A Weight-based Map Matching Method in Moving Objects Databases[1]

Huabei Yin Ouri Wolfson

Department of Computer Science
851 South Morgan (M/C 152)
Chicago, IL 60607, USA
{hyin, wolfson}@cs.uic.edu

Abstract

In location management, the trajectory represents the motion of a moving object in 3D space-time, i.e., a sequence of (x, y, t). Unfortunately, location technologies, cannot guarantee error-freedom. Thus, map matching (a.k.a. snapping), matching a trajectory to the roads on the map, is necessary. We introduce a weight-based map matching method, and experimentally show that, for the offline situation, on average, our algorithm can get up to 94% correctness depending on the GPS sampling interval.

1. Introduction

We view location management, i.e., the management of transient location information, as the problem of managing a set of spatio-temporal points of the form (*x*, *y*, *t*). Such a point indicates that a moving object *m* was at geographic location with coordinates (*x*, *y*) at time *t*. These spatio-temporal points may be generated, for instance, by a GPS receiver on board *m*. We will call such point a GPS point, although it may be generated by other means.

The important problem arising in location management is that GPS receivers are imprecise. Indeed, a data point of a typical GPS receiver has an error that ranges from several feet to tens of meters. In most cases, the motion of a vehicle occurs on a road network, and thus the error of a GPS point can be corrected by matching the GPS point onto the road network. Indeed, it is impossible to answer many queries precisely if the locations of moving objects are off-the-road, i.e., "retrieve the number of vehicles that are between exits 48A and 52A of I90 in the last hour". Thus, snapping, namely matching a trajectory to the streets/roads on the map, is necessary.

Currently, the prevalent map matching method available is a straightforward one which snaps each GPS point to the closest road segment. However, this method will often produce incorrect results. For example, Figure 1 illustrates a road network and several GPS points. Clearly the vehicle traveled on road segment *A*, but this is deduced only by examining the entire sequence of GPS points, and snapping GPS points *a*, *b* onto the closest road segment will produce an incorrect result.

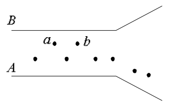

Figure 1. A trajectory snapping example

There are two kinds of snapping. One is called the offline snapping problem, which finds the overall route (i.e., a sequence of arcs in the map) of a vehicle after the trip is over. The other is called the online snapping, which during a trip, in real time, determines the road segment on which the vehicle currently is located. In this paper we concentrate only on the offline snapping problem due to the space limitation, and propose a new weight-based method for the offline situation.

The rest of the paper is organized as follows. Section 2 presents our offline snapping approach and the experiment. Section 3 discusses relevant literature. Section 4 concludes this paper.

2. Offline Snapping

Let us introduce our snapping approach first. Remember that, given a trajectory *Tr* from a source location to a destination location, the objective of trajectory snapping is to find a sequence of arcs in the map that is most resemble to the correct route of *Tr*. Our offline approach views the similarity as the distance between the trajectory *Tr* and the snapped route *R*, which is in turn to be defined as the sum of the distances between *Tr* and every arc of *R*. So, given a map *M*, and a trajectory *Tr*, we can compute the distances between *Tr* and every arc *e* in *M* in the offline environment. Consider each distance as the weight of the respective arc, the smallest weight path between the start arc and the end arc is the snapped route

[1] Researches supported by NSF Grants ITR-0086144, NSF-0209190, 0330342, 0326284.

of *Tr*. Here, we apply *Dijkstra's* shortest path algorithm to get such smallest weight path.

To compute the weight of each arc, we have the following observations: (1) the possible route of the trajectory should be close to the geometry of the arc; (2) each arc of the possible route of the trajectory is in the same or similar direction to that of the corresponding subtrajectory. These observations are two factors contributing to the weight for each arc. However, the relative contribution of each factor is unclear and impossible to determine precisely a priori. So, we propose a 3D-view (3D for short) weight algorithm to compute the weight of each arc. Our 3D weight algorithm doesn't need to fine-tune the coefficients of these factors. Intuitively, the motion on the snapped route should be close to the motion of the trajectory in 3D. Given an arc *e*, *e* can be raised from its 2D polyline to 3D as follows. First find the time-points t_i, t_j of the closest locations on *Tr* to the start vertex and end vertex of *e* respectively. Then use linear interpolation of the 2D polyline between these times, to raise the 2D polyline to 3D. Then the weight of *e* is defined as the integral of Euclidean distance between the subtrajectory from t_i to t_j and the 3D arc, divided by $|t_j - t_i|$.

