Green Driver: AI in a Microcosm

Jim Apple, Paul Chang, Aran Clauson, Heidi Dixon, Hiba Fakhoury, Matt Ginsberg,
Erin Keenan, Alex Leighton, Kevin Scavezze and **Bryan Smith**
On Time Systems, Inc.
355 Goodpasture Island Road, Suite 200
Eugene, OR 97401

Abstract

The Green Driver app is a dynamic routing application for GPS-enabled smartphones. Green Driver combines client GPS data with real-time traffic light information provided by cities to determine optimal routes in response to driver route requests. Routes are optimized with respect to travel time, with the intention of saving the driver both time and fuel, and rerouting can occur if warranted. During a routing session, client phones communicate with a centralized server that both collects GPS data and processes route requests. All relevant data are anonymized and saved to databases for analysis; statistics are calculated from the aggregate data and fed back to the routing engine to improve future routing. Analyses can also be performed to discern driver trends: where do drivers tend to go, how long do they stay, when and where does traffic congestion occur, and so on. The system uses a number of techniques from the field of artificial intelligence. We apply a variant of A* search for solving the stochastic shortest path problem in order to find optimal driving routes through a network of roads given light-status information. We also use dynamic programming and hidden Markov models to determine the progress of a driver through a network of roads from GPS data and light-status data. The Green Driver system is currently deployed for testing in Eugene, Oregon, and is scheduled for large-scale deployment in Portland, Oregon, in Spring 2011.

1 Overview

Green Driver is a routing app for GPS-enabled smartphones that is designed to provide drivers with effective, efficient routes based on the best available real-time information regarding traffic and other conditions.

Of course, we are hardly alone in this ambition. Previous (and, to some extent, current) attempts to provide such services have been frustrated by the lack of good real-time information itself; the development of intelligent transportation systems (ITS) generally has been limited by immature communications technology and expensive infrastructure for vehicle detection. The historical goal of ITS has been to exploit information and communications technology to better manage transportation systems, to improve driver and vehicle safety, to optimize travel times and to save fuel. The recent wide appearance of smartphone platforms and free navigation apps has created an opportunity to manage driver behavior in a cohesive, system-wide framework. By tracking the GPS positions of mobile phones, we can sample traffic density and movement patterns in real time without the need for expensive infrastructure. We can then use this collective data for the purposes of system-wide optimization. A traffic-aware navigation app can then be provided with the real time information needed to compute optimal routes that minimize travel time and fuel usage.

Unfortunately, there is a chicken-and-egg problem here. It is only widespread use of GPS apps on mobile phones that allows the generation of the real-time traffic data that the apps themselves need to be effective. As a result, while many efforts have been made to obtain traffic data from cell phones (Intellione, Mobile Millenium, Waze) or to develop user-friendly navigation apps (Garmin, Google, Magellan, TomTom), there has yet to be a fully successful implementation of a real time traffic-aware navigation app.

We believe that this problem can be overcome by providing potential users with value in another form. There is an additional source of data that has a profound impact on routing decisions and is available immediately without the "crowdsourcing" needed to construct real-time traffic information. That data involves traffic lights. The fundamental innovation in Green Driver is that the presentation of routes that are traffic-light aware provides sufficient adoption incentive to overcome the chicken-and-egg problem described above. The routes are constructed by combining client GPS data with real-time traffic light information provided by cities to respond to user route requests. Routes are optimized with respect to travel time, with reroutes occurring if conditions change enough for a reroute to save the user significant time. During a routing session, client phones necessarily communicate with a centralized server in order to update their positions and receive new routes when needed; that server also collects and anonymizes GPS data from the users. Green Driver is unique in its use of city-provided traffic light information in its routing methodology.

