Chapter 7
Trajectory Analysis for Driving

John Krumm

Abstract This chapter discusses the analysis and use of trajectories from vehicles on roads. It begins with techniques for creating a road map from GPS logs, which is a potentially less expensive way to make up-to-date road maps than traditional methods. Next is a discussion of map matching. This is a collection of techniques to infer which road a vehicle was on given noisy measurements of its location. Map matching is a prerequisite for the next two topics: location prediction and route learning. Location prediction works to anticipate where a vehicle is going, and it can be used to warn drivers of upcoming traffic situations as well as give advertising and alerts about future points of interest. Route learning consists of techniques for automatically creating good route suggestions based on the trajectories of one or more drivers.

7.1 Introduction

GPS trajectories from vehicles are useful in a variety of ways to make driving better. It is easy to gather trajectories using small GPS loggers, and it can be even easier with GPS-equipped mobile phones and in-car personal navigation devices. Figure 7.1 shows a typical GPS trajectory recorded from a vehicle. Such trajectories typically consist of a sequence of time-stamped latitude/longitude points.

Many of the techniques discussed in this chapter can be considered crowdsourcing, which take a large collection of GPS traces from different drivers to create something useful. The chapter starts with techniques for making road maps from GPS trajectories. This is an alternative to expensive, special purpose road surveys that can have trouble keeping up with road changes. The chapter next discusses map matching. Given a road map, this is the problem of assigning GPS points to particu-

John Krumm
Microsoft Research, Microsoft Corpration, Redmond, WA USA
e-mail: jckrumm@microsoft.com

Fig. 7.1 The black dots show a GPS trajectory recorded in a vehicle, staring from the right side. The inset is a close-up of part of the trajectory, showing some of the individual points. In this case, the points were recorded every one second.

lar roads. Map matching is often an annoying, but sometimes necessary prerequisite to the chapter's next two topics: destination prediction and route learning. Destination prediction attempts to predict a driver's destination during a trip, with the goal of providing warnings and useful information in time for the driver to act. Route learning is a process that observes drivers' preferences based on GPS trajectories and uses these preferences to influences future route suggestions.

7.2 Making Road Maps from Trajectories

Digital road maps are one of the most important resources for assisting drivers, from planning a route on a desktop PC to real time route guidance in a car. Creating these maps is expensive, because it usually requires trained, dedicated personnel in specially equipped vehicles to drive the streets for the sole purpose of mapping. Technologists have explored other, less expensive methods of making digital road maps. One of the earliest attempts was to use aerial imagery, from airplanes and satellites, along with computer vision algorithms, to automatically find roads, such as the early work by Tavakoli and Rosenfeld [4].

Another inexpensive approach is manual crowdsourcing, as exemplified by the successful OpenStreetMap (OSM) project founded by Steve Coast [13]. OSM's registered users can update the map by editing and adding entities, including geographic features derived from aerial images, out-of-copyright maps, and uploaded GPS traces.

In this section, however, we explore the automated processing of GPS trajectories to create a map. Figure 7.2 shows a set of GPS trajectories that our research group has collected over the past few years in Seattle, WA USA. Given that the raw traces

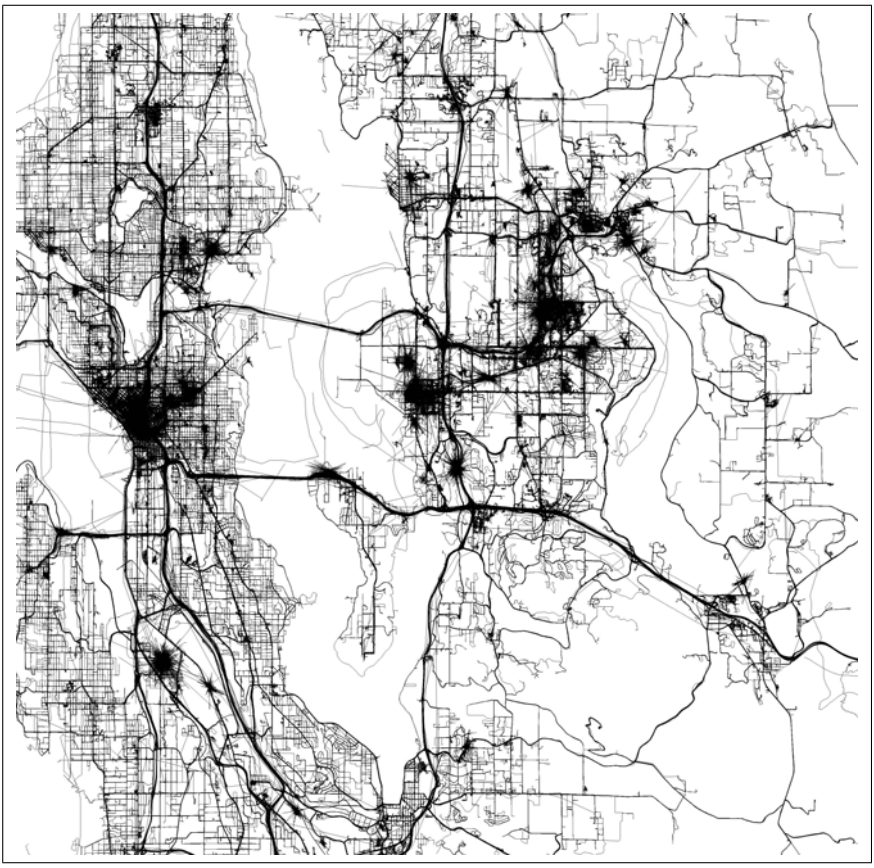


Fig. 7.2 These are GPS traces taken from volunteer drivers in Seattle, WA USA. Based on GPS traces like this, it seems plausible that one could build a road map. The dark, star-like clusters come from idling GPS loggers that occasionally recorded an outlier location.

already look like a road map, it seems plausible to automatically make a road map from such data.

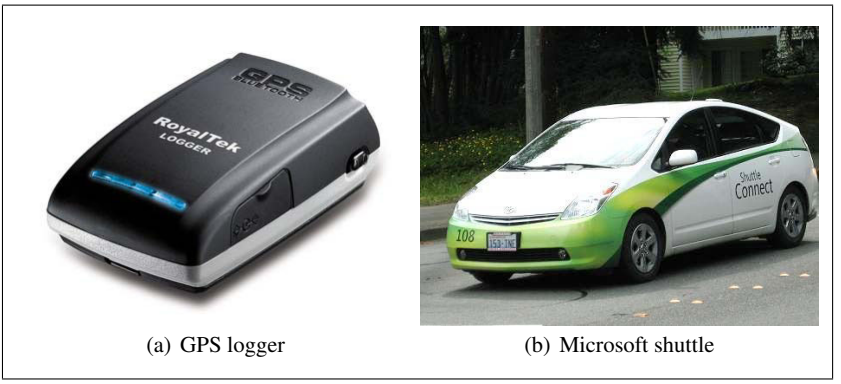
One of the first attempts to make a road map from GPS traces came from Rogers et al. in 1999 [24]. In a series of papers, this idea was developed into an end-to-end system that starts with differential GPS data and concludes with a refinement of an existing map, including finding lanes and lane transitions through the intersections [9]-[25]. The process involves smoothing and filtering the GPS data, matching to an existing map, spline fitting for the road centerline, clustering to find lanes, and refinement of the intersection geometry.

Worrall and Nebot approached the problem of finding roads in a large mining site without using any previous maps [14]. Their algorithm starts by clustering GPS points with similar locations and headings. These clusters are in turn linked, and then the linked clusters are segmented into straight line segments and curves. The

algorithm uses least squares to fit lines and circular arcs to these segments, giving a compact representation of the mine's roads.

Of course road maps need to represent much more than the centerlines of the roads. Roads have several other important features, some of which could be derived from GPS data. These features include direction of travel, number of lanes, traffic controls like stop lights and stop signs, speed limits, road names, turn restrictions, and height and weight restrictions. It is an interesting challenge to imagine techniques for deriving these features automatically.

The next three sections describe three projects in our own research lab aimed at inferring a road map and road details from raw GPS data. The goal of the first project was to make a routable road map, the second to find intersections, and the third to count lanes.



(a) GPS logger (b) Microsoft shuttle

Fig. 7.3 We installed GPS loggers (a) in Microsoft shuttles (b) to record GPS data for automatically making road maps.

7.2.1 Routable Road Map

Our lab's first project in this area was an attempt to create a routable road map from recorded GPS data [10]. For this experiment, we installed GPS loggers on Microsoft shuttle vehicles operating on and between Microsoft corporate campuses in the Seattle, WA area, shown in Figure 7.3. The GPS loggers we used were Royal-Tek RBT-2300 models. These loggers are convenient because they can hold 400,000 GPS points (including latitude, longitude, altitude, speed, and time stamp), and they can be powered from the vehicle's cigarette lighter. For this experiment, we deployed loggers on 55 different Microsoft shuttles, recording GPS points every 1 second for approximately three weeks.

We split the GPS traces from each shuttle into discrete trips and dropped some of the GPS points for easier processing. Specifically, we split the GPS points into

trips by looking for gaps of at least 10 seconds or 100 meters between temporally adjacent GPS points. To reduce the amount of data, we dropped points that were within 30 meters of a previous point, unless the point was part of a turn (change in direction over last three points greater than 10 degrees), when we allowed the points to be as close as 10 meters apart.

