Concurrent and Robust Topological Map Matching

Roy Levin
Technion – Israel Institute of Technology
royl@cs.technion.ac.il

Elad Kravi
Technion – Israel Institute of Technology
ekravi@cs.technion.ac.il

Yaron Kanza
Technion – Israel Institute of Technology
kanza@cs.technion.ac.il

ABSTRACT

Offline map matching is a process of associating a sequence of GPS location readings measured with a device held by a traveling user, to the real-world roads that were presumably traveled by the user. The main goals in map matching are (1) to provide an association which is as accurate as possible with respect to actual traveled roads, and (2) to compute the matching as efficiently as possible. We describe our implementation of a map matching algorithm that enables parallel computation of the matching, being developed as part of the ACM SIGSPATIAL CUP 2012 contest, and we present the results of an experimental evaluation over the data of the contest. We show that our algorithm is efficient and robust in the sense that it maintains a high level of accuracy even when the sampling rate is low.

Keywords

Map matching, trajectory, GPS, parallel computation, GIS

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms, Experimentation

1. INTRODUCTION

Hand-held devices with an embedded GPS allow recording the location history of a user traveling with such a device. From the recording, a sequence of *GPS readings* is generated, where each reading comprises the location and the time of the measurement. This sequence of measurements produces a trajectory that can be used for many purposes, such as analyzing the travel habits of individuals or of groups of people, for estimating travel times and traffic conditions, supporting recommendation systems by suggesting to users routes that are frequently traveled by other users, etc.

ACM SIGSPATIAL GIS '12, November 6-9, 2012. Redondo Beach, CA, USA

In order to know on which roads users actually traveled, when given their trajectories, we need to *match* the GPS readings to real-world roads, i.e. associate each GPS reading to the road on which the user traveled at the time of the measurement. This is not an easy task as there can be errors in the matching because locations measured using a GPS are imprecise. The *accuracy* of a matching is the percentage of correct associations of GPS readings to roads out of the entire number of associations. A main task of matching algorithms is to produce a matching as accurate as possible.

Another difficulty stems from the variations in the sampling rates of the trajectories. A sampling rate is the number of recorded GPS readings per time unit. A high sampling rate increases the amount of information collected by the device. This may help achieve greater accuracy in matching a trajectory to the road network, however, it requires more storage and to process more data. Thus, sampling rates are affected by the need to save computation, reduce communication, spare storage, and by system limitations [3]. Therefore, it is desirable that a map matching algorithm will be robust in the sense that it should maintain accuracy and efficiency for different sampling rates.

Map matching can be conducted in two fashions—as an online or as an offline process. In the online version, the GPS readings are received as a stream of measures, in realtime. In the offline version, the trajectories are recorded and the entire input is provided at once. In this paper we deal with the offline version of the problem, however, the efficiency of our solution may also make it useful for online matching.

The map matching problem was studied extensively in the last years [4]. In a geometric map matching, the shape of the arcs is used to associate parts of the trajectory to the road network [1]. In the topological approach, the connections in the road network are used for creating the matchings [7]. In a probabilistic map matching the likelihood of different associations is modeled using a probability function and the matching is constructed accordingly [2].

In this paper we present a map matching algorithm that is efficient and robust. For efficiency, the algorithm is designed as a parallel process that can utilize several threads of the operating system or several computers.

2. FRAMEWORK

In this section we define the research problem.

Roads. A road network is essentially a directed graph where each vertex has a *location* specified as a pair of latitude longitude. Let $G = (V, E)$ be a graph that represents a road network where V is the set of vertexes and $E \subset V \times V$ is

the set of edges. For an edge $e = (v_1, v_2)$ we refer to v_1 as the *start* (or *left*) vertex and to v_2 as the *end* (or *right*) vertex. A path in the graph is defined in the usual way, as a sequence of vertexes $v_1, \ldots, v_n$ where $(v_i, v_{i+1}) \in E$, for every $1 \leq i \leq n-1$.

A *road* is a path $v_1, \ldots, v_n$ in G such that each vertex v_i other than v_1 and v_n, only appears in the edges (v_{i-1}, v_i) and (v_i, v_{i+1}). In other words, a road is a path that may only intersect with other roads in its start and end vertices. We denote a road as the a sequence $e_1, \ldots, e_k$ of the edges it comprises. An edge e_i *precedes* (*succeeds*) an edge e_j on road r, if they both belong to r, and e_i appears in r prior to (after) e_j.

