Fig. 4a shows that as expected the percentage of correct estimations increase as the allowable difference increases and conversely the percentage of false estimations reduces. Also of note is that the percentage of estimates is greater than 100%. This means that if we correctly estimate all waypoints, there may be a number of false readings. Fig. 4b shows that as the time between readings increased the percentage of correct estimations reduced, as did the number of estimates while the percentage of false estimations remained steady. Both charts show that varying the standard deviation of the time between readings has a negligible effect on the percentages returned.



(a) Varying the allowable error



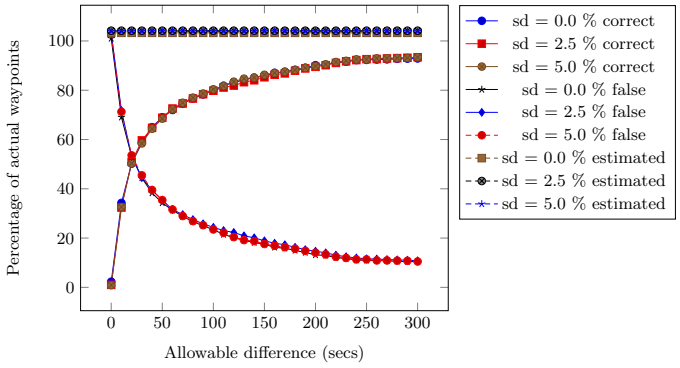
(b) Varying the time between readings

Fig. 4: Results for ε of 5 % and 15 seconds for a complete trace

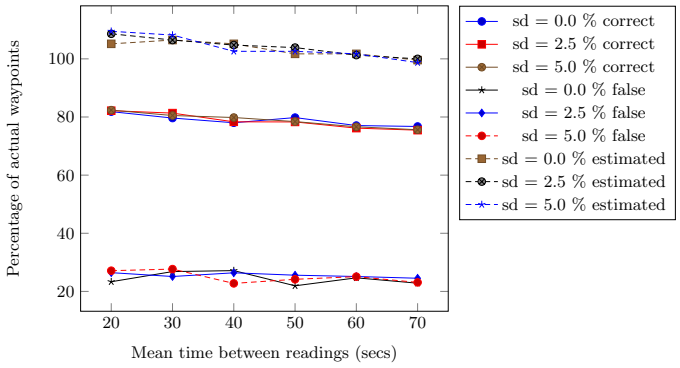
Fig. 5a and Fig. 5b show similar results for the same values of ε when the traces are missing blocks of readings.

Table 2 shows a summary where the standard deviation of mean time between readings is 0.0 and the allowable difference is 200 seconds, where the performance only slightly varies for different combinations of ε.

Confusion matrices were also constructed to evaluate the algorithm performance for estimating waypoints. The output from the algorithm is a sequence of k intervals, you can expand this by adding $k + 1$ non-overlapping additional intervals so that every point of the trace is contained in exactly one of the intervals. A true positive is an original interval that contains a waypoint. A true negative is an additional interval that does not contain a waypoint. A false negative is an additional interval that does contain a waypoint and a false positive is an interval that does not contain a waypoint. For a trace with n waypoints there



(a) Varying the allowable error



(b) Varying the time between readings

Fig. 5: Results for ε of 5 % and 15 seconds for a trace missing blocks of readings

ε		% Estimated		% Correct		% False	
Percentage	Time	None missing	Blocks missing	None missing	Blocks missing	None missing	Blocks missing
2.5	5	104.89	103.30	91.02	90.09	13.86	13.22
2.5	10	104.89	103.30	91.02	90.09	13.86	13.22
2.5	15	104.89	103.30	90.95	90.01	13.94	13.29
2.5	20	104.89	103.30	90.95	90.01	13.94	13.29
5	5	104.89	103.30	91.02	90.09	13.86	13.22
5	10	104.89	103.30	91.02	90.09	13.86	13.22
5	15	104.89	103.30	91.09	90.16	13.79	13.15
5	20	104.89	103.30	91.09	90.16	13.79	13.15
7.5	5	104.89	103.30	91.02	90.09	13.86	13.22
7.5	10	104.89	103.30	91.02	90.09	13.86	13.22
7.5	15	104.89	103.30	91.09	90.16	13.79	13.15
7.5	20	104.89	103.30	91.09	90.16	13.79	13.15
10	5	104.89	103.30	91.02	90.09	13.86	13.22
10	10	104.89	103.30	91.02	90.09	13.86	13.22
10	15	104.89	103.30	91.09	90.16	13.79	13.15
10	20	104.89	103.30	91.09	90.16	13.79	13.15

Table 2: Summary of performance of Algorithm

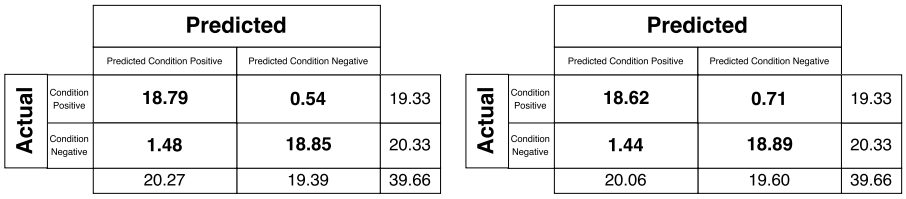
will be n positive conditions and $n+1$ negative conditions. For each trace we will have m estimations. Fig. 6 details the relationships in the confusion matrix.