Our process for making a map proceeded in two steps. First we "clarified" the GPS traces in an effort to mitigate the effect of measurement noise. Second, we clustered the clarified traces into discrete roads represented by a graph with nodes and edges.

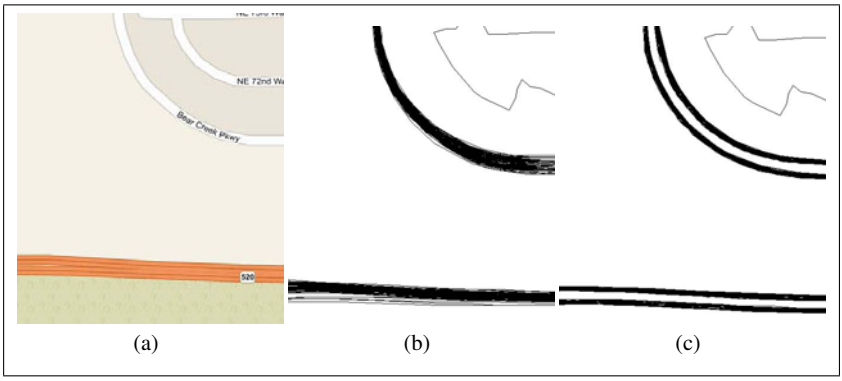


Fig. 7.4 Part (a) shows a road map that corresponds to the raw GPS traces in (b). After clarifying, we get the traces in (c) that are more compact and that show the roads' two directions of travel.

7.2.1.1 Clarifying GPS Traces

GPS measurements inevitably have noise, which can be reasonably modeled with a Gaussian distribution [13]. As an example of GPS noise, Figure 7.4 shows some of the GPS data we collected along with a section of a road map for the same area. The raw GPS traces are spread, making it more difficult to infer the location of the road and to differentiate the two directions of travel.

We developed a technique to clarify the GPS traces by imagining each trace as a special kind of electrostatically charged wire. Pairs of wires with the same direction of travel were attracted to each other over short distances, while pairs of wires with the opposite direction of travel repelled each other over short distances. The traces could move in response to these forces, but they were also anchored with imaginary springs to their original locations to prevent too much deviation.

These imaginary forces are simulated as energy potential wells, as shown in Figure 7.5. Figure 7.5(a) shows a cross sectional view of two traces. The inverted Gaussian potential well is centered on one trace, and it tends to pull the other trace toward it at the bottom of the potential well. The force is proportional to the derivative of

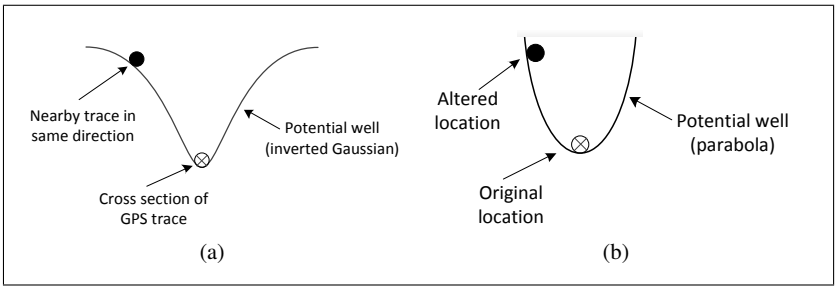


Fig. 7.5 These illustrations show cross sections of GPS traces as small circles. In (a), a nearby GPS trace is attracted to the potential well around another trace. In (b), a trace is attracted back to its original position.

the potential well. These attractive forces are counterbalanced by forces that tend to keep the traces from moving too much, as in Figure 7.5(b). This parabolic potential well simulates a spring that pulls a trace back to its original position.

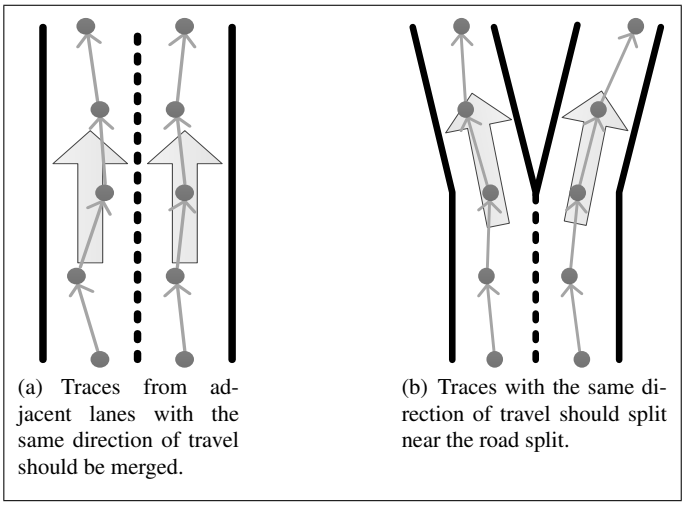
We modify the attractive force of the potential well in Figure 7.5(a) to help separate opposite directions of travel. The computed force is multiplied by the cosine of the angle between the two traces. Thus, traces with nearly the same direction will be subject to nearly the full attractive force ($\cos 0^\circ = 1$), while traces with nearly the opposite direction will feel a repelling force ($\cos 180^\circ = -1$). We had to adjust this cosine-modulated force slightly to prevent strange effects from traces that drifted left past the opposite direction of travel. Details are in [10].

The two types of potential wells in Figure 7.5 are each governed by a small set of parameters giving their amplitude and width. One way to set these parameters is to experiment with actual traces to see which parameters work best. Instead, we wrote equations to simulate the behavior of the potential wells and traces and solved for the parameters numerically. We specifically looked at two scenarios that were relatively easy to capture with equations: adjacent lanes in the same direction and a road split, shown in Figure 7.6. For the adjacent lanes, we adjusted the potential well parameters so the traces would merge, and for the road split, we tried to maintain the location of the split in spite of tendency for traces in the two lanes to merge.

The result of the clarification step is shown in some examples in Figure 7.7. The procedure successfully consolidated traces going in the same direction and separated traces going in the opposite direction, helping to suppress the GPS noise.

7.2.1.2 Merging Traces

To build a routable road map, we need to convert the GPS traces into a road network represented by a graph of nodes and edges. After clarifying the GPS traces, the next step is to merge them into a graph. Figure 7.8 shows an example of merging. We begin with an empty graph and choose one clarified GPS trip as the first set of nodes and edges. Each GPS point is a node, and the connections between temporally



(a) Traces from adjacent lanes with the same direction of travel should be merged.

(b) Traces with the same direction of travel should split near the road split.

Fig. 7.6 We adjusted the parameters of our potential functions to achieve the desired effects in these two situations.

adjacent nodes are edges. To add more clarified traces, we merge new nodes with existing nodes. For a given candidate node to add, we look for the nearest nodes with the same direction of travel. If the nearest node is close enough, we merge the two nodes. If necessary, we also add an edge to preserve the connectivity of the candidate node with previous nodes in its trip. More details of this merging algorithm are in [10].

7.2.1.3 Routable Road Network

With the road network created, we can run a route-planning algorithm to compute actual driving routes. Some of these routes are shown in Figure 7.9, along with comparisons to routes computed with ™ Maps. Our routes match well. Where there are deviations, one is due to the fact that a road had closed, which our graph represented correctly over the slightly out-of-date Bing™ version. Another deviation was due to the fact that we never observed a GPS trace on a road that was actually there. Overall, however, this technique of generating road maps shows that it is possible to create a routable road map from raw GPS traces without the expense of paid drivers in specialized vehicles.

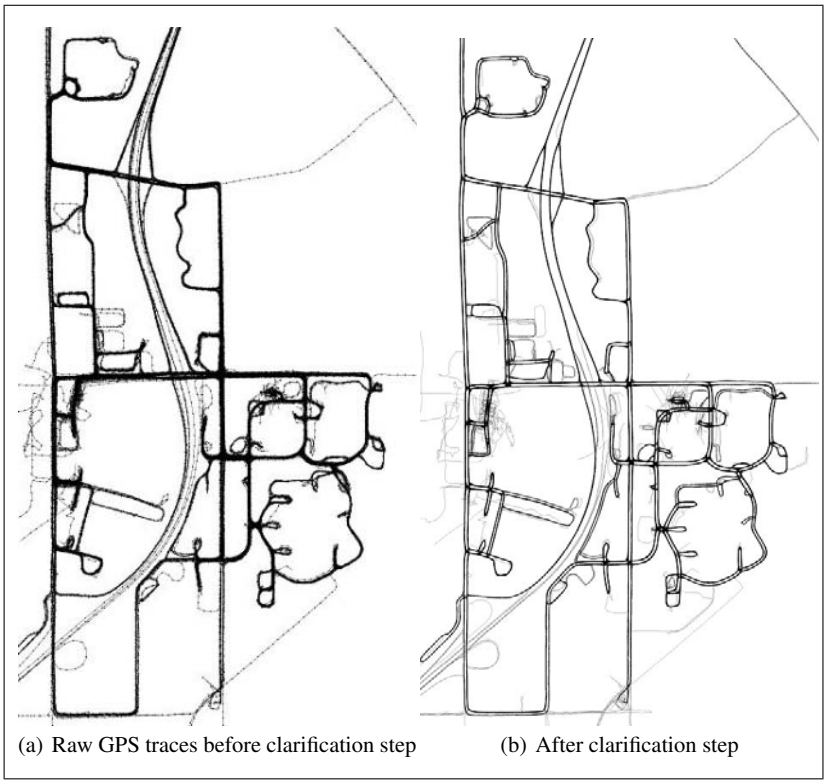


Fig. 7.7 The GPS traces in (a) were taken from shuttles around Microsoft in Redmond, WA USA. The clarified version of the traces is shown in (b). Here, the directions of travel have been separated, and traces in the same direction are pulled together.