Given two roads, $r' = \{e'_1, \ldots, e'_{k'}\}$ and $r'' = \{e''_1, \ldots, e''_{k''}\}$ we say that r' is *directly connected to* r'' if the last vertex of r' is equal to the first vertex of r''. A road r_1 is *connected to* r_m if there is a sequence of roads $r_1, \ldots, r_m$ such that r_i and r_{i+1} are connected, for every $1 \leq i \leq m-1$. We say that an edge e_1 is *connected to* edge e_2 if (1) they both belong to the same road and e_1 precedes e_2, or (2) if e_1 and e_2 belong to r_1 and r_2, respectively, such that r_1 is connected to r_2.

Trajectories. A *location reading* refers to a triple which consists of a point (i.e., latitude and longitude) and a time stamp, representing a recording of the location and the time of a GPS reading. We denote a location reading by (p, t). A *raw trajectory* is a list of location readings sorted in ascending order according to their time stamp. We refer to the *sampling gap* of the raw trajectory as the average time interval between its consecutive location readings. It is the inverse of the *sampling rate*.

Matching Trajectories to Roads. Given a trajectory and a road network, a *matching* of the trajectory to the network is a mapping μ that associates each location reading with a road of the network.

Matching trajectories to roads is based on associating location readings to edges of the network while taking into account the connectivity of the network and the distance of each point from the edge to which it is associated. Formally, given a point p, i.e. a pair of latitude, longitude, the *projection* of p on an edge e is a point p' such that p' is on the line connecting the start vertex of e to its end vertex, and the Haversine distance [5] between p and p' is minimal with respect to any other point on e. The *distance* between a point p and an edge e, denoted $d(p, e)$, is the distance between p and its projection p' on e.

The *edge match* of a location reading $l = (p, t)$ and an edge e is a tuple $m = (p, t, p', e)$ where p' is the projection of p on e. We refer to p as the *point* of m, to t as the *time stamp* of m, to p' as the *projection* of p in m and to e as the *edge* of m.

We say that two edge matches are *connected* if their edges are connected. Finally, we define the *transition cost* between two edge matches m_1 and m_2 to be the network distance from the point of m_1 to that of m_2 over the given road network. There may be different approaches for evaluating this cost. In Section 3, we describe the approach we used.

Our algorithm computes a set of edge matches, thus it is easy to produce from its answer a matching, by removing the projection p' from each tuple (p, t, p', e).

Problem definition. The accuracy of a matching is the percentage of *correct* assignments of points to edges out of all the assignments. In typical scenarios, the accuracy of the matching cannot be tested, however, when the actual location of the user is known, for each location reading, as in the data of the ACM SIGSPATIAL CUP 2012 [6], the accuracy of the matching can be evaluated.

Given a trajectory and a network, the goal is to compute, as efficiently as possible, a matching μ that is as accurate as possible. Our target is to provide a robust method that can handle raw trajectories with various sampling rates, and we utilize parallel computation to increase efficiency.

3. ALGORITHM

In this section, we describe the map matching algorithms that we have developed. First, we present Algorithm 1, namely, the *Robust Map Matching* algorithm (RMM). Then, we describe the *Concurrent Robust Map Matching* (CRMM) algorithm which enables applying RMM as parallel processes.

Both algorithms receive as input a raw trajectory $\mathcal{T} = [(p_1, t_1), \ldots, (p_n, t_n)]$ and a set of roads R. They generate from the location readings of $\mathcal{T}$ a list of corresponding edge matches $\mathcal{M} = [(p_1, t_1, p'_1, e_1), \ldots, (p_n, t_n, p'_n, e_n)]$.

We begin by providing an outline of Algorithm 1. In iteration i, the location reading (p_i, t_i) is matched with each nearby edge to create a corresponding edge match m_j^i, where $j \in \{1, \ldots, |E_{\text{nearby}}|\}$. In Algorithm 2, which is called in Line 14 of Algorithm 1, each of these edge matches is examined against the edge matches generated in iteration $i-1$. At the end of iteration i, for each $j \in \{1, \ldots, |E_{\text{nearby}}|\}$, $c\left(m_j^i\right)$ is the minimal transition cost leading from any of the edge matches generated in the first iteration to those generated in iteration i, and $l\left(m_j^i\right)$, is the corresponding list of transitions. This is achieved by concatenating, in each iteration i, the edge match m_j^i to the edge match $m_{j'}^{i-1}$ such that $c\left(m_{j'}^{i-1}\right) + \mathit{TransitionCost}\left(m_{j'}^{i-1}, m_j^i\right)$ is minimized. Thus, at the end of Algorithm 1, the returned list of edge matches minimizes the sum of its transition costs.