Now, let us present our experiment.

In our experiments, we used a map of the Chicago metropolitan area. We actually drove in this area and collected 28 real trajectories as the experimental data. The GPS sampling interval is one second. The correct route of each trajectory is recorded manually by the traveler. For each trajectory, we considered sampling every *k* seconds, where *k* = 2, 4, 6, 8, 10, 20, 30, 40, 50, 60, 120, 300, 600, or 900. We use the *correct matching percentage* to evaluate the routes produced by the offline snapping algorithm. Consider a route *R'* that is obtained by applying the offline snapping algorithm on *Tr*. We compute the *edit distance* between *R'* and the correct route *R* of *Tr*, where *edit distance* is the smallest number of insertions, deletions, and substitutions required to change *R'* to *R*. We use the dynamic programming method [2] to compute the edit distance. The correct matching percentage of *R'* is calculated by the following equation:

$$OFF_{\text{correct}} = 100\times(1 - ed/n) \qquad (1)$$

ed is the edit distance, and *n* is the number of arcs in the correct route.

The straightforward closest-block snapping maps the trajectory vertices to the closest arc in the map. If two consecutive trajectory vertices are snapped to the same arc *e*, *e* is recorded once in the resulting route. Thus, no two consecutive arcs in the snapped route are the same.

On average, our algorithm is correct up to 94% of the time, depending on the GPS sampling interval. It is always superior to the straightforward closest-block snapping, and the superiority is up to 92%. In addition, for the offline situation, it's difficult to correctly snap the trajectory to the streets when GPS sampling intervals are larger than 120 seconds – the average correctness of both snapping methods is less than 60%.

3. Related Works

In the literature of map matching, surprisingly few works have been done considering the importance of the problem. The existing methods [1][3][4] use a GPS receiver to track the moving objects. Compared with our weight-based snapping algorithm, these techniques have following drawbacks: (1) they do not discuss the role of the time-interval between two consecutive GPS samples, whereas we do so. The resulting route for the offline situation may be disconnected when the GPS sampling interval is large. (2) It is difficult to decide the relative significance of parameters of the similarity in [3]. This relative significance is captured by the coefficients of these parameters, which are required as input. Whereas our 3D view weight algorithm successfully avoids this decision. (3) Their algorithms are aimed at the online situation. (4) One kind of the input data, the errors associated with the map, is required by [3] but it is not usually available for ordinary users; in contrast, it is easy to obtain the input data for our weight-based method.

4. Conclusion

In this paper we addressed the problem of snapping a trajectory to the road network after the trip is over. We introduced the weight-based snapping algorithm for this purpose, and compared it with the straightforward method which snaps a GPS point to the closest road segment. We showed that for the offline situation the weight-based snapping algorithm outperforms the straightforward method by tens of percentage points.

Our weight-based snapping algorithm can be easily extended to solve the online problem by returning the last arc of the offline snapped route as the online result.

References

[1] D. Bernstein, A. Kornhauser, "An introduction to map matching for personal navigation assistants". New Jersey TIDE Center, 1996. http://www.njtude.org/reports/mapmatching.pdf

[2] http://www.merriampark.com/ld.htm

[3] M.A. Quddud et al, "A General Map Matching Algorithm for Transport Telematics Applications". Centre for Transport Studies, Imperial College London, 2002: http://www.cts.cv.ic.ac.uk/documents/publications/iccts00271.pdf

[4] C.E. White, D. Bernstein, A.L. Kornhauser, "Some map matching algorithms for personal navigation assistants". Transportation Research Part C, 8, pp 91-108, 2000.