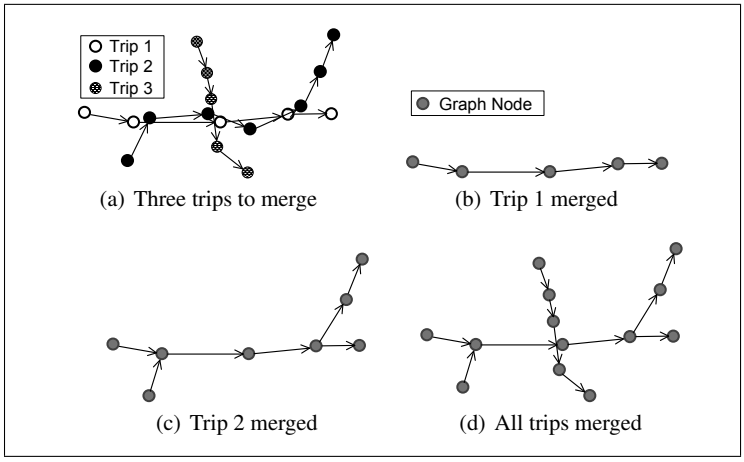


Fig. 7.8 This shows an illustration of our GPS trace merging algorithm to product a graph representation of the road network.

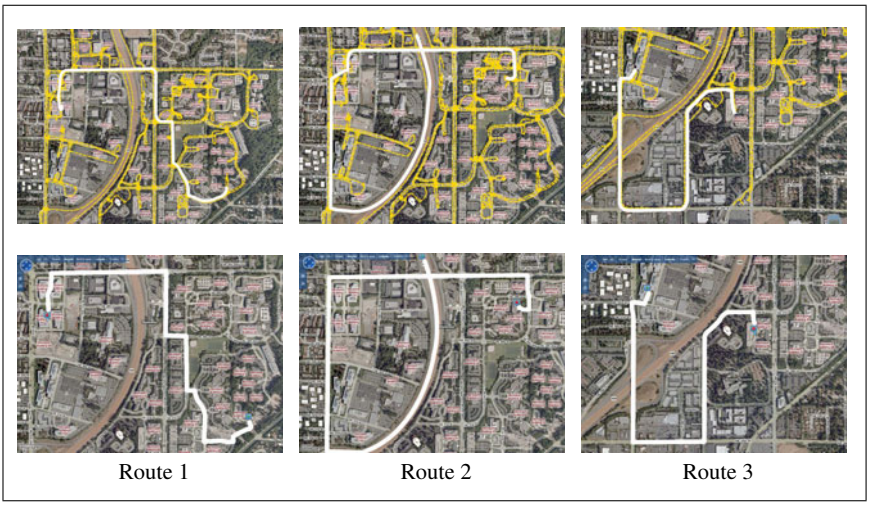


Fig. 7.9 The upper rows shows the road network we generated and routes planned on this network in white. The bottom row shows the same routes planned in Bing™Maps.

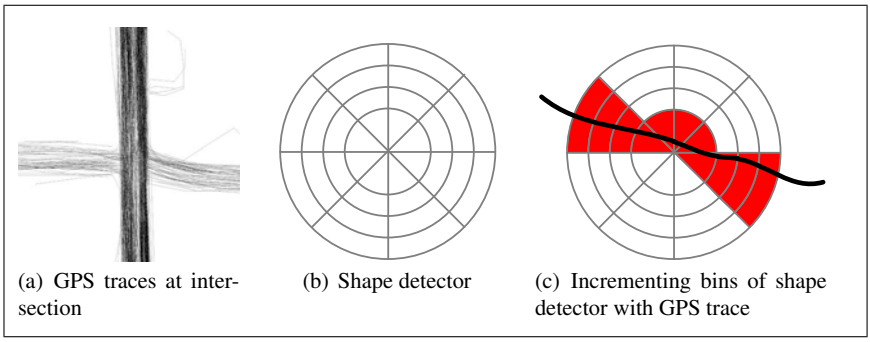
7.2.2 Intersection Detection

While the work above on making a routable road network finds road intersections implicitly, we wanted an explicit way of finding intersections, because this is where roads are often most interesting.

7.2.2.1 Detecting Intersections

Figure 7.10(a) shows GPS traces on an intersection. It might be possible to create an algorithm that closely examines the geometry of GPS traces like this to infer that they represent an intersection, but it would be difficult to scale an algorithm like this to all types of intersections with varying densities of GPS traces. Instead, we created a shape detector that can be automatically trained to find intersections of many different types, as shown in Figure 7.10(b).

The shape detector works by virtually placing it at some latitude/longitude point and examining the underlying GPS traces. The detector is similar to a two-dimensional histogram in that each bin counts the number of traces passing through it, as shown in Figure 7.10(c). After completing the counts, we normalize them so they sum to one in an effort to make the detection less dependent on the absolute number of GPS traces at the intersection. We also rotate the shape detector to a canonical orientation. Specifically, we rotate so the bin with the maximum value is at an angle of zero. This makes the detector less sensitive to the absolute orientation of the intersection.



(a) GPS traces at intersection (b) Shape detector (c) Incrementing bins of shape detector with GPS trace

Fig. 7.10 Part (a) shows typical GPS traces at a road intersection. The shape detector in (b) is used to find intersections. A GPS trace increments the bins it intersects in the shape detector, as in (c)

With the counts normalized and with the detector rotated to a canonical orientation, we extract the counts into a feature vector with each vector element corresponding to one bin of the shape detector.

Our goal is to apply a standard machine learning technique to the feature vectors. In order to do this, we need training data consisting of positive and negative samples of intersections. Since we have a ground truth map with the actual locations of intersections in the region of our GPS data, we use it to find positive feature vectors for training, and we take negative examples centered at GPS points that are at least 20 meters from any known intersection. Given these positive and negative training samples, we use Adaboost [26] to learn a classifier, although several other classifiers likely would have worked as well.

The algorithm outlined above reduces the problem of finding road intersections to a classic detection problem. We can thus assess the performance of the algorithm with a receiver operating characteristic (ROC) curve, as shown in Figure 7.11. This shows the tradeoff between correctly finding intersections (true positives) and hallucinating intersections that are not actually there (false positives) as we adjust the sensitivity of the Adaboost classifier. The ideal operating point is in the upper left corner, where all the intersections are found and there are no false positives. We optimized the geometry of our shape detector (e.g. number of circles, number of angular slices) by finding the geometry that gave the best ROC curve.

7.2.2.2 Refining Intersections

Although our main goal is to find intersections, we find roads by looking for GPS traces that directly connect the detected intersections. Some of the intersections and roads we found are shown in Figure 7.12. Some of the intersections we did not find are due to the fact that we did not have enough GPS traces passing through them.

With these roads, we can refine the locations of the intersections using the Iterated Closest Point (ICP) algorithm [7]. In general, ICP is a technique to find the

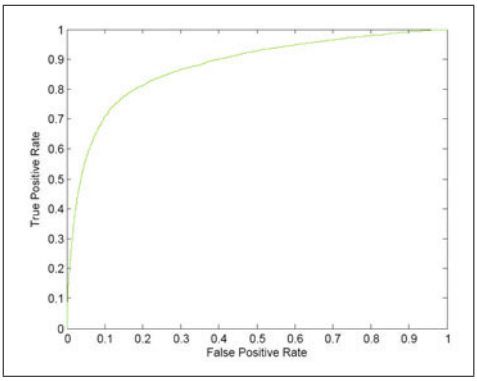


Fig. 7.11 This is the receiver operating characteristic (ROC) curve of our intersection detector. More correct detections (true positives) come at the expense of more false positives.

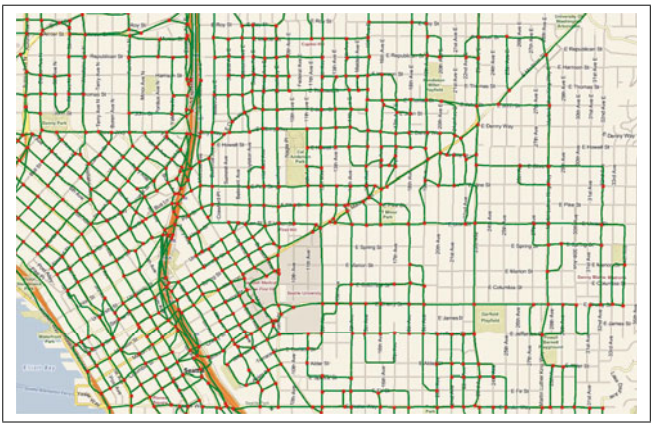


Fig. 7.12 These are some of the intersections found by our intersection detector and the roads filled between the detected intersections.

geometric transformation between measured points on a model curve or surface and the model curve or surface itself. In our case, the measured points are the GPS points near the intersection, and the model is the roads making up the intersection. We use ICP to find the geometric transformation between the intersection's roads and the GPS points, which tends to center the roads on the intersection, giving a more accurate measurement of the intersection's location. Before ICP, the mean distance between our detected intersections and their ground truth locations on a map was 7.2 meters. After ICP, the mean error was reduced to 4.6 meters.

