From GPS Traces to a Routable Road Map

Lili Cao
Department of Computer Science
University of California
Santa Barbara, California USA
+1 805 893 3417
lilocao@cs.ucsb.edu

John Krumm
Microsoft Research
Microsoft Corporation
Redmond, Washington USA
+1 425 703 8283
jckrumm@microsoft.com

ABSTRACT

This paper presents a method for automatically converting raw GPS traces from everyday vehicles into a routable road network. The method begins by smoothing raw GPS traces using a novel aggregation technique. This technique pulls together traces that belong on the same road in response to simulated potential energy wells created around each trace. After the traces are moved in response to the potential fields, they tend to coalesce into smooth paths. To help adjust the parameters of the constituent potential fields, we present a theoretical analysis of the behavior of our algorithm on a few different road configurations. With the resulting smooth traces, we apply a custom clustering algorithm to create a graph of nodes and edges representing the road network. We show how this network can be used to plan reasonable driving routes, much like consumer-oriented mapping Web sites. We demonstrate our algorithms using real GPS data collected on public roads, and we evaluate the effectiveness of our approach by comparing the route planning results suggested by our generated graph to a commercial route planner.

Categories and Subject Descriptors

I.5.3 [Pattern Recognition]: Clustering algorithms

General Terms

Algorithms, Measurement, Experimentation, Theory.

Keywords

GPS, road map.

1. INTRODUCTION

Vehicle route planning depends on a representation of the road network. The representation must be accurate in terms of the connectivity and directionality of the road segments in order to apply basic route planning algorithms. It must also be geometrically accurate in order to display routes and give clear instructions about how far to drive between turns. This paper describes our method for creating an accurate representation of the road network starting with GPS traces from regular vehicles. Our method begins by smoothing the raw traces into a coherent set of paths. From these paths, we derive a graph of nodes and edges representing the road network. This graph is suitable for planning driving routes.

Figure 1: We used GPS data logged from shuttle vehicles driving around the Microsoft corporate headquarters campus in Redmond, WA, USA.

Traditionally, digital representations of the road network have been derived from data that comes from specialized vehicles roaming the road network, operated by skilled data gathers. This is expensive and limited by the number of specialized vehicles. A more recent approach to building a representation of the road network is exemplified by OpenStreetMap [4], which uses a combination of GPS traces and satellite images as input to a manually edited road map. While this has more scalability potential than the traditional method, it still depends on manual editing.

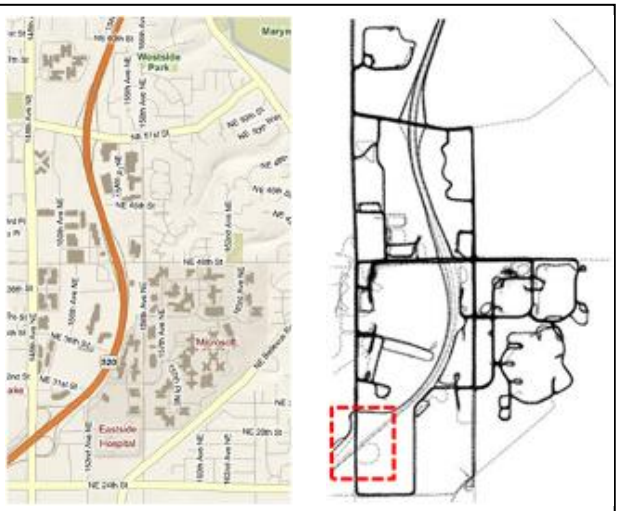
In this paper, we explore the feasibility of creating a road network representation automatically using GPS traces from non-specialized vehicles, like delivery trucks or regular cars, driven by people going about their regular business. This has the advantage of easy scalability, as it is relatively inexpensive to equip a regular vehicle with a GPS logger.

2. PROBLEM STATEMENT

In this section, we describe our collected GPS data, formalize the problem of road network generation, and outline our proposed approach.

2.1 Raw GPS Data

For our experiments, we collected trace data by deploying GPS loggers on 55 Microsoft Shuttles, as shown in Figure 1. Our loggers were RoyalTek RBT-2300 GPS loggers with a SiRF Star III chipset and WASS enabled. The shuttles roam around the Microsoft campus in Redmond, Washington, USA, continuously



(a) Test area (b) Raw GPS traces

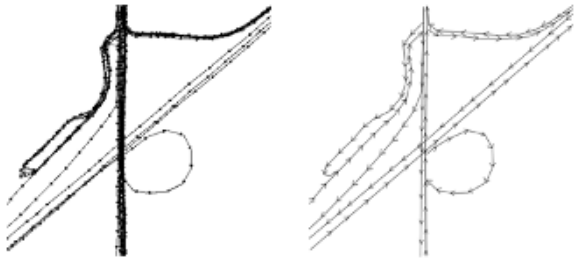
Figure 2: (a) A commercial map of our test area. (b) The raw GPS traces from our experiment.

during the day. They service both fixed and on-demand routes between buildings. The GPS loggers record time-stamped latitude/longitude coordinates with an interval of 1 second. We collected the recorded data over approximately 3 weeks. From each shuttle vehicle, we retrieved an average of about 360,000 time stamped latitude/longitude pairs. This corresponds to about 100 hours of data from each shuttle. While most of the loggers shut off automatically while the shuttles were parked, some did not, giving us some data overnight while the vehicles were idle. We can parse the data into a set of individual "trips" by looking at gaps in the timestamps when the vehicles were turned off and also extended periods of near-zero speed. Specifically, we split the data into discrete trips whenever we found a gap of at least 10 seconds or 100 meters between temporally adjacent GPS samples. We preprocessed the trip data to reduce the number of points for more efficient processing. In particular, we only retained points that were either at least 30 meters away from the previous point or at least 10 meters away if the change in direction over the last three points was greater than 10 degrees. This last condition helped preserve more points when the vehicle made a turn.

Figure 2(b) shows an overview of our collected data, with all trips plotted together. Each trip is plotted by connecting every two consecutive points with a straight line. By comparing Figure 2(b) with Figure 2(a) which is the corresponding area map clipped from Microsoft Bing Maps, we observe that the data roughly characterizes the road structure. However, even though the pictures are in approximate agreement, our aim is to compute an accurate symbolic representation of the underlying road network from the raw GPS data.