3.1 A Detailed Description of RMM

We now explain Algorithm 1 in detail. The algorithm consists of a main loop, beginning at Line 1, and an inner loop, beginning at Line 8. The main loop iterates over the location reading indexes. Lines 2-6 deal with finding the nearby road edges. This is done by beginning with an initial distance threshold and multiplying it by two, repeatedly, until there are enough edges near the currently examined location reading.

The set M_{next} holds the edge matches generated in iteration i and M_{prev} holds those generated in iteration $i-1$. In the inner loop, beginning at Line 8, the edge matches of iteration i are created and added to M_{next}. In the first iteration of the main loop (for $i = 1$), the cost of each m_j^i is set to 0 and the list of edge matches leading to it is set to be an empty list. For $i > 1$, $c\left(m_j^i\right)$ is set to ∞ to indicate it has not yet been calculated and the sub-procedure *ProcessTransitions*, which is depicted in Algorithm 2, is called for each edge match m_j^i.

Algorithm 2 calculates the minimal sum of transition costs from the initial edge matches to a given one (i.e, to m_{rhs}). The loop in Line 4 examines the edge matches from M_{prev} that are connected to m_{rhs}. Finding the edge matches that are connected to m_{rhs} can be efficiently done by checking only edge matches whose edges precede that of m_{rhs}. If there are no preceding edges, a hash map is used to retrieve

only the edge matches whose roads are connected to the road associated with m_{rhs}. If the number of connecting edge matches is too low, the loop iterates over every edge match in M_{prev}. (Generally, if the sampling rate is high enough, iterating over all the edges in M_{prev} will not be necessary.) The iteration is carried out in descending order according to the transition costs of the edge matches in M. The reason it is ordered this way is to allow the loop to terminate early when reaching an edge match m_{lhs} such that its cost is already greater than the cost found thus far for m_{rhs} (i.e., the minimal transition cost for m_{rhs} has already been found).

Algorithm 1 Map Matching Meta-Algorithm

MapMatcher$(R, [(p_1, t_1), \ldots, (p_n, t_n)])$

Input 1: a set of roads R
Input 2: a raw trajectory $[(p_1, t_1), \ldots, (p_n, t_n)]$
Output: a set $[(p_1, t_1, p'_1, e_1), \ldots, (p_n, t_n, p'_n, e_n)]$ of edge matches

for $i = 1$ to n **do**
  $E_{\text{nearby}} \leftarrow \emptyset$
  $d \leftarrow$ BaseNearbyDistanceThreshold
  while $|E_{\text{nearby}}| <$ NumRoadsThreshold **do**
    $E_{\text{nearby}} \leftarrow \{e \mid e \in r \text{ and } r \in R \text{ and } d(p_i, e) \leq d\}$
    $d \leftarrow 2 \cdot d$
  $M_{\text{next}} \leftarrow \emptyset$
  for $j = 1$ to $|E_{\text{nearby}}|$ **do**
    $p' \leftarrow$ *projection*$(p_i, line(e_j))$
    $m_j^i \leftarrow (p_i, t_i, p', e_j)$
    $M_{\text{next}} \leftarrow M_{\text{next}} \cup \{m_j^i\}$
    if $i > 1$ **then**
      $c(m_j^i) \leftarrow \infty$
      ProcessTransitions$(R, m_j^i, M_{\text{prev}})$
    else
      $c(m_j^i) \leftarrow 0$
      $l(m_j^i) \leftarrow []$
  $M_{\text{prev}} \leftarrow M_{\text{next}}$
return $l(m)$ s.t. $m \in M_{\text{next}}$ and $c(m)$ is minimal

Algorithm 2 Process Transitions Subprocedure

ProcessTransitions(R, m_{rhs}, M_{prev})

Input 1: a set of roads R
Input 2: an edge match m_{rhs} of the form (p_i, t_i, p', e)
Input 3: a set M_{prev} of previous matches

$M \leftarrow \{m \mid m \in M_{\text{prev}} \text{ and } m \text{ is connected to } m_{\text{rhs}} \text{ over } R\}$
if $|M| <$ MinNumOfMatchesThreshold **then**
  $M \leftarrow M_{\text{prev}}$
for each m_{lhs} in M **orderby desc** $c(m_{\text{lhs}})$ **do**
  if $c(m_{\text{lhs}}) \geq c(m_{\text{rhs}})$ **then**
    break
  $tc \leftarrow c(m_{\text{lhs}}) +$ *TransitionCost*$(m_{\text{lhs}}, m_{\text{rhs}})$
  if $tc < c(m_{\text{rhs}})$ **then**
    $c(m_{\text{rhs}}) \leftarrow tc$
    $l(m_{\text{rhs}}) \leftarrow l(m_{\text{lhs}}) \cup [m_{\text{lhs}}]$