More details about our intersection detection work can be found in [14].

7.2.3 Finding Traffic Lanes

Having discussed how to use GPS traces to create a routable road map and pinpoint intersections, this section concludes with a discussion of how to find traffic lanes. A detailed discussion of this method appears in [11].

An example of this problem is shown in Figure 7.13. On the left is a set of GPS traces going through an intersection, and on the right is an aerial view of the same intersection. Our goal is to find the lanes of traffic from the GPS traces. GPS traces from adjacent lanes have significant overlap, which makes this a difficult problem. This overlap is partially due to GPS noise. In addition, some of the GPS spread comes from the fact that the GPS loggers were not always centered in the vehicle, but placed at various positions across the dashboard.

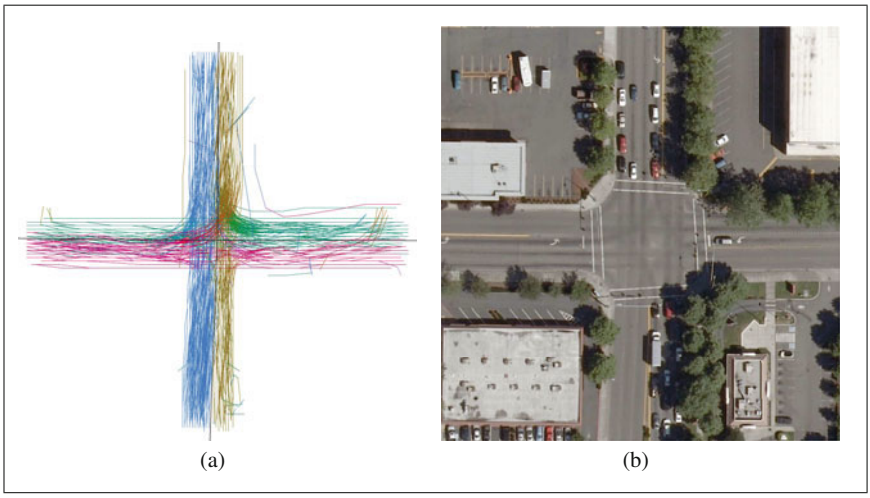


Fig. 7.13 The GPS traces in (a) are colored by their direction of travel. They came from the intersection in (b). The lanes in the GPS data are difficult to distinguish due to GPS noise.

If we model GPS noise as Gaussian as suggested in [13], this suggests that the spread of GPS traces across multiple lanes can be modeled as a mixture of Gaussians (GMM), as shown in Figure 7.14. Specifically, the GMM applies to the points where the GPS traces intersect a perpendicular line across the road. The mixture of Gaussians represents a continuous probability density expressed as a weighted sum of Gaussian distributions, as in Equation 7.1.

$$p(x) = \sum_{j=1}^{k} w_j \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(x-u_j)^2}{2\sigma_j^2}\right) \tag{7.1}$$

Here, k gives the number of lanes across the road. The variables μ_j and σ_j give the center and standard deviation of the GPS traces in the jth lane, respectively. The w_j

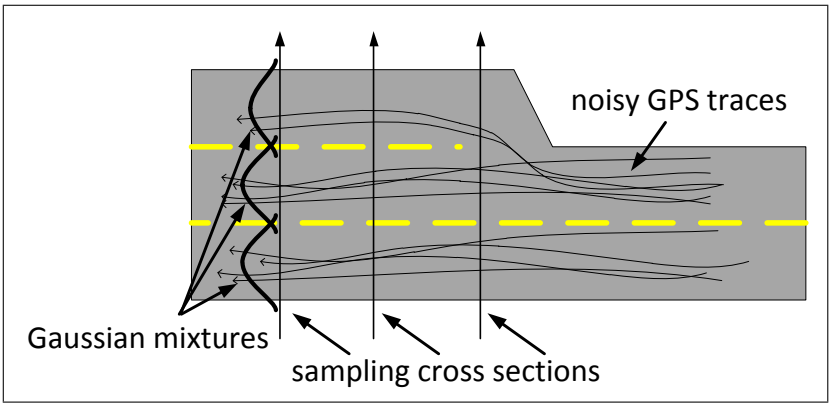


Fig. 7.14 We fit a Gaussian mixture to GPS traces to find lanes of traffic.

represents the fraction of GPS traces in each lane. While there are standard techniques to find k, $\mu_1 \ldots \mu_k$, and $\sigma_1 \ldots \sigma_k$, we found these did not work well. Instead, we developed a special expectation maximization (EM) algorithm to find the lane parameters. This algorithm is discussed in detail in [11]. The basic assumptions are:

- Lanes are evenly spread across the road, expressed as $\mu_j = \mu + (j-1)\triangle\mu$, $j = 1 \ldots k$.
- GPS traces have the same spread in each lane, i.e. $\sigma_j = \sigma$, $j = 1 \ldots k$.

In addition, we impose priors on $\triangle\mu$ and σ^2. To choose between different values of k (the number of lanes), we impose a penalty term that prefers lane widths of 5 meters. Figure 7.15 shows an example of how the fit improves with our restricted GMM. We tested this technique on roads approaching three intersections, with a separate model fit for each direction of travel. We looked at several cross sections along each road. Compared to a general GMM, our new technique achieved a lower percentage error in estimating the number of lanes as well as more consistent estimates of the lanes' spacing and spread.

7.3 Map Matching

It is common to convert GPS trajectories from a sequence of raw latitude/longitude coordinates to a sequence of roads. Knowledge of which road a vehicle is on is important for assessing traffic flow, inferring the vehicle's route, and predicting where the vehicle is going. The problem of converting a GPS sequence to a sequence of roads is called "map matching," and it is not as easy as it first appears. Figure 7.16 illustrates the problem, showing three points from a sequence and the multiple roads to which they could match.

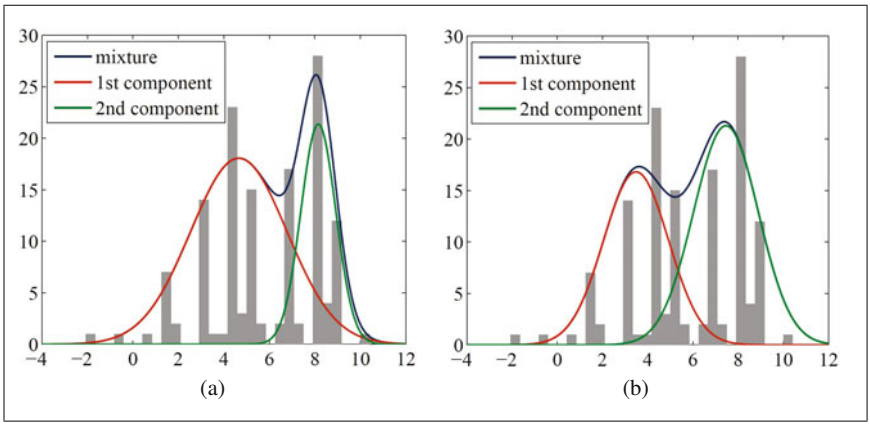


Fig. 7.15 Before imposing restrictions on our Gaussian mixture model, the resulting Gaussians are uneven, as in (a). After the restrictions, the two Gaussians have the same width as we expect of traffic lanes, as in (b).

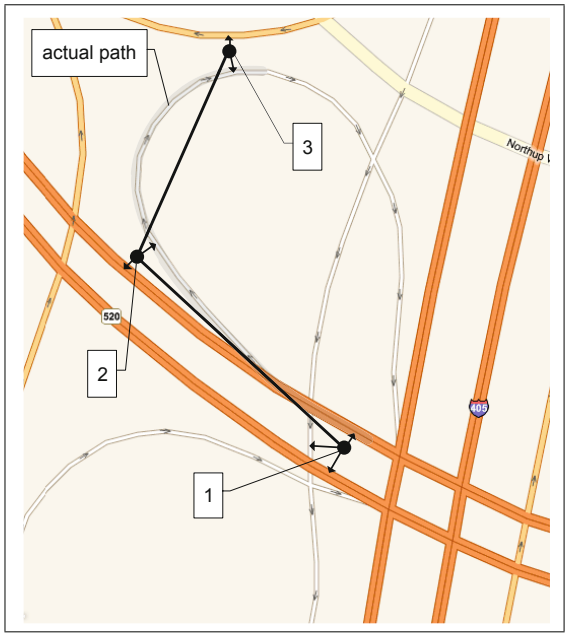


Fig. 7.16 Map matching is the process of finding which road corresponds to each GPS point. For these three GPS points, there are multiple nearby roads, but the true path is obvious considering continuity.