2.2 The Road Network Generation Problem

Given the raw GPS data, our goal is to infer a road network which can answer *route planning* queries. A route planning query takes a start/end pair as input and results in a route from the start to the end destination. The route should be optimized in terms of certain metrics such as length or travel time. The resulting route is often described in friendly terms, including the names of roads, how far to drive between turns, and sometimes even landmarks along the way. To support such a query, the following characteristics of the roads can be useful:



(a) Raw GPS traces (b) Ideal result

Figure 3: (a) A close-up of the GPS traces from the rectangle shown in Figure 2(b). (b) The corresponding ideal output as a directed graph.

- The connectivity and geometry of the roads;
- The average speeds and speed limits;
- The type of the roads (*e.g.* highway);
- The number of lanes for each road.

Obviously, among the above characteristics, the most fundamental and essential one is the connectivity and geometry, which can be described as a directed graph, consisting of a set of vertices and directional edges. For example, for the selected rectangular area in Figure 2(b), which is shown in detail in Figure 3(a), the ideal output as a directed graph is shown in Figure 3(b). Each vertex has an associated latitude/longitude, and connecting edges each have an associated length in meters. The fact that the graph is directed means that the resulting routes can respect one-way streets.

In this paper, we will focus on inferring this most important characteristic (*i.e.* connectivity and geometry). As we will show later, inferring solely this characteristic is already sufficiently challenging and interesting.

From Figure 3, we identify the biggest challenge of solving this problem. Since GPS traces have errors due to the inherent noise in GPS, it is nontrivial to decide whether two closely located segments of GPS trips are sampled from the same road segment or from two nearby road segments. Furthermore, for individual traces, the noise effect in the discrete points mixes with the effect of a vehicle's varying speed and direction, making the noise hard to separate and eliminate.

Besides the GPS noise, there are other challenges caused by the diversity and complexity of the road/city structure. For instance, at a parking lot, the GPS traces look like a set of chaotic points, which should not be mistaken as being from regular roads.

Because of the above challenges, generating the road network for route planning purposes still requires careful editing by humans. In the OpenStreetMap [4] community, users upload their GPS traces from casual trips to the website. There are specialized members of the community who gather all the traces for a particular area, analyze them manually, and create or edit the maps by hand. On the contrary, in this paper we try to automate the whole process, with minimum human intervention.

2.3 Related Approaches

There are several existing collaborative map editing efforts. One of the most successful is TomTom's MapShare [7] feature which allows users to edit map characteristics from the company's portable navigation devices. Users can block or unblock streets, reverse traffic directions, change turn restrictions, edit speed limits, edit street names, and add/edit points of interest. Google Maps [3] allows users to reposition the locations of street addresses. Both the TomTom and Google edits are eventually made available for other users of the respective maps.

The relatively well-known OpenStreetMap (OSM) [4] project uses GPS traces, satellite images, out-of-copyright maps, and manual editing to create road maps freely available on the Web. Thousands of volunteers have contributed data and editing, and about 40 volunteer to create and improve the system's infrastructure. In addition to the connectivity and geometry that we try to infer, OSM volunteers add and edit street names and other geographic features. Our effort is aimed at automating at least part of the map creation process without relying on manual intervention.

Another approach to our problem is to extract the road network from aerial images, such as in the pioneering work of Tavakoli and Rosenfeld in [6]. Such techniques could compliment ours, but our effort here is aimed at extracting as much as we can from GPS data.

Research efforts most closely related to ours include the work of Edelkamp and Schrödl [2], Schöredl *et al.* [5], and Worral and Nebot [9] who, like us, start with raw GPS traces and create road maps. Our work differs in that we present a simple, intuitive method to clarify the GPS traces to account for the inevitable noise in location measurement. We note, however, that the first two of the previous three references go beyond our work and infer lane structure, which we do not attempt.

2.4 Our Approach

As mentioned in the previous subsection, the biggest challenge in solving the road network generation problem is to account for the noise in the GPS traces. Inspired by this observation, we propose our approach which runs in two steps.

Step 1: Clarify the GPS traces to minimize the effect of the GPS errors. Specifically, we group nearby GPS traces (which are likely from the same road) together by simulating physical attraction between them.

Step 2: Based on the results from Step 1, we use a simple graph generation algorithm to infer the structure of the road network.

It is important to note that Step 1 is the most crucial part of our approach. Given the clarified GPS traces with minimized noise effects, it is significantly easier to infer the structure of the road network. Moreover, we believe that the clarification technique used in Step 1 has general applicability to the automatic inferences of road and traffic characteristics, such as speed limits, road types, and lane structures.

We describe in detail Step 1 and Step 2 in Section 3 and Section 4, respectively.

3. CLARIFYING THE GPS TRACES

We clarify our GPS traces in order to mitigate the effect of measurement noise in GPS. Our goal is to group together traces and points that belong to the same road. Furthermore, we want to separate traces that belong to different directions on the same road. This step is aimed at simplifying the process of creating a traditional graph representation of the road network.

3.1 Algorithm Overview

To clarify the GPS traces, we simulate physical attraction between different GPS traces, such that traces coming from the same road are grouped together. Figure 4 illustrates the main idea. We simulate two types of attraction forces, described next. In the following, our terminology is that a "trace node" is a measured latitude/longitude point from our GPS logger. For both types of forces, each trace node is temporarily unfrozen from its original position and allowed to move in response to forces generated on it. After the point is moved, the point is refrozen, another point is unfrozen, and so on for all the points.

Type 1: For each trace node A, we simulate an attraction force from each of the other trace segments (e.g. segment BC in Figure 4). Specifically, we first find the direction orthogonal to the vehicle's movement at position A, *i.e.* the direction orthogonal to DE in Figure 4. We then draw a line through A in this direction. All GPS trace segments intersecting with the line will generate an attraction force acting on A.

What are the strengths of the forces from nearby traces? Intuitively, trace segments that are close to A should generate stronger attractions than trace segments that are far from A. This is because our goal is to group together traces from the same road, and nearby traces are more likely to be from the same road. A simulated gravitational force has the correct characteristics, because its force decreases with distance. Thus, each nearby trace pulls trace node A toward it. This tends to pull together trace nodes that are slightly separated into a tight bundle of traces.