3.2 Calculating Transition Costs

In the RMM algorithm we calculate the transition cost from an edge match $m_{j'}^{i-1}$ to an edge match m_j^i according to the minimal travel distance from the location reading of $m_{j'}^{i-1}$ to that of m_j^i over the set of roads in R. In Figure 1 we illustrate the transition cost from an edge match $m_i = (p_i, t_i, p'_i, e_1)$ to the edge match $m_{i+1} = (p_{i+1}, t_{i+1}, p'_{i+1}, e_3)$. In this example, the three edges e_1, e_2 and e_3 all reside on the same road.

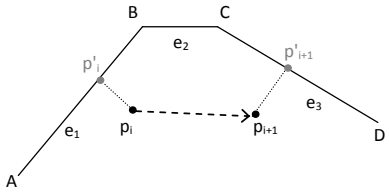

Figure 1: In the RMM algorithm the transition cost between m_i to m_{i+1} is the total distance traveled when traversing the path going via the points $[p_i, p'_i, B, C, p'_{i+1}, p_{i+1}]$.

For two edge matches $m_1 = (p_1, t_1, p'_1, e')$ and $m_2 = (p_2, t_2, p'_2, e'')$ we distinguish between two cases. In the first, e' and e'' are connected. In this case calculating the overall distance is simple. However, if they are not connected, then finding the shortest distance between the start point of e' to the end point of e'' requires some more effort. For this purpose we generate from R a directed graph G' consisting of those edges that are potentially reachable from p_1 and p_2 based on the estimated travel speed, according to the samples readings. That is, the edges of G' are those for which their distance from p_1 plus the distance from p_2 is below some predefined threshold. The threshold is set according to the time difference between the two edge matches and an estimated maximal travel speed. Now, by using a grid index, the graph can be constructed efficiently by examining only edges that reside within the grid cells that are contained in the bounding ellipse whose foci are p_1 and p_2. After constructing this graph, we apply the A* algorithm to find the minimal distance from e' to e'' as discussed above. Hence, instead of holding the entire graph of R in memory we generate, upon demand, only the required subregion of it. This significantly reduces the amount of memory required for running our algorithm, allowing it to deal with a larger number of roads (e.g., the entire road set of a state or of a country).

3.3 Adding Concurrency

We now explain how to add concurrency to the computation of the matching, to increase the efficiency of the computation. The idea behind supporting concurrency is as follows. Given a list of location readings $[(p_1, t_1), \ldots, (p_n, t_n)]$, we partition them into k lists of similar sizes. That is, we define $L_j = [(p_1, t_1), \ldots, (p_{n_j}, t_{n_j})]$, where $j \in \{1, \ldots, k\}$, such that $\sum_{j=1}^{k} n_j = n$. Each list is sent separately to be processed by Algorithm 1. Instead of returning the edge match list, the algorithm returns the set M_{next} it generated. Hence, after applying Algorithm 1 on each list of location readings we produce the intermediate edge match sets $\{M_{\text{next}}^1, \ldots, M_{\text{next}}^{n_j}\}$. We adjust Algorithm 2, as next explained, to allow it to merge consecutive edge match sets.

We next explain how to *merge* two edge match sets denoted, $M'_{\text{next}} = \{m'_1, \ldots, m'_{n'}\}$ and $M''_{\text{next}} = \{m''_1, \ldots, m''_{n''}\}$. Given an edge match $m'' \in M''_{\text{next}}$, we examine the list of edge matches $l(m'') = [mm''_1, \ldots, mm''_{nn}]$. We say that

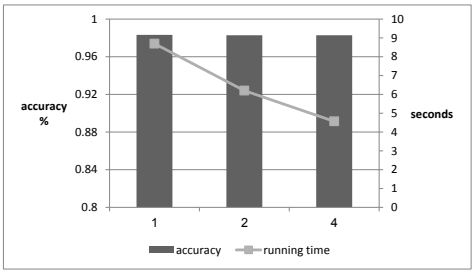

Figure 2: Accuracy and running time as a function of the number of threads.