The first inclination for solving the map matching problem is to match the GPS point to the nearest road. This technique falls in the category of "geometric" algorithms for map matching, as explained in the survey articles in [6, 23]. The combination of GPS noise and multiple nearby roads, however, means that this technique often fails. In fact, matching to the nearest road generally serves only as a weak technique with which to demonstrate the superiority of more sophisticated algorithms.

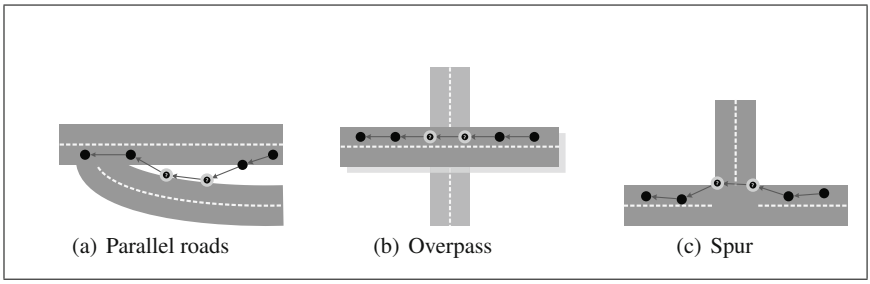


Fig. 7.17 These are three problematic scenarios for map matching. Simply matching to the nearest road will result in mistakes for the points with gray outlines.

Figure 7.17 shows three common scenarios where map matching can be difficult. In all three, multiple roads cause ambiguity in matching.

A step up in sophistication and robustness beyond geometric algorithms are so-call topological algorithms [6, 23] that pay attention to the connectivity of the road network. Clearly this would help solve the map matching problems illustrated in Figure 7.16 and Figure 7.17, where mistaken matches tend to violate the continuity of the drive. Representative of these algorithms are those that use the Fréchet distance to measure the fit between a GPS sequence and candidate road sequence [5, 8]. Here, candidate routes are assessed against the polyline created by connecting the GPS points with straight line segments. The candidate routes are created from the road network, and thus represent feasible driving paths which rule out sudden jumps to nearby roads. The Fréchet distance measures the dissimilarity between the GPS polyline and the candidate route. [5] describes the Fréchet distance as:

> Suppose a person is walking his dog, the person is walking on the one curve and the dog on the other. Both are allowed to control their speed but they are not allowed to go backwards. Then the Fréchet distance of the curves is the minimal length of a leash that is necessary for both to walk the curves from beginning to end.

In addition to noise in the GPS data, map matching algorithms must deal with occasional outliers. Also, the GPS sampling rate may be low. These problems can be addressed by probabilistic algorithms [23] that make explicit provisions for GPS noise and consider multiple possible paths through the road network to find the best one.

A new class of map matching algorithms has emerged recently that embrace both the topology of the road network and the noise and outliers in the GPS data, exemplified by [22]-[21]. These algorithms find a sequence of roads that simultaneously

come close to the noisy GPS data and form a reasonable route through the road network. The description below concentrates on [22], from our lab, as a well-tested and representative sample of algorithms like this.

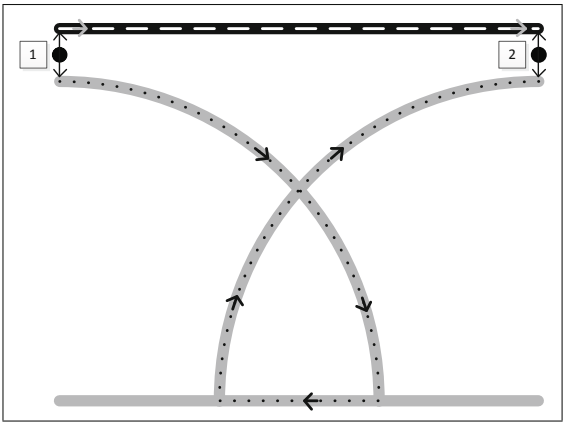


Fig. 7.18 GPS points 1 and 2 can match to either the black road or gray road. Matching to the black road is preferred, because the length of the resulting route is about the same as the distance between the GPS points.

7.3.1 Hidden Markov Model for Map Matching

There is a natural tradeoff in map matching between matching a GPS point to the nearest road and matching it to a road that makes sense in the context of other matched points. For example, Figure 7.16 shows three GPS points and nearby roads. None of the points is exactly on a road, due to GPS noise and possible geometric errors in the map. Thus, each GPS point has multiple candidate roads to which it could be matched, shown by the arrows at each point. However, it is obvious to us which route the driver took based on the continuity of the sequence of road matching candidates. There is a tradeoff between two criteria: the matched road should be close to the GPS point, but the sequence of roads should make a reasonable path. A hidden Markov model (HMM) makes this tradeoff explicit using probability distributions.

The first probability distribution for the HMM models the GPS observations and their possible matching roads. Specifically, this probability distribution models GPS noise, which can be represented by a Gaussian centered around the actual GPS measurement [13]. This means that roads that are farther away from the GPS point have a smaller probability of matching the GPS point. We measure the distance between the GPS point and the road as the great circle distance between the GPS point and the nearest point on the road. In Figure 7.16, these distances are the lengths of the

lines with arrows. Based on ground truth data from manual map matching, we estimated the standard deviation of this Gaussian distribution to be 4.07 meters for our data, which is a reasonable value for GPS noise [22].

The other probability distribution in the HMM, the transition probabilities, helps enforce the continuity of the route in terms of the transitions between pairs of GPS points. Intuitively, we want to avoid matching to a nearby road that would require a convoluted route to get to the next matched road. For example, in Figure 7.16, if GPS point "2" matched to the highway rather than the curving ramp, the subsequent route to any of the roads near point "3" would be long and inefficient.

The structure of the HMM requires these transition probabilities be a function of two, temporally adjacent road match candidates. We achieve this by considering the distance between the GPS points (great circle distance) and the length of the route that it would take to traverse this distance starting and ending on the road match candidates.

Figure 7.18 shows an example. If the two GPS points match to the black road, then the route distance between the two matched points is approximately the same as the great circle distance between the GPS points. If they instead matched to the gray roads, the route distance is much longer. In our experiments, we found that the absolute value of the difference between the great circle distance and route distance for correct road matches follows an exponential probability distribution, i.e.

$$p(d) = \frac{1}{\beta} e^{-d/\beta} \tag{7.2}$$

where d is the absolute difference between the great circle distance and route distance. This distribution is maximum at $d = 0$, and decreases monotonically for $d > 0$. In the example in Figure 7.18, d would be small if the two GPS points matched to the black road, leading to a larger probability. Matching to the gray road, d would be much larger, along with a lower probability.

To implement the computation of these transition probabilities requires that we run a route planner between all candidate pairs of matches for all temporally adjacent GPS points. To make this faster, we ignore all road matches that are more than 200 meters away from the GPS measurement.

The transition probabilities above are computed by looking at the differences in length between the GPS pairs and resulting routes. In [20] we looked at using time differences instead, with similar results. Time differences, however, can vary due to traffic speeds.

For each GPS point, the HMM looks at all the possible road matches within 200 meters. The HMM algorithm finds the optimal sequence of matched roads that maximizes the product of the observation probabilities (Gaussian for GPS noise) and transition probabilities (exponential on distance differences). Figure 7.19 shows some examples of how our algorithm works in spite of the problems illustrated in Figure 7.17. In Figure 7.19, the roads highlighted in white show the correctly matched route segment. The roads highlighted in gray show the route the vehicle would have to drive if each GPS point matched to the nearest road. This often results

in an unreasonable, circuitous route, necessary to pass through mistakenly matched roads.

Although the HMM is a robust technique for map matching, it can fail if the GPS point falls far from a road (e.g. a GPS outlier) or if the underlying map is wrong. For this reason, in [22] we show how to break the HMM at these points and recombine the matched route by removing offending GPS points.

(a) Parallel roads (b) Overpass (c) Spur

Fig. 7.19 These three scenarios correspond to the three in Figure 7.17. The HMM map matching algorithm works correctly. The black lines connect the GPS points, the white road shows the correct match, and the gray path shows the path necessary to satisfy naively matching the nearest road to each GPS point.

7.4 Destination Prediction

Drivers could benefit from a prediction of their destination. They could be informed of upcoming traffic jams, road construction, speed limit changes, and other warnings. With prediction, drivers could also be alerted about gas stations or charging stations along their route. Retailers near the driver's destination might pay a premium to deliver ads to vehicles predicted to arrive nearby. People doing a local Web search on a mobile device could get results clustered around their destination.

Our lab has developed a general, probabilistic framework for destination prediction [17, 19]. It operates on a vehicle's partial trajectory to predict where the trajectory will end. The framework depends on a discretization of the map, such as a tiling of square cells, as in Figure 7.20(a). In this discretization, each GPS point is represented by cell that contains it. Although the remainder of this discussion will use the cell-based discretization, we have also experimented with alternative discretizations, such as representing each GPS point by its nearest road intersection (Figure 7.20(b)), or road. In any case, the GPS trail is converted to a sequence of discretized points $C = \{c_1, c_2, c_3, \ldots, c_n\}$. In general, this is a partial trajectory, and a new element is added to the end whenever the vehicle encounters a new cell.