In terms of energy potential, this attractive force corresponds to a potential energy well shown in Figure 4 (middle), where the force pulls toward the direction of minimum energy. Suppose the

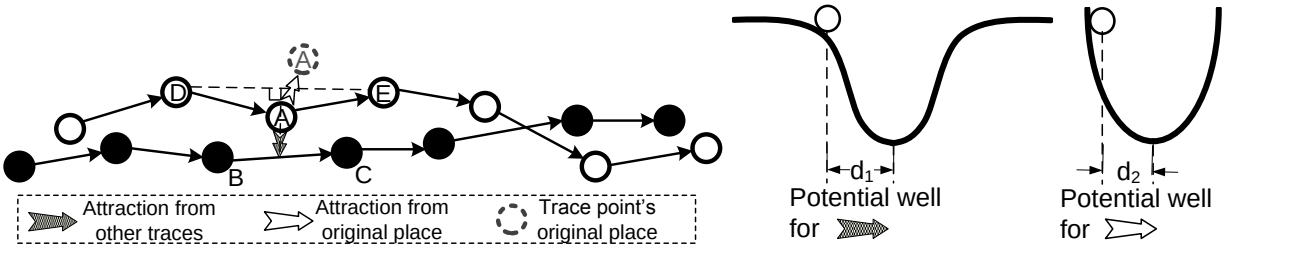


Figure 4: The attraction forces simulated in the clarification algorithm. For each trace node (e.g. A), there are two types of attractions: attractions from other traces (e.g. segment BC), and attractions from the node's original position. The potential well for the second type is a parabola.

distance of A to segment BC is d_1. Then the attraction force is the derivative of the potential well at distance d_1 from the center. This potential well is an inverted Gaussian distribution, and we give the equation governing this force in section 3.3 on parameter selection.

Type 2: While the attraction force Type 1 clearly mitigates the noise effect of GPS traces, only simulating Type 1 is not sufficient for our algorithm. With only attraction forces among traces, all traces will finally be grouped together, no matter how far apart they originally are. Thus it is natural to prevent the traces from significantly deviating from their original positions.

Inspired by this observation, we simulate an attraction force from each node's original position. This force, however, should not be a gravitation force. Intuitively, the farther the node is from its original position, the stronger the force should be. Therefore, force Type 2 is a spring force, which corresponds to a potential well shown in Figure 4 (right). Suppose the distance of A to its original position is d_2, the attraction force is the derivative of the potential well at distance d_2 from the center. As with the Type 1 force, we give the equation governing this force in section 3.3 on parameter selection.

For each trace node, we calculate the resultant of the forces that it receives. We then move each node towards the direction of the resultant with a small step. We iterate the whole system until it stabilizes, *i.e.* all nodes experience resultant forces smaller than a predefined small constant.

3.2 Differentiating lanes of opposite directions

The basic algorithm in the previous subsection groups nearby GPS traces together, regardless of their directions of travel. For the GPS traces in Figure 5(b), the result is shown in Figure 5(c). Clearly, lanes of opposite directions are not differentiated. It is beneficial, however, to differentiate lanes of opposite travel directions, and output a desired result as in Figure 5(e).

Intuitively, this idea could be realized by simply using a *repelling* force instead of an attraction force between two trace segments with opposite directions. Formally, we can multiply the (Type 1) force generated by a segment with a direction factor:

$$force \leftarrow force \cdot \cos(dir_{node}, dir_{seg})$$

where dir_{node} is the direction of the vehicle's movement at the node (e.g. direction $\overrightarrow{DE}$ in Figure 4), dir_{seg} is the direction of the trace segment (e.g. direction $\overrightarrow{BC}$ in Figure 4), and $\cos(dir_{node}, dir_{seg})$ is the cosine of the angle between the two directions. Thus when dir_{node} and dir_{seg} have opposite directions, $\cos(dir_{node}, dir_{seg}) < 0$, and the force is a repelling force; when dir_{node} and dir_{seg} have common directions, $\cos(dir_{node}, dir_{seg}) > 0$, and the force is an attraction force. When dir_{node} and dir_{seg} are orthogonal, the force equals 0, which is desired, especially at intersections where traces cross perpendicularly.

However, instead of giving us the expected output as in Figure 5(e), the revised algorithm actually outputs the result in Figure 5(d). We observe that there is a "twist effect", *i.e.* the trace nodes in one direction are repelled to different sides of the trace in the other direction. This is because when the traces from two directions are close to each other, they are randomly repelled to either side depending on the original positions.

One possible fix for the "twist effect" could be to introduce another type of force, namely a tension force, which prevents the traces from being twisted by keeping each trip smooth. This solution, however, adds more complexity to the algorithm and generates more parameters that we should potentially tune.

We adopt a simple patch to fix this twist effect. Notice that whenever two lanes of opposite directions are on the same road, it is always the case that the opposite direction lanes are on each other's left side, at least in the United States where we took our data. Therefore, it is advisable to limit the repelling force to only affect a trace's left side. Formally,

$$\underline{\text{If}}\ \cos(dir_{node}, dir_{seg}) < 0, \underline{\text{and}}\ node \text{ on the right side of } seg,$$
$$\underline{\text{then}}\ force \leftarrow 0.$$

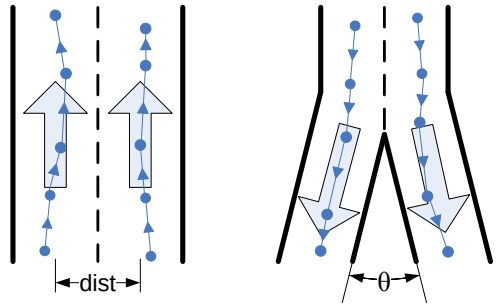
The revised algorithm produces the result in Figure 5(e), which is exactly what we want.

3.3 Parameter Choices

The two attraction forces used in our algorithm carry a number of parameters. The potential well of a gravitation force (Type 1) is decided by two parameters M and σ_1; The potential well of a spring force (Type 2) is decided by a parameter k. Obviously, we have to judiciously choose these parameters such that the algorithm groups GPS traces correctly. Instead of guessing and testing a large set of parameters, we perform theoretical analysis on typical road scenarios to gain insights on the reasonable values of the parameters.

(a) Area map (b) GPS data (c) No repelling force (d) With repelling force (e) With improved repelling force

Figure 5: (a) The area map. (b) The original GPS traces. (c) The result without repelling forces between traces of opposite directions. (d) The result with repelling forces between traces of opposite directions. (e) The improved result with repelling forces between traces of opposite directions, *only towards their left sides*.