Green Driver is positioned at the convergence of many trends. First the growing number of people carrying smartphones means that it is common for individuals to have one or more apps running for long periods, including apps that communicate over a data network. Second, many smartphone users seem comfortable allowing their apps to communicate personal information to centralized servers, especially if they have some assurance that this information will not be shared without their permission. Third, software development kits make it a fairly simple matter to design and implement GPS-centric apps for many models of smartphones. Fourth, digital street maps of high quality are now available in the public domain. Fifth, cities increasingly include remote monitoring and control of lights in their traffic infrastructure. Sixth, global warming and over-reliance on fossil fuels have led consumer to seek ways to reduce their energy footprint and energy costs. Finally, most of us are commuters to some extent, with a natural desire to lessen the time spent on the road while still obeying traffic controls.

A general goal for smartphone apps is to provide a service that is both simple to use and fills a need, perhaps even a need that wasn't recognized until the app's introduction. If the app is also fun to use, so much the better. With traffic-routing apps there is also a safety concern: Users should not have to take their eyes from the road to interact with the app. Green Driver seeks to address these design goals, both by handling spoken route requests from users and by providing audio turn-by-turn routing instructions. The graphical user interface renders a perspective-view map bordered by simple-to-understand navigation widgets and also inform drivers of relevant traffic controls, such as the speed limit for the current road segment and the expected status (green or red, expected delay if red) of upcoming traffic lights. We also take proactive steps to encourage safer driving, such as:

1. Presentation of a route that is designed to hit green lights only if the driver conforms to posted speed limits,
2. An on-screen speedometer that displays the current speed and current speed limit, and no other numbers, and changes color if the speed limit is exceeded, and
3. Differential presentation of information to encourage safer driving. As an example, red lights are generally displayed with the number of seconds until the light turns green. But green lights are only displayed that way if the driver can "make" the green light without speeding. This encourages drivers to arrive comfortably at green lights instead of racing toward them, and a preliminary deployment of this to half of our users suggested that it did indeed have the desired effect.[1]

Our focus in this paper is on the specifically AI-related problems that we have faced in developing the Green Driver system. Not all of these problems are solved and in many cases, we have used solutions that appear to work acceptably in practice but which can doubtless be improved considerably with better understanding. We focus here on our search engine for the routing problem and our method for analyzing noisy GPS data with dynamic programming and hidden Markov models. Green Driver includes many aspects not discussed here, including (but hardly limited to) the UI implementation, methods to ensure user privacy, geocoding and the associated string matching issues, database design, scalability, mining of GPS data for marketable traffic studies, and interaction with city traffic engineers to improve our access to traffic light information.

[1]At least initially. Drivers appeared to slow down when this feature was introduced, but some time later, they habituated it and their behavior returned to previous levels.

2 Light-Controller Model

As the traffic signals themselves underlie the routing problem that we solve for our users, let us begin by describing the lights and how we model them. Our goal here is not to provide the wealth of detail that would typically appear in a paper on traffic engineering, but to present the information that will eventually bear on the routing problem itself.

Most traffic light controllers provide eight standard phases. Each phase may control one or more movements (vehicular, bicycle, pedestrian, transit), and the duration of each phase is constrained by a set of timing parameters. The phases are organized in a sequence called a *cycle*; the total time required to complete the cycle is called the *cycle length*. Each phase in the cycle may be skipped if there is no demand, have a fixed time, or have an extendable time depending on demand. Demand is often measured by detectors such as induction coil sensors. For major thoroughfares with smaller crossing streets, it is often desirable to allow platoons of vehicles to traverse several traffic lights in succession without stopping. These traffic lights are said to be *coordinated*, in that the phases of each light corresponding to thoroughfare movement have a guaranteed green time within each cycle, and the activation times of these green times are offset to reflect the average travel time between these lights. Coordinated lights have the same fixed cycle length, typically on the order of 60-100 seconds.

"Coordinated" is not the same as what are conventionally called "timed". For a timed light (e.g., what is typically found in a busy downtown area), the signal timing is fixed and the phase of the light can be predicted absolutely given the state of a shared clock. For coordinated lights, the light state is known only for particular windows within the overall cycle; the windows are selected to support the notional platoon of vehicles moving along the major road. Outside of this window, the light may be red or green, depending on the presence (or absence, respectively) of vehicles on side streets competing with those on the major thoroughfare.