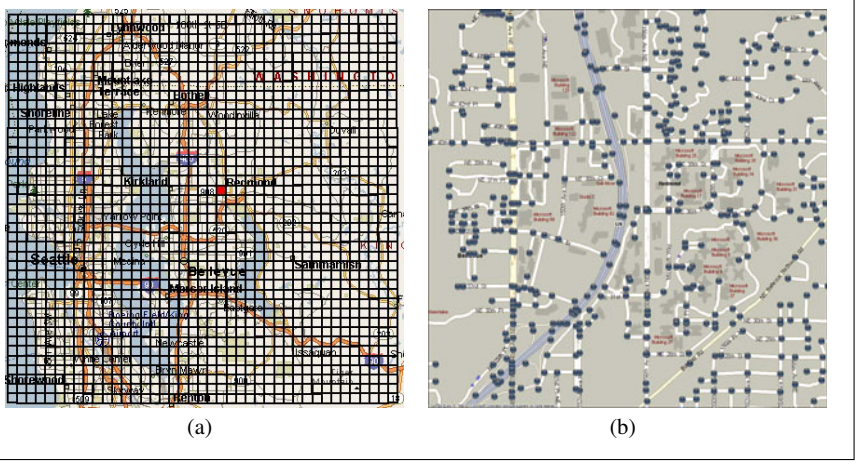


Fig. 7.20 For destination prediction, GPS points are discretized based on which grid cell they fall in (a), or which intersection they are close to (b).

Whenever a new element is added to the sequence C, our prediction is reinvoked to compute a probability for each cell on the map giving the probability that the current trip will end in that cell. Figure 7.21 shows an example of two partial trajectories starting near the center and going south. Cells with darker borders have higher probabilities.

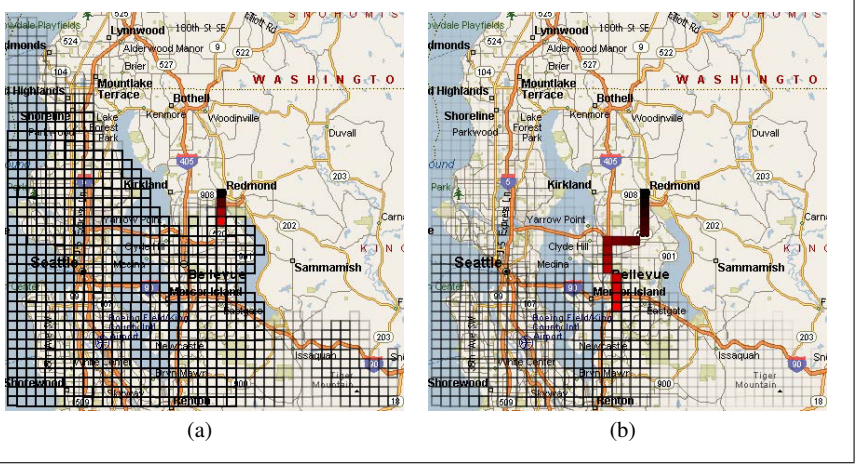


Fig. 7.21 As the trip progresses, we compute a destination probability for each cell. Cells with darker borders have higher destination probabilities.

We use Bayes rule to compute the destination probability of a cell c^* based on the partial trajectory C:

$$p(c^*|C) = \frac{p(C|c^*)p(c^*)}{\sum_{j=1}^{N} p(C|c_{(j)})p(c_{(j)})} \tag{7.3}$$

The three parts of the right side of Equation 7.3 are:

- $p(C|c^*)$ - Likelihood of partial trajectory C given the candidate destination cell c^*.
- $p(c^*)$ - Prior probability of c^* being a destination.
- $\sum_{j=1}^{N} p(C|c_{(j)})p(c_{(j)})$ - normalizing factor that makes cell probabilities sum to one.

The normalizing factor in the denominator is easy to compute after computing the numerator for each candidate destination c^*. The next two sections will discuss the computation of the likelihood and the prior.

7.4.1 Destination Likelihood from Efficient Driving

Our destination likelihood term is based on our assumption that drivers take fairly efficient routes to their destination. In order to test this, we gathered GPS data from 118 volunteer drivers representing 4300 different trips [17]. We discretized each trip with the grid in Figure 7.20(a), so each trip was represented by a sequence of grid cells. For each cell in each trip, we computed the driving time to the last (destination) cell using a conventional route planner.

We measured efficiency by tracking the minimum computed driving time to the destination cell as each trip progressed. That is, for each cell in the trip, we computed whether or not the driving time from that cell to the destination was the minimum driving time over all the cells up to that point in the trip. If drivers were perfectly efficient in their driving, and if our route planner gave accurate driving times, we would expect the minimum driving time to the destination cell to decrease with each cell-to-cell transition.

In fact, we found that drivers decreased the minimum driving time to their destination for 62.5% of the cell-to-cell transitions. This seems low, but could be due to the fact that drivers may be sensitive to traffic, while our route planner was not. Also, we computed driving times based on the road nearest the center of each cell, which may also be inaccurate. Despite this, we can give a simple equation for the likelihood term in Equation 7.3 based on efficient driving.

$$p(C|c^*) = \prod_{i=2}^{n} \begin{cases} p & \text{if } c_i \text{ is closer to } c^* \text{ than any previous cell in } C \\ 1-p & \text{otherwise} \end{cases} \tag{7.4}$$

Here p=0.625, based on our experiment. Also recall that $C = \{c_1, c_2, c_3, \ldots, c_n\}$, so n is the number of cells in the partial trajectory. By "closer" in Equation 7.4, we mean that c_i has the minimum driving time to c^* compared to all previous cells in the sequence. This likelihood term tends to reduce the probability of cells that the driver is driving away from, as illustrated in Figure 7.21. Equation 7.4 is invoked every time the driver enters a new cell, and it is evaluated for every cell c^* in the grid. As shown in Figure 7.21, the likelihood is generally reduced for cells that the driver appears to be passing up.

One challenge of implementing this likelihood based on driving times is the computation of driving times. For each cell in the trip $C = \{c_1, c_2, c_3, \ldots, c_n\}$, it is necessary to compute the driving time to each candidate cell c^* in the whole grid. This is the classic "single-source shortest path" problem where each cell in C serves as a starting point. In practice, we pre-compute the driving times between all pairs of cells in our grid. To save computation time and storage, we assume the driving time between two cells is independent of which cell is the start cell. Even with this assumption, for N cells in the grid, there are $N(N-1)/2$ distinct cell pairs. In the grid shown in Figure 7.20, there are $N = 41^2$ cells, which makes over 1.4 million distinct cell pairs. Using a convention route planner on a conventional PC, we can compute routes at a rate of about 60 per second, which translates to about 6.5 hours of computation time for our grid. We have recently switched to an algorithm called PHAST for much faster computation of "single-source shortest path" [12]. This is fast enough that we can compute driving times in real time without the need for pre-computation.

The next section discusses the computation of the prior probability, $p(c^*)$, which completes the terms in Equation 7.3.

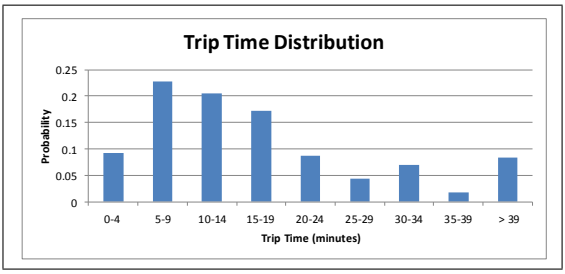


Fig. 7.22 We can use a distribution of trip times as a prior probability for destination prediction.

7.4.2 Destination Priors

We know that there are several other clues to a driver's destination in addition to the efficient driving likelihood discussed above. These other clues can be formulated

as prior probabilities over the grid of cells, denoted as $p(c^*)$ in Equation 7.3. This section discusses some useful priors.

7.4.2.1 Driving Time

Our intuition is that most car trips are fairly short. For instance, if someone started a trip in Los Angeles, we might be suspicious if we saw a high probability destination prediction for New York City. This intuition is correct, according to the U.S. 2001 National Household Transportation Survey (NHTS) [3]. Figure 7.22 shows a probability distribution of trip times from the 2001 NHTS. Given a starting cell in the grid, we can use our computed trip times and the trip time distribution to compute a driving time prior for each cell.