(a) Adjacent lanes in same direction, traces should merge

(b) Road split, traces should not merge beyond split

Figure 6: (a) A scenario where two adjacent lanes are in the same direction. (b) A senario where two roads merge.

3.3.1 Adjacent Lanes in Same Direction

As shown in Figure 6(a), we consider a simple case where two adjacent lanes from the same road are in the same direction. Ideally, we should choose the parameters such that our algorithm will group the traces on the two lanes together. On the other hand, however, if the traces are from two parallel roads that are close to each other, our algorithm should group the traces into two separate sets.

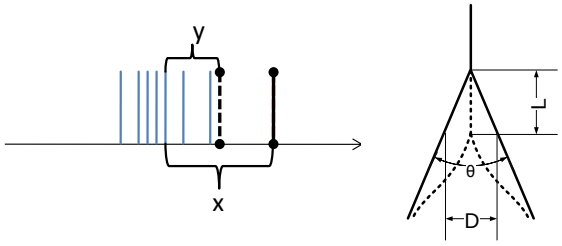
We formalize this case into a simple theoretical problem as in Figure 7(a). Assume there are set of N parallel traces distributed across the same road. Also assume, due to the GPS noise, the position of any trace admits a Gaussian distribution with mean 0 (*i.e.* the center of the road) and variance σ_2 (caused by the noise of the GPS signals). A Gaussian has been advocated as a reasonable distribution for GPS error [8]

Now let us consider what happens when a probe trace (the bold line) is placed at position x. According to the algorithm, it will be attracted by both the set of traces and its original position. Suppose it finally stops at position y (the dashed bold line).

We are interested in the relationship between x and y. The ideal relationship is shown in Figure 8(a). Specifically, when x is small, especially when it is smaller than the maximum width of a one-way road, it is most likely that the probe trace is also drawn from the same road. Thus, ideally we should have $y = 0$, i.e. the trace is drawn to the center of the road. On the other hand, when x is large, especially when it is larger than the minimum distance between two different parallel roads, it is most likely that the probe trace was taken from a drive on another road. Thus, ideally we should have $y = x$ to keep it from being grouped to the wrong road. Our goal is to choose the parameters such that the relationship between x and y is close to this ideal case.

The relationship between x and y depends on the parameters M, σ_1 and k, as follows. We concentrate on an imaginary GPS point located on the road. Each trace intersected along a line laterally across the road from this point produces Type 1 potential well:

$$p(t) = -\frac{M}{\sigma_1\sqrt{2\pi}}\exp\left(-\frac{(t-t_0)^2}{2{\sigma_1}^2}\right) \qquad (1)$$



(a) adjacent lanes

(b) road split

Figure 7: (a) A model for analyzing the scenario in Figure 6 (a). The probing trace (bold) is originally at a distance x from the center of all the traces, and is attracted and stops (dashed bold) at a distance of y from the center. (b) A model for analyzing the scenario in Figure 6 (b). Solid lines represent the original traces and dashed lines represent the traces after clarification.

In this equation, t_0 is the lateral position of the attracting trace with respect to the unfrozen GPS point that is being subjected to the forces of all the nearby traces. This potential energy well has its minimum at t_0, which means it is trying to attract the GPS point to this position. We assume that traces along this segment of road are distributed randomly according to a Gaussian distribution centered on the center of the road at a lateral position of zero. Thus the probability density function of the nearby traces is

$$f(t_0) = \frac{1}{\sigma_2\sqrt{2\pi}}\exp\left(-\frac{{t_0}^2}{2{\sigma_2}^2}\right) \qquad (2)$$

It is easy to see that the expected potential well that one trace generates is the convolution of $p(t)$ and $f(t)$. Given N traces, we can approximate the resultant potential well as

$$W_1(t) = (p * f)(t) \cdot N = -\frac{MN}{\sqrt{2\pi({\sigma_1}^2+{\sigma_2}^2)}}\exp\left(-\frac{t^2}{2({\sigma_1}^2+{\sigma_2}^2)}\right) \qquad (3)$$

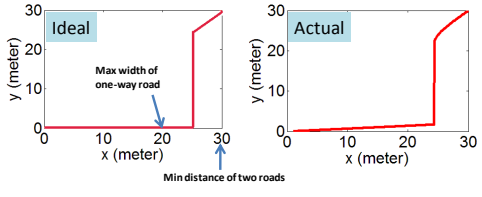
Since the attractive force is the derivative of the potential energy function, $W_1(t)$ results in an attraction force of

$$F_1(t) = \frac{d}{dt}W_1(t) \qquad (4)$$

Next, we can calculate the Type 2 attraction force from the probe trace's original position (i.e. position x) as a spring with spring constant k:

$$F_2(t) = k(x-t) \qquad (5)$$

Note that y is the position where the unfrozen point stops moving, so this lateral point is given by



(a) ideal y vs. x

(b) actual y vs. x

Figure 8: (a) The ideal y as a function of x. (b) The actual y as a function of x when we choose selected parameters.

$$F_1(y) = F_2(y) \quad (6)$$

Given (3), (4), and (5), we can write (6) as

$$-\frac{M \cdot N}{\sqrt{2\pi(\sigma_1^2 + \sigma_2^2)}} \frac{d}{dy} \exp\left(-\frac{y^2}{2(\sigma_1^2 + \sigma_2^2)}\right) = k(x - y) \quad (7)$$

Equation (7) is a transcendental equation that can be solved numerically. In practice, we should choose parameters M, σ_1 and k such that the solution of (7) is close to the ideal case in Figure 8(a).

For instance, with system parameters $\sigma_2 = 5$ and $N = 20$, we can choose $\sigma_1 = 5$, $k = 0.005$, and $M = 1$ to make the solution of Equation (7) close to ideal, as shown in Figure 8(b). These are the parameters that we used for the remainder of our processing.

3.3.2 Two roads Split

As shown in Figure 6(b), the second case that we analyze is when two roads split. For simplicity, suppose the two roads are straight with angle θ between them before they split, and traces are collected from both roads. Our clarification algorithm will make part of the two traces attract and stick together. We should choose parameters such that the number of traces that stick together prior to the split is reasonably small.