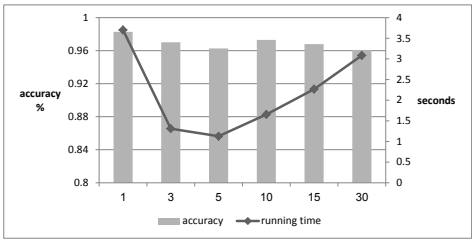

Figure 3: Accuracy and running time as a function of the number of seconds between consecutive measures (inverse of sampling rate).

mm''_1 (mm''_{nn}) is the *head* (*tail*) of the edge match m''. We apply Algorithm 2 passing to it each of the head edge matches in M''_{next}, and M'_{next} as M_{prev}. In addition, in Line 7 of Algorithm 2, we add the cost of the tail edge match corresponding to tail edge match from M''_{next}.

Next, we begin by merging M^1_{next} and M^2_{next} into $M^{\{1,2\}}_{\text{next}}$. Then, $M^{\{1,2\}}_{\text{next}}$ and M^3_{next} are merged into $M^{\{1,2,3\}}_{\text{next}}$. The merging process continues until all the intermediate edge matches are merged to form the final edge match set denoted $M^{\{1,2,\ldots,n_j\}}_{\text{next}}$. Finally, the resulting edge match list is $l(m)$ s.t. $m \in M^{\{1,2,\ldots,n_j\}}_{\text{next}}$ and $c(m)$ is minimal.

4. EXPERIMENTAL EVALUATION

We evaluated our algorithm to test its accuracy and running times. The running times we measured refer to the time it takes to run the algorithm excluding the time for reading the data into the memory and building the data structures.

4.1 Dataset

The experiments were conducted over the dataset of the contest [6]. The data contains raw trajectories of GPS readings with a sampling rate of a reading per second. By simple manipulation we filtered the GPS traces to create lower sampling rates. This was done to test the effects of various sampling rates on our algorithm.

4.2 Results

The concurrent computation described in the previous section was performed using threads of the operating system. Accordingly, our first experiment is aimed to show that an increase in the number of employed threads results in a speedup of the computation. Note, however, that an increase in the number of threads requires an initial partition into more sets L_j, as explained in Section 3.3. We show that this has only a minor effect on the accuracy of the computation.

Figure 2 illustrates the effect of an increase in the number of threads on the computation time. We can see that an increase in the number of threads improves the running time, with a minor effect on the accuracy, which cannot even be seen in the graph. According to this test we set the number of threads to be four in the following experiments.

The *robustness* of a map matching algorithm is the ability to keep its accuracy when decreasing the sampling rate. To test the robustness of the algorithm, we examined it over different sampling rates. This was done by discarding some of the readings of the original dataset to increase the gaps. For instance, a gap of 10 means that there is a GPS reading every 10 seconds and all the other readings are discarded.

The results are depicted in Figure 3. It shows that the accuracy tends to decrease as we enlarge the sampling gap. This is due to the fact that a decrease in the sampling rate increases the effect of each erroneous measure. However, we can see the robustness, where a significant decrease of the sampling rate causes only a small decrease in the accuracy.

The effect on the running time is more intricate. On one hand, a lower sampling rate means that there are less points to process and this decreases the running time of the computation. On the other hand, a lower sampling rate causes an increase in the time it takes for the algorithm to extend the matching to a successive point, because it works harder to find paths to connect each pair of consecutive points in the sequence. The combination of these two effects can be seen in Figure 3 where the running times decrease and then increase as we change the sampling rate. A sampling gap of 5 is optimal in the tested dataset, because there are no superfluous points, on one hand, and there are enough points to connect them easily, on the other hand.

5. REFERENCES

[1] R. Joshi. A new approach to map matching for in-vehicle navigation systems: the rotational variation metric. In *Proc. of Intel. Trans. Sys., IEEE*, pages 33 –38, 2001.

[2] E. Krakiwsky, C. Harris, and R. Wong. A kalman filter for integrating dead reckoning, map matching and gps positioning. In *Position Location and Navigation Symposium, 1988, IEEE*, pages 39 –46, 1988.

[3] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Map-matching for low-sampling-rate gps trajectories. In *Proc. of the 17th ACM SIGSPATIAL GIS*, pages 352–361, 2009.

[4] M. A. Quddus, W. Y. Ochieng, and R. B. Noland. Current map-matching algorithms for transport applications: State-of-the art and future research directions. *Transportation Research Part C: Emerging Technologies*, 15(5):312 – 328, 2007.

[5] R. W. Sinnott. Virtues of the haversine. *Sky and Telescope*, 68(2):159, 1984.

[6] Web site. http://depts.washington.edu/giscup/.

[7] C. E. White, D. Bernstein, and A. L. Kornhauser. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies*, 8(1):91 – 108, 2000.