7.4.2.2 Ground Cover

The U.S. Geological Survey classifies each 30m x 30m square of the U.S. into one of 21 different ground cover types, such as water, pasture, urban, and commercial [1]. An image of their classifications for the region around Seattle, Washington USA is shown in Figure 7.23(a). We looked up the ground cover for each trip in our GPS data to create a ground cover prior. In our experiment, we found that the two most popular ground cover types at destinations were:

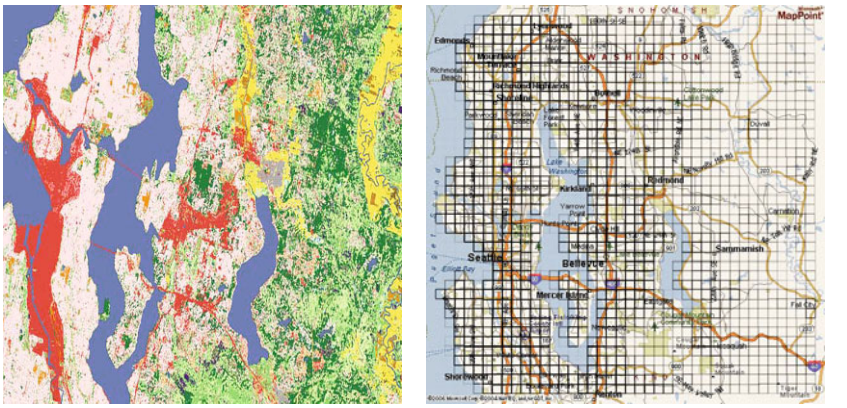
- Commercial/Industrial/Transportation "Includes infrastructure (e.g. roads, railroads, etc.) and all highly developed areas not classified as High Intensity Residential." [2]
- Low Intensity Residential "Includes areas with a mixture of constructed materials and vegetation. Constructed materials account for 30–80 percent of the cover. Vegetation may account for 20 to 70 percent of the cover. These areas most commonly include single-family housing units. Population densities will be lower than in high intensity residential areas." [2]

7.4.2.3 Other Priors

Other destination priors include a driver's personal destinations, i.e. places that he or she goes frequently, such as their home and work place. In [19] we implemented this as an "open world" model, raising the probability of previously visited locations, but leaving some non-zero probability for new places.

Another prior could be a function of which types of businesses are present. We have found, not surprisingly, that restaurants are more common destinations than dentists. This fact could help raise the probability of certain cells in the map.

Of course, all of these priors could vary by the time of day and day of week, as we expect certain types of destinations to rise and fall in popularity as time progresses.



(a) (b)

Fig. 7.23 (a) shows the USGS ground cover classes around the Seattle, Washington USA area. The resulting destination probabilities are shown in (b). Water and unpopulated areas are less popular destinations.

7.4.3 Route Prediction

While the focus of this section has been destination prediction, a related problem is route prediction, where the goal is predict a driver's entire route rather than just their stop location. In summary, here are three route prediction techniques we have explored:

Trip Observations: Observe several trips from a driver. When a new trip starts, attempt to find a good match with a previous trip, and use the matched trip as a route prediction [15].

Markov Model: For short sequences (1–10) of observed road segments for a particular driver, build a probabilistic model for which road segment comes next [16].

Turn Proportions: To predict which way a driver will turn at an intersection, look at which turns bring the driver closer to the most destinations. This tends to assign low probabilities to turns that result in dead ends or closed neighborhoods [18].

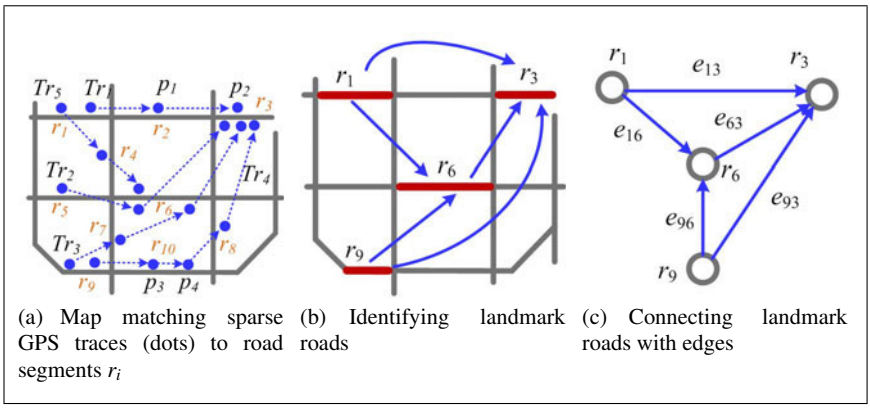
7.5 Learning Routes

It is easy to get driving directions from Web-based mapping sites and GPS-based navigation devices. When they were first available, these directions would give the fastest route assuming time-invariant driving speeds. More recently, the computed

routes are sensitive to current or predicted traffic speeds, helping to route around congestion.

Such directions, however, still fail to account for route subtleties that may affect speeds (e.g. turn delays) or driving pleasure (e.g. complexity). To address this, researchers have built routing algorithms that are heavily influenced by the behavior of real drivers whose routes are presumably closer to optimal.

This section presents two routing algorithms that exemplify learning by observing real drivers. The first, T-Drive, uses GPS traces from taxi cabs to create routes that are measurably faster than those from conventional routers. The second, TRIP, uses an individual's driving behavior to create better, new routes.



(a) Map matching sparse GPS traces (dots) to road segments r_i

(b) Identifying landmark roads

(c) Connecting landmark roads with edges

Fig. 7.24 These are the steps to creating a routing graph from taxi traces in T-Drive.

7.5.1 T-Drive: Learn from Taxis

T-Drive [28] looked at GPS traces from taxi drivers in Beijing, China to compute fast driving routes based on the assumption that taxi drivers are generally good at efficiently navigating through the city. This includes the fact that the best route may vary with the time of day as traffic conditions vary.

The research started with three months of GPS data from over 33,000 Beijing taxis. These taxis were already equipped with GPS sensors for the purpose of logging their locations for business intelligence. Unfortunately, the GPS points were separated in time by 2–5 minutes. This complicates the process of map matching, because the route taken between GPS samples is ambiguous. This led the same research team to create a new map matching algorithm specially aimed at sparsely sampled GPS traces [29]. This step of the process is illustrated in Figure 7.24(a).

It is unlikely that a newly requested start and end point will correspond exactly to a previous taxi route, especially a previous route at the same time of day as the

request. Thus, T-Drive identified "landmark" road segments that contained a significant number of taxi traces. These landmarks are not necessarily connected, but they serve as start points, end points, and waypoints for computing new routes. Figure 7.24(b) shows an illustration of these landmark road segments.

Two landmark roads are connected to each other if there have been enough taxi traces connecting them. These connections, illustrated in Figure 7.24(c), serve as edges in a "landmark graph". This graph is similar to the traditional road network graph used for conventional route planning, but its nodes (landmark roads) represent popular taxi roads, and its edges (landmark edges) represent taxi routes between the landmark roads.

Based on GPS observations of the taxis, the travel times associated with the landmark edges vary. T-Drive introduces a clustering technique called Variance-Entropy Clustering that splits the observed driving times on these edges into time periods throughout the day. The clustering splits time to produce periods of stable travel time distributions for each edge. Thus, there may be one distribution of travel times for 7:15 a.m. to 7:50 a.m., followed by a different distribution for 7:50 a.m. to 8:10 a.m., etc.

In planning a route, the route planner uses the landmark graph as a conventional road graph, but with two changes. First, a conventional road graph has a single cost scalar associated with each edge, such as the travel time or length of the edge. Instead, T-Drive has travel time distributions for each edge. Thus, prior to each route computation, the user must specify a value of α, $0 \leq \alpha \leq 1$, which indicates how close they are to a taxi's speed and aggressiveness. This value is used to choose one travel time for each edge from the edge's travel time distribution.

The second difference is that the T-Drive graph has time-varying travel times. The clustering process, combined with a choice of α, gives different travel times in different time periods for each edge. Thus, the expected travel time of an edge near the end of the route may change as the driver executes the route. This is a so-called "time-dependent fastest path" problem for which there are standard solutions, one of which T-Drive uses.

The first test of the T-Drive router used GPS trajectories from 30 drivers to show that T-Drive gave accurate driving times compare to real driving times. The researchers found that $\alpha = 0.6$ produced a good match between the two.

Having established that T-Drive gave accurate driving times, it was compared against synthetic route queries computed on a graph with constant edge speeds and on a graph with real-time traffic information. For about 60% of the queries, T-Drive gave faster routes than the graph with constant travel times, and it also outperformed the graph with real-time traffic. This is because the traffic-sensitive router had to resort to constant speeds on roads where it had no data and because T-Drive could implicitly account for subtleties like pedestrians blocking the road and cars parking.

Finally, T-Drive was compared against real drivers taking T-Drive routes and routes computed from Google Maps™. 81% of T-Drive's routes were faster, and saved 11.9% of driving time on average.

Recently, an extension of T-Drive system performs a self-adaptive driving direction service for a particular user. In this self-adaptive service, a mobile client (typi-

cally running on a GPS-phone) records a user's driving routes and gradually learns the user's driving behavior from the GPS logs [27]. Specifically, a user's driving behavior is represented by a vector of custom factors, $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \ldots\}$, where α_i denotes how fast the user could drive on the landmark edge i as compared to taxi drivers. This is different from the original T-Drive system using only one custom factor for a user. The motivation of this work is that an individual's driving behavior varies in routes and driving experiences. For example, traveling on an unfamiliar route, a user has to pay attention to the road signs, hence drive relatively slowly. Meanwhile, after becoming experienced in driving a user is likely to drive faster than he/she did before.