We formalize the problem as in Figure 7(b). Solid lines represent the original traces and dashed lines represent the traces after clarification. L is the distance prior to the merge that traces start sticking together, so we seek to minimize the value of L so the location of the merge is not misrepresented. The analysis below gives the relationship between θ and L.

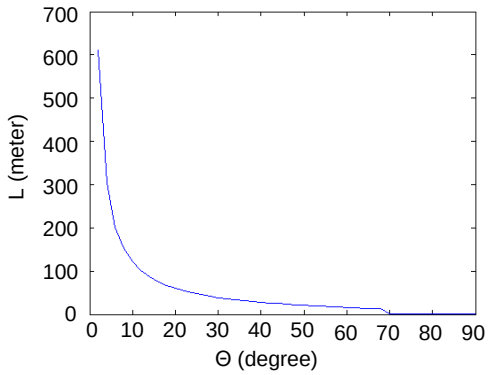


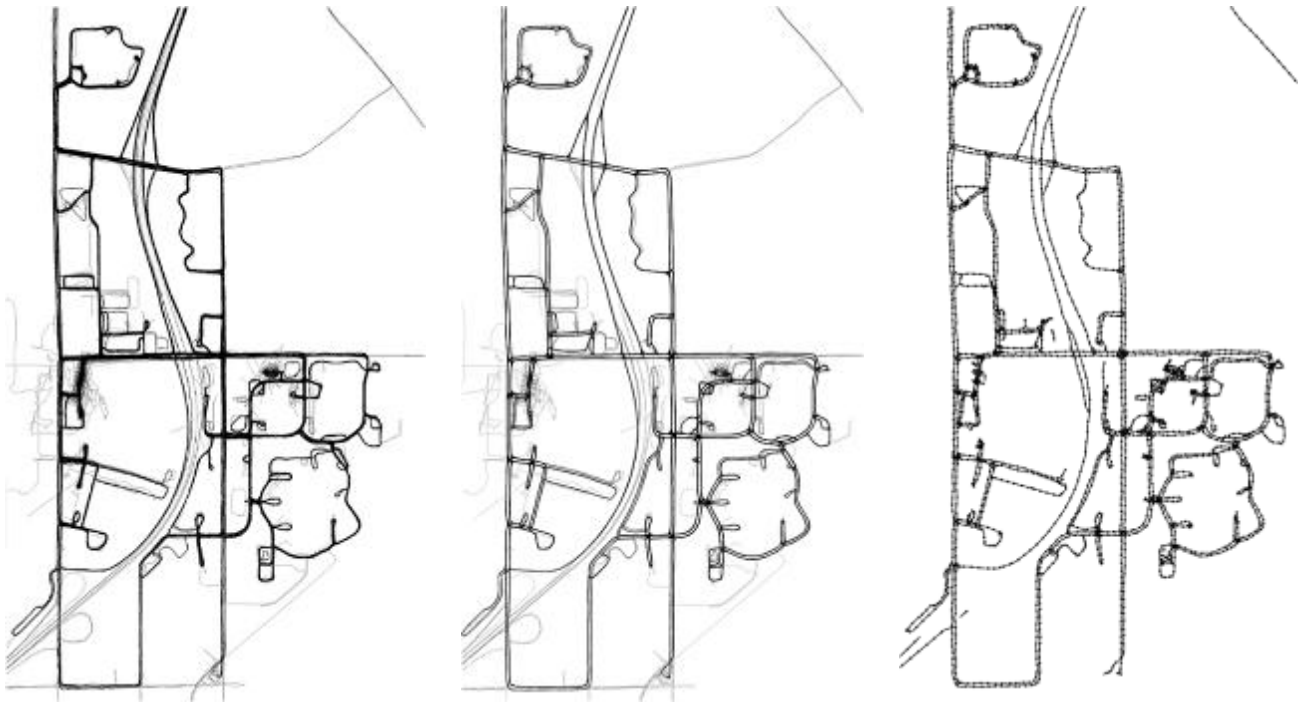
Figure 10: For a road split as in Figure 6(b), we computed this relationship between the angle of the merge and how far after the split traces are mistakenly grouped together.

Similar to the analysis in Section 3.3.1 (b), given θ, as well as the parameters σ_2, N, σ_1, k and M, we can numerically calculate the maximum D such that traces from two roads with distance D between them are grouped together. As shown in Figure 7(b), L is uniquely decided by D as

$$L = \frac{D}{2} \Big/ tan\frac{\theta}{2} \quad (8)$$

For example, with the same parameters as in Section 3.3.1: $\sigma_2 = 5$, $N = 20$, $\sigma_1 = 5$, $k = 0.005$, and $M = 1$, the relationship between θ and L is shown in Figure 10.

The advantage of the two analyses above is that we can compute



(a) Original GPS data (b) Clarified GPS data (c) Routable graph

Figure 9: An overview of the results of the clarification and the graph generation. (a) The original GPS traces. (b) The GPS traces after clarification. (c) The generated routable road network.

reasonable parameters for our clarification algorithm from first principles, avoiding the need for extensive experimentation.

3.4 Optimization for Efficiency

The clarification algorithm, when implemented naively, could be computationally expensive: For each trace node, we need to scan and calculate the potential (Type 1) attraction from all other GPS trace segments. This yields an immediate complexity of $O(M^2)$ per iteration, where M is the number of nodes in the GPS data set. On our data set, it takes 20 minutes to complete a single iteration. This number will scale poorly when the map size increases.

To improve the efficiency and scalability, we adopt the following optimization. For each node, instead of scanning *all* other GPS trace segments, we only scan the GPS segments *within a small square* (100m x 100m) centered at this node, and ignore all segments outside the square. The rationale behind this design is that the segments far from the node generate negligible amount of (Type 1) attraction, so we can safely ignore them. To efficiently find all segments within a small square, we use a kD-tree [1] to index all the GPS nodes.

After the optimization, it takes *15 seconds* on our data set to complete a single iteration, compared to the *20 minutes* without optimization.

3.5. Results

Figure 9(b) shows an overview of the result of clarification on the raw GPS data in Figure 9(a). To better understand the results, we also select four typical areas and show the results in Figure 11 (third column). Compared to the raw GPS traces, it is clear that our algorithm significantly suppresses the noise in the raw traces. While the noise is not completely eliminated in all areas (e.g. the third row in Figure 11), we believe that this level of clarity is sufficient for further processing such as the road network generation in the next section.