For populating the matrix the true positive can be calculated by comparing the estimates to the waypoints and if a waypoint is in an estimate then it is a true positive. When all the true positives have found then the remainder of the matrix can be filled in.

Across all variations of mean time between readings the number of points in the routes ranged from 350 to 1774, and the average number of trace points per interval per test ranged from 3.5 to 18. Fig. 7a shows the confusion matrix when there are no missing readings in the trace and Fig. 7b when there are missing readings in the trace.

		Predicted		
		Predicted Condition Positive	Predicted Condition Negative	
Actual	Condition Positive	**True Positive**	**False Negative**	n
	Condition Negative	**False Positive**	**True Negative**	n + 1
		m	2n + 1 - m	2n + 1

Fig. 6: Confusion Matrix



(a) Complete trace (b) Trace missing blocks of points

Fig. 7: Confusion matrices for traces

Table 3 shows relevant measures of the efficacy of the algorithm in estimating waypoints. These show that the algorithm has high levels of accuracy, precision and recall, and that it performs equally in both the case that the trace is complete or the trace is missing blocks of data.

	No missing readings	Missing blocks of readings
Precision	0.927	0.928
Accuracy	0.949	0.946
Recall	0.972	0.963

Table 3: Efficiency of Algorithm

6 Conclusion

In this paper we have identified a method to infer where waypoints may occur in a GPS trace which does not require any prior knowledge about the person creating the trace or the timestamps on the trace only a graph of the area concerned. Our algorithm consists of two stages: looking for heading changes which infer a waypoint exists, and after splitting the trace into sub-traces between the heading changes exploiting shortest path calculations to infer where other waypoints may exist. With this method we achived a recall of up to 97%, a precision of 93% and an accuracy of 95% and the algorithm degrades gracefully as the GPS traces become sparse and irregular.

Future work will incorporate improving the algorithm to make it more robust in the real world. This will involve checking the assumption the people travel the shortest path between two points by studying available trace data. Paths are selected based on estimates with true values being determined by driving speed, congestion, traffic lights, pedestrian crossings, etc. The effects of incorporating these into the graph will be examined along with using timestamp data from GPS traces. We will extend the experiments to measure the effect of GPS traces which contain errors. Finally we will integrate these shortest path methods with existing data analytic methods in mobility mining to improve inference.

Acknowledgement. This project has been funded by Science Foundation Ireland under Grant Number SFI/12/RC/2289.

References

1. Dijkstra, E.W.: A note on two problems in connexion with Graphs. In: Numerische Mathematic, 1:269-271 (1959)
2. Hart, P.E., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. In: IEEE Transactions on Systems Science and Cybernetics, 4:100-107 (1968)
3. Goldberg, A.V., Harrelson, C.: Computing the shortest path: A* meets graph theory. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'05) (2005)
4. Sanders, P., Schultes, D.: Engineering highway hierarchies. In: ACM Journal of Experimental Algorithmics, 17(1):1-40 (2012)
5. Geisberger, R., Sanders, P., Schultes, D., Vetter, C.: Exact routing in large road networks using comtraction hierarchies. In: Transportion Science, 46(3):388-404 (2012)
6. Bast, H., Delling, D., Goldberg, A.V., Müller–Hannemann, M., Pajor, T., Sanders, P., Wagner, D., Werneck, R.F.: Route planning in transportation networks. Technical Report MSR-TR-2014-4, Microsoft Research, Redmond, WA. (2014)
7. Tanaka, K., Kishino, Y., Terada, T., Nishio, S.: A destination prediction method using driving contexts and trajectory for car navigation systems. In: Proceedings of ACM Symposium on Applied Computing (2009)
8. Alvarez-Garcia, J.A., Ortega, J.A., Gonzalez-Abril, L., Velasco, F.:Trip destination prediction based on past GPS log using a hidden markov model. In: Expert Systems with Applications: An International Journal, vol. 37 (2010)
9. Kotthoff, L., Nanni, M., Guidotti, R., O'Sullivan, B.: Find Your Way Back: Mobility Profile Mining with Constraints. In: Proceedings of Principles and Practice of Constraint Programming: 21st International Conference (CP 2015) (2015)
10. Xue, A.Y., Zhang, R., Zheng, Y., Xie, X., Huang, J., Xu, Z.:estination prediction by sub-trajectory synthesis and privacy protection against such prediction In: Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013) (2013)
11. Kafsi, M., Grossglauser, M., Thiran, P.: Traveling Salesman in Reverse: Conditional Markov Entropy for Trajectory Segmentation In: ICDM 2015: 201-210 (2015)
12. https://www.google.com/maps/
13. https://www.mapquest.com/
14. http://map.project-osrm.org/
15. https://graphhopper.com/maps/
16. https://www.openstreetmap.org/