7.5.2 Learn from Yourself

While T-Drive helps drivers imitate taxis to find the fastest route, Trip Router with Inividualized Preferences (TRIP) helps drivers find routes that match their implied preferences [27]. Using GPS data from 102 different drivers who made 2517 total trips, TRIP was designed to compute new routes that paid attention to both traffic speeds and individual driver's preferences for certain roads. TRIP was based on the fact that drivers do not always choose the fastest route to their destination, even if they are familiar with the area. This is illustrated in Figure 7.25, which shows four different routes for a driver's morning commute. The driver's usual route is significantly different from the shortest distance route, the fastest route computed based on traffic speeds, and the fastest route computed by Microsoft MapPoint®. In fact, by looking at the GPS trips in the TRIP study, drivers choose the fastest route only about 35% of their trips. These are drivers who are familiar with the area.

Interestingly, the work in TRIP was preceded by necessary work on map matching, similar to the work on T-Drive above, resulting in another HMM-based algorithm [20].

TRIP worked in two steps. The first was to assign time-varying road speeds to every road segment encountered by all the drivers. TRIP's road speeds were computed as the average speed seen on each road, separated into 15-minute time intervals to account for variations throughout the day. There was one set of speeds for weekdays and another set for weekends. If a road had no GPS data for a certain time interval, its speed was taken as the system-wide average of roads of the same type (e.g. arterial, highway, ramp) for that time interval. If there was no data available for a time interval, the road's speed was taken as its speed limit. Similar to T-Drive, these real time road speeds provide the basis for computing fast routes between any two points.

The efficient routes computed from measured road speeds served as a baseline for incorporating driving preferences that go beyond efficiency, which was TRIP's second step. TRIP examined all the recorded trips for each driver to compute each driver's "inefficiency ratio" r. The inefficiency ratio reflects the amount by which a driver's actual routes are less time-efficient than the computed fastest routes. For a

given route, the numerator of the ratio is the duration of the fastest route between the endpoints, computed using TRIP's road speed estimates from above. The denominator is the driver's actual driving time. This r is almost always less than one, unless the driver has taken the fastest route or, occasionally, has found a route faster than that based on TRIP's speed estimates. The driver's average efficiency ratio, r, represents the amount he or she is implicitly discounting the time cost of their preferred roads.

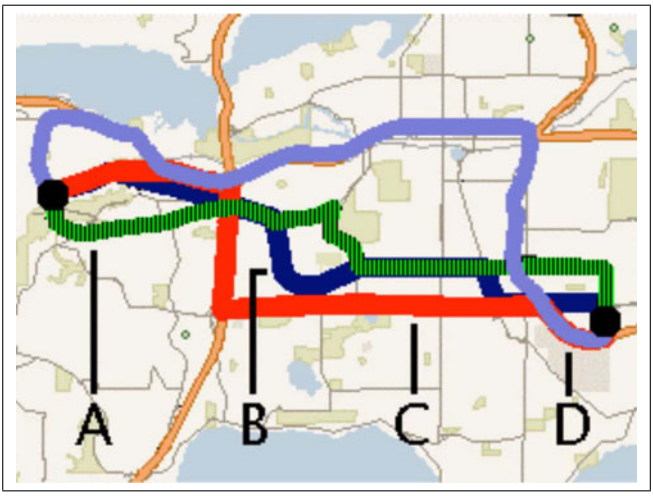


Fig. 7.25 This shows four routes for a drivers morning commute. (A) shows the drivers usual route, (B) is the shortest distance route, (C) is the fasted route based on our own GPS study, and (D) is the fastest route according to MapPoint®.

A driver's efficiency ratio is used to discount the cost of their preferred roads for computing routes. Specifically, TRIP computes the time cost of road segment i as t_i, based on GPS observations from all drivers as explained above. For planning a route, TRIP uses the driver's average efficiency ratio as follows to compute new costs c_i of every road segment:

$$c_i = \begin{cases} \bar{r}t_i & \text{if road } i \text{ previously driven by this driver} \\ t_i & \text{otherwise} \end{cases} \tag{7.5}$$

The effect of this discounting is a reduction in cost of roads the driver has already driven, meaning they are more likely to be chosen by the otherwise conventional route planner.

In tests, TRIP's computed route matched the driver's actual route in 46.6% of the test cases. This compares favorably with computing routes based on the estimated road speeds, which matched the test trips only 34.5% of the time. Using a traditional route planner based on static speed limits, the computed routes matched only 30% of the time.

7.6 Summary

This chapter discussed the use of GPS traces to first construct a road map, match new GPS traces to an existing road map, and then use matched traces to predict a driver's destination and to compute optimal routes. Each of these efforts depends on an easily gathered repository of GPS traces of drivers as they drive normally, which demonstrates the value of logged GPS data.

References

1. Multi-Resolution Land Characteristics Consortium (MRLC), National Land Cover Database. http://www.mrlc.gov/
2. NLCD 92 Land Cover Class Definitions. http://landcover.usgs.gov/classes.php
3. U.S. Department of Transportation, F.H.A. 2001 NHTS (National Household Travel Survey). http://nhts.ornl.gov/introduction.shtml#2001
4. Building and road extraction from aerial photographs. Systems, Man and Cybernetics, IEEE Transactions on **12**(1), 84 –91 (1982)
5. Alt, H., Efrat, A., Rote, G., Wenk, C.: Matching planar maps. In: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, SODA '03, pp. 589–598. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2003)
6. Bernstein, D., Kornhauser, A.: An introduction to map matching for personal navigation assistants
7. Besl, P.J., McKay, N.D.: A method for registration of 3-d shapes. IEEE Trans. Pattern Anal. Mach. Intell. **14**, 239–256 (1992)
8. Brakatsoulas, S., Pfoser, D., Salas, R., Wenk, C.: On map-matching vehicle tracking data. In: Proceedings of the 31st international conference on Very large data bases, VLDB '05, pp. 853–864. VLDB Endowment (2005)
9. Bruntrup, R., Edelkamp, S., Jabbar, S., Scholz, B.: Incremental map generation with gps traces. In: Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE, pp. 574 – 579 (2005)
10. Cao, L., Krumm, J.: From gps traces to a routable road map. In: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '09, pp. 3–12. ACM, New York, NY, USA (2009)
11. Chen, Y., Krumm, J.: Probabilistic modeling of traffic lanes from gps traces. In: Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '10, pp. 81–88. ACM, New York, NY, USA (2010)
12. Delling, D., Goldberg, A.V., Nowatzyk, A., Werneck, R.F.: Phast: Hardware-accelerated shortest path trees (2011)
13. van Diggelen, F.: Gnss accuracy: Lies, damn lies, and statistics
14. Fathi, A., Krumm, J.: Detecting road intersections from gps traces. In: Proceedings of the 6th international conference on Geographic information science, GIScience'10, pp. 56–69. Springer-Verlag, Berlin, Heidelberg (2010)
15. Froehlich, J., Krumm, J.: Route prediction from trip observations. In: Society of Automotive Engineers (SAE) 2008 World Congress
16. Krumm, J.: A markov model for driver turn prediction. In: Society of Automotive Engineers (SAE) 2008 World Congress
17. Krumm, J.: Real time destination prediction based on efficient routes. In: Society of Automotive Engineers (SAE) 2006 World Congress

18. Krumm, J.: Where will they turn: predicting turn proportions at intersections. Personal Ubiquitous Comput. **14**, 591–599 (2010)
19. Krumm, J., Horvitz, E.: Predestination: Inferring destinations from partial trajectories. In: Proceedings of the 8th International Conference on Ubiquitous Computing (UbiComp 2006)
20. Krumm, J., Letchner, J., Horvitz, E.: Map matching with travel time constraints. In: Society of Automotive Engineers (SAE) 2007 World Congress
21. Lou, Y., Zhang, C., Zheng, Y., Xie, X., Wang, W., Huang, Y.: Map-matching for low-sampling-rate gps trajectories. In: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '09, pp. 352–361. ACM, New York, NY, USA (2009)
22. Newson, P., Krumm, J.: Hidden markov map matching through noise and sparseness. In: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '09, pp. 336–343. ACM, New York, NY, USA (2009)
23. Quddus, M.A., Ochieng, W.Y., Noland, R.B.: Current map-matching algorithms for transport applications: State-of-the art and future research directions. Transportation Research Part C-emerging Technologies **15**, 312–328 (2007)
24. Rogers, S., Langley, P., Wilson, C.: Mining gps data to augment road models. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '99, pp. 104–113. ACM, New York, NY, USA (1999)
25. Schroedl, S., Wagstaff, K., Rogers, S., Langley, P., Wilson, C.: Mining gps traces for map refinement. Data Min. Knowl. Discov. **9**, 59–87 (2004)
26. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, vol. 1, pp. I–511 – I–518 vol.1 (2001)
27. Yuan, J., Zheng, Y., Xie, X., Sun, G.: Driving with knowledge from the physical world (to appear). In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '11
28. Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G., Huang, Y.: T-drive: driving directions based on taxi trajectories. In: Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '10, pp. 99–108. ACM, New York, NY, USA (2010)
29. Yuan, J., Zheng, Y., Zhang, C., Xie, X., Sun, G.Z.: An interactive-voting based map matching algorithm. In: Proceedings of the 2010 Eleventh International Conference on Mobile Data Management, MDM '10, pp. 43–52. IEEE Computer Society, Washington, DC, USA (2010)