4. ROUTABLE GRAPH GENERATION

In this section, we present Step 2 of our approach. Namely, based on the results of the last section, we design a simple graph

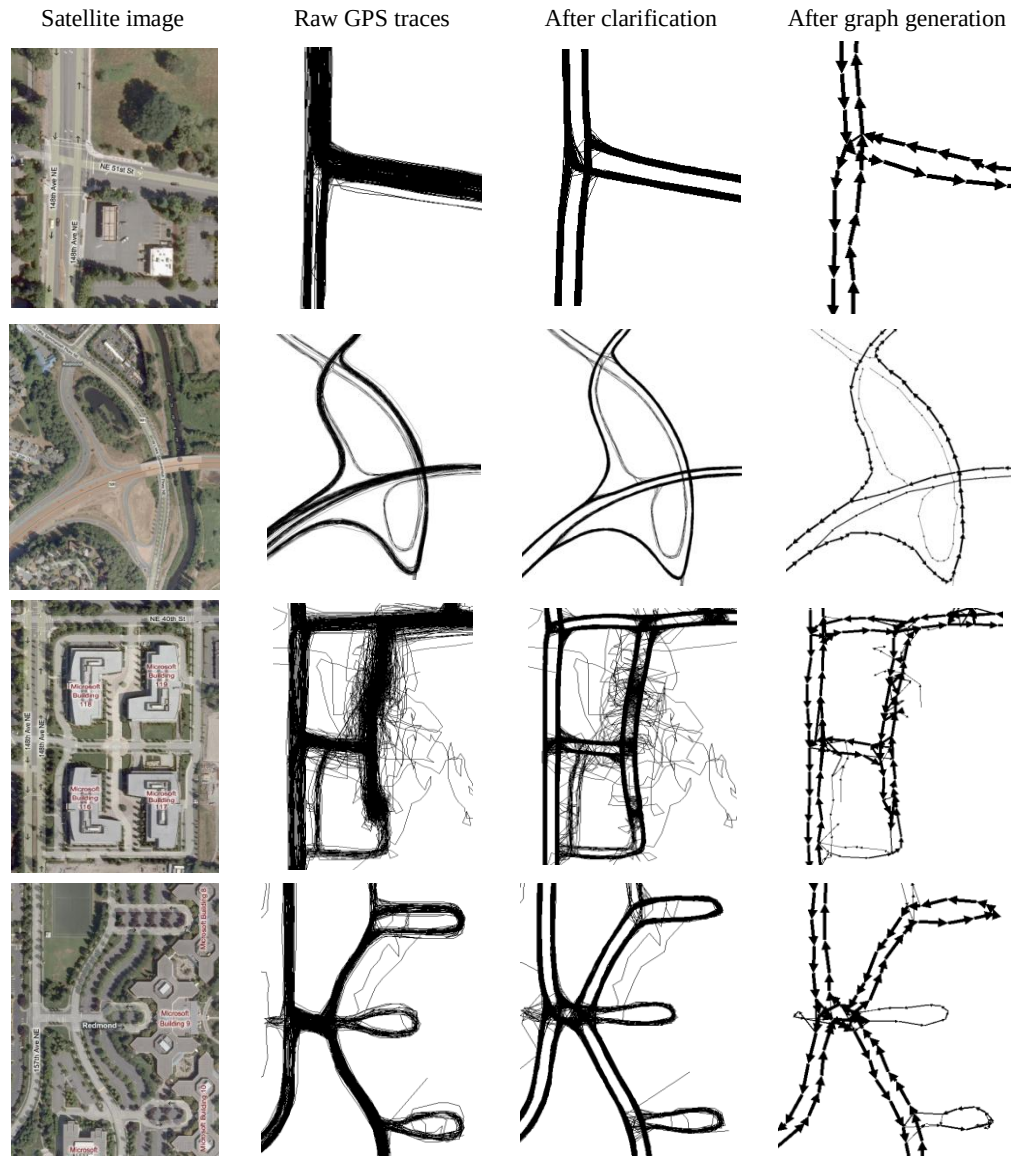


Figure 11: A detailed view of selected areas of our results. In the graph generation results (last row), every graph edge is drawn with weight proportional to its volume, i.e. the number of GPS trips that go through this edge.

generation algorithm to generate a directed graph describing the connectivity and geometry of the road network. We then show that the resulting graph is able to answer route planning queries satisfactorily.

4.1 The Graph Generation Algorithm

Overview: The graph generation algorithm works in an incremental way. Initially the graph is empty. Given the set of clarified GPS traces, we process the trips sequentially and build up the graph incrementally. For each trip, we process the trace nodes in time order. For each trace node, we search the graph (which is under construction) and decide whether it should be merged with an existing graph node (called a target node). If so, we merge the trace node with the target node; otherwise, we create a new graph node. A more formal description is shown in Algorithm 1 below.

Algorithm 1: The Graph Generation Algorithm. (Input: the clarified GPS traces, in the form of a set of trips T. Output: the graph G characterizing the connectivity and geometry of the road network.)

1: $G = \emptyset$;
2: **for** each trip *t* in T **do**
3: prevNode = null;
4: **for** each node *n* in trip *t* **do**
5: **if** *n* should be merged to graph node $v \in G$ **then**
6: **if** prevNode ≠ null and there is no path in G from prevNode to *v* within less than 5 hops, **then** add edge(prevNode → *v*) to G;
7: prevNode = *v*;
8: **else**
9: add node *n* to *G*;
10: **if** prevNode ≠ null, **then** add edge(prevNode → *n*) to G;
11: prevNode = *n*;
12: **end if**
13: **end for**
14: **end for**

When to merge: We now specify the criteria used in Line 5 of Algorithm 1 for merging a trace node *n* to a target node *v*. Intuitively, a trace node *n* is merged to a target node *v* when the algorithm infers that the two nodes come from the same road segment. Specifically, we merge them when there is an edge $e = v \rightarrow v'$ or $e = v' \rightarrow v$ in the graph and the following three conditions are all satisfied: a) the *distance* from *n* to *e* is smaller than a parameter d_m; b) the difference of the *directions* of *n* and *e* is smaller than a parameter r_m (Recall that the direction of a trace node A is defined by the vector $\overrightarrow{DE}$ if D, A, E are subsequent nodes in the trace, see Figure 4.) c) *n* is closer to *v* than to *v'*.

How to merge: By merging a trace node *n* to a target graph node *v*, we do not create new nodes in the graph. However, we have to make sure the connectivity information in the trace is recorded by the graph *G* (Line 6 of Algorithm 1). To do so, we maintain a variable prevNode, which is the graph node that the previous trace node of *n* maps to. To record the connectivity information, we require that *v* is reachable from prevNode in a small number of hops. If the current graph *G* does not reflect this information, we add a new edge from prevNode to *v*.

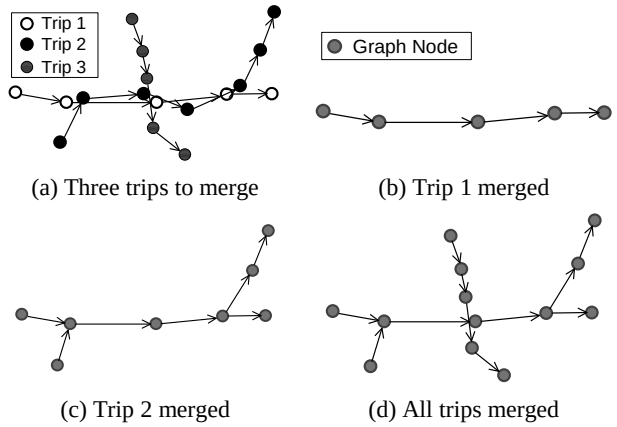


Figure 12: An illustration of Algorithm 1. (a) The input, in terms of 3 trips. (b) The graph G after processing Trip 1. (c) The graph G after processing Trip 2. (d) The graph G after processing Trip 3.

Example: Figure 12 illustrates the execution of Algorithm 1 on a set of 3 trips (Figure 12 (a)). Trip 1 is completely copied to G since there are no existing graph nodes to merge with (Figure 12 (b)). For Trip 2 (Figure 12(c)), the 2nd, 3rd, 4th and 5th nodes are merged to existing graph nodes, and the 1st, 6th, and 7th nodes are copied to G. The edges are created in a way to record the connectivity described by the trips. For Trip 3 (Figure 12(d)), none of the nodes are merged, so all the trace nodes are mapped to new graph nodes.

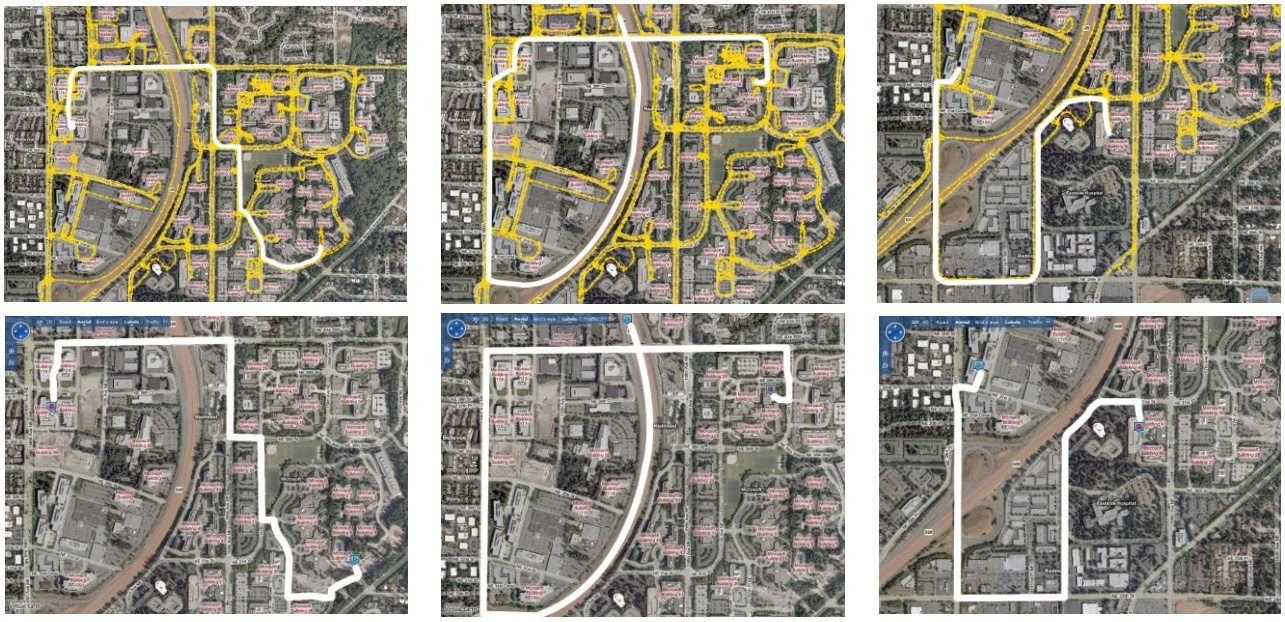
Recording traffic volume: Along with generating the graph, we also record the traffic volume of each edge, i.e. the number of trips that go through the edge. The volume is calculated as follows. Whenever a new edge is created (Line 6 and 10), the volume is set to one. Whenever we merge a node *n* to a target node *v* and there is a path in G from prevNode to *v* within less than five hops (Line 6), we increase the volumes of the edges along the path by 1.

Cleanup: From the GPS traces (Figure 9(a)), we observe that there are random outliers due to GPS signal errors. These segments usually map to graph edges with volume of one. Also, road segments that are travelled a very few times are mapped to graph edges with a low volume number. We regard these road segments as "unreliable" road segments since they may be closed or unreliable. Therefore, after we generate the graph, we remove any edge with volume less than three to obtain a more reliable graph.

4.2 Results

Figure 9(c) shows an overview of the result of graph generation. Figure 11 (fourth column) shows the results for selected areas in detail. In Figure 11, every graph edge is drawn with weight proportional to its volume. It is clear that the generated graph captures the most important connectivity and geometry properties of the road network in the experimental area.

To examine the capability of the produced graph in answering route planning queries, we select source-destination pairs and use Dijkstra's algorithm to find the shortest paths between the pairs. The results for 3 different source-destination pairs are shown in Figure 13 (upper row), where the graph is drawn in yellow and the



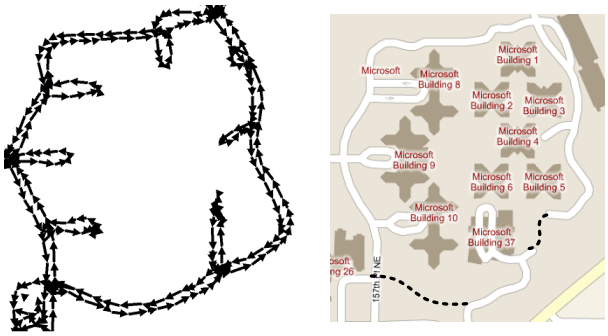
Route 1 Route 2 Route 3

Figure 13: Upper row: The route planning results of the generated map graph. The graph is drawn in yellow and the routes are drawn in red. The background is the satellite image matched to the graph. Lower row: The corresponding route planning results given by Microsoft Bing Maps [10].

computed routes are drawn in red. The backgrounds are satellite images matched to the graph.

We also compare the results with those given by Microsoft Bing Maps [10] shown in Figure 13 (lower row). We found that the routes closely match in most cases. There are, however, some cases when the routes do not match. We analyzed these cases and found they can be classified into two types:

- The generated graph provides more up-to-date information. This includes the cases when a road has recently finished construction while the Bing map was not updated to reflect this information (Figure 14). Also, we found a road that has recently been closed for road work is not part of our generated graph, but is included in the Bing map.
- The generated graph is limited by the trace data's coverage. For instance, since Microsoft Shuttles seldom travel on some freeway segments, those segments are identified as "unreliable" segments and are not included in the graph.



(a) Generated graph (b) Missing roads found

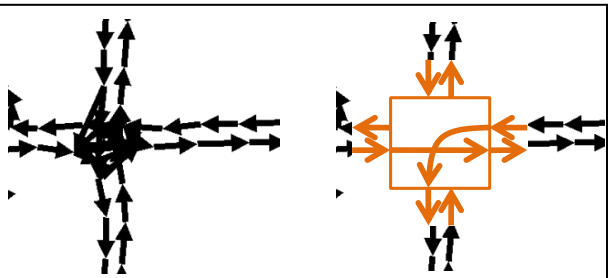
Figure 14: (a) The generated map graph. (b) The map from Microsoft Bing Maps. The dotted lines represent newly-constructed roads that were not updated at Bing Maps, but are present in the generated map graph.

4.3 Discussions

In this section, we use a simple algorithm to generate the graph from the clarified GPS traces. While the generated graph answers route planning queries satisfactorily, it could still be refined by applying more elaborate techniques. In the following, we discuss two possible improvements.

Intersection clarification: From the generated graphs we can see that the road intersections are not elegantly clarified. As shown in Figure 15(a), while the connectivity is correctly represented by the graph, there are some redundant edges that should be removed.

A possible technique to clarify the intersections is the following. First, we identify the intersection using pattern recognition, locating the square area in Figure 15(b). Next, we analyze the traces that enter the square and exit the square and identify



(a) Graph at intersection (b) Possible refinement

Figure 15: (a) An intersection in the generated map graph. (b) A possible technique to refine the graph. First identify this intersection (the square), then identify entrances and exits, and connect them by looking at the traces.

"entrances" and "exits" of this intersection. Finally, we connect "entrances" to "exits" by looking at the traces – if there is a trace that enters from an "entrance" and exits from an "exit", we connect the "entrance" and the "exit".

Parking lot recognition: Our proposed algorithm will generate a set of messy edges in parking lots. To improve the graph, particular procedures are needed to recognize and process the traces at a parking lot. We could use pattern recognition or computer vision techniques to identify these these areas. Also, note that the behaviors of vehicles at parking lots are significantly different from those at regular roads. For instance, the speeds of vehicles are usually much slower, and the traces from a parking lot are usually at the beginning or end of a single trip. Thus we may use the timestamps of the traces to help identify parking lots.

5. CONCLUSION

This paper has demonstrated a new method for creating a routable road map from GPS traces of everyday drivers. This is an alternative to the expensive process of creating a map using dedicated drivers and vehicles. To deal with the inevitable noise in GPS data, we presented an innovative approach to clarifying the GPS traces using simulations of physical forces among the traces. This process groups together traces going in the same direction on the same road. The physical simulation lends itself to theoretical analysis, which helps us pick the parameters governing the simulated forces. We presented a simple algorithm for merging the clarified traces into a representation of the road network in terms of nodes and edges. We showed how this graph representation can be used to plan reasonable driving routes.

We identified future work including the clarification of intersections and the recognition of parking lots. Other problems still left to be addressed include finding street names, address ranges, speed limits, and traffic controls like stoplights. We also plan to test the robustness of the proposed approach using other public GPS traces online.

REFERENCES

1. Berg, M.d., et al., *Computational Geometry: Algorithms and Applications*. Second ed. 2000: Springer-Verlag.

2. Edelkamp, S. and S. Schrödl, *Route Planning and Map Inference With Global Positioning Traces*, in *Computer Science in Perspective: Essays Dedicated to Thomas Ottmann*. 2003, Springer-Verlag: New York. p. 128-151.

3. Google. *Google Maps*. [cited 2009]; Available from: http://maps.google.com/.

4. Haklay, M. and P. Weber, *OpenStreetMap: User-Generated Street Maps.* IEEE Pervasive Computing, 2008. 7(4).

5. Schoredl, S., et al., *Mining GPS Traces for Map Refinement.* Data Mining and Knowledge Discovery, 2004. 9(1): p. 59-87.

6. Tavakoli, M. and A. Rosenfeld, *Building and Road Extraction From Aerial Photographs* IEEE Transactions on Systems, Man, and Cybernetics, 1982. SMC-12: p. 84-91.

7. TomTom. *TomTom Map Share Technology*. [cited 2009]; Available from: http://www.tomtom.com/page/mapshare.

8. vanDiggelen, F., *GNNS Accuracy: Lies, Damn Lies, and Statistics*, in *GPS World*. 2007. p. 26-32.

9. Worrall, S. and E. Nebot, *Automated Process for Generating Digitised Maps Through GPS Data Compression*, in *2007 Australasian Conference on Robotics & Automation*. 2007: Brisbane, Australia.

10. Microsoft. Microsoft Bing Maps. [cited 2009]; Available from: http://maps.bing.com/.