The traffic light controller model that we have developed is essentially an extension of the standard light controller model (Koonce et al. 2008). We predict the future state of a light by applying standard controller rules with predefined sets of timing parameters to live light status updates from the city traffic light system. We extend this predictive capability to phases that do not have a fixed green time in the cycle by building stochastic models to track the level of demand, the green times for these phases, and the wait times for the next green time. This light controller model is used to compute the likelihood of a vehicle arriving during the green time for any particular light, or a probability distribution of wait times for the next green if the vehicle arrives when the light is red. These delays are what is needed by the eventual routing algorithm itself; note that the generated probability distributions depend strongly on the arrival time of the vehicle. As we will see in the next section, this renders basic best-first search algorithms such as A* search nonoptimal.

For illustration purposes, let us give a high-level description of one such stochastic model for a coordinated light in which the cycle length is fixed and all times refer to number of seconds after the start of the cycle. Given an arrival time a, our goal is to computed $p_a(d)$, a probability distribution on the delay time d given the arrival time a.

Assuming that the vehicle in question is on the major road as opposed to a side street, we break the analysis into three cases. First, if the vehicle is going to arrive during a period when the light is guaranteed to be green, the delay is obviously 0. Second, if the vehicle is going to arrive during a period when the light may be green or red *after* the end of the current cycle, we use average historic information for this light to evaluate the projected delay as a function of the arrival time in the cycle. Finally, if the vehicle is going to arrive during a period when the light may be green or red *during* the current cycle, we continue to use average historic information as our predictor but must now condition on the current state of the light as well. (For example, a light that is currently green outside of the guaranteed green time is guaranteed not to change to red if there will not be enough time for it to change back to green before it is required to do so by the underlying coordination mechanism.)

We have constructed other stochastic models for uncoordinated lights, or for movements that depend on more than one phase, such as protected-permissive left turns.

3 Stochastic Shortest Path Problem

The stochastic nature of traffic lights (and of traffic itself) cause the route planning problem to be stochastic as well. This "stochastic shortest path problem" (SSP), and has been studied by numerous authors (Bander and White 2002; Bertsekas and Tsitsiklis 1991; Boyan and Mitzenmacher 2001; Hall 1986; Wellman et al. 1995).

Green Driver involves a slightly modified version of the SSP, since there may be movement restrictions present at some times but not at others (e.g., no left turn at a particular intersection during rush hour). The goal of the calculation remains the same, however: Given an origin and destination, compute a route that minimizes expected travel time.

A* is nonoptimal for SSP, principally because travel time distributions along segments depend on the arrival time at the beginning of the segment (Bander and White 2002; Boyan and Mitzenmacher 2001). While a robust dynamic programming algorithm has been developed to solve SSP, and many optimized variants have been developed over the past several years, we believe these algorithms remain too slow for widespread deployment in systems such as Green Driver.[2] The dynamic programming approach is essentially a Dykstra-like algorithm, relying on a flood fill method and losing the heuristic savings provided by A*. Green Driver will need to serve many thousands of simultaneous users in a city of even modest size, updating their routes approximately once a second (this being the frequency with which signal data is generally updated, or the time needed for a vehicle to move a significant distance). It is therefore necessary to respond to a routing query in a handful of milliseconds, and we have not been able to achieve these levels of performance without using A* or something like it.

[2]These other algorithms also typically assume that arc costs are distributed lognormally, an assumption that is invalid in our domain, where multimodal distributions arise from "missing" any particular light.

From a theoretical perspective, the view we take is that the "value" of a path to a particular en route point is not just a time (e.g., a deterministic value, or the average value in the stochastic case). Instead, the value is taken to be the probability *distribution* of arriving at the location at a given time, given some particular path from the origin point. From $p_{in}(t)$ giving the probability distribution of arriving at an intersection at time t, we compute the probability distribution for *leaving* the intersection as

$$p_{out}(t) = \int p_{in}(x) p_x(t-x) dx, \qquad (1)$$

where p_x is the light delay function described in the previous section. The distribution p_{out} is then similarly convolved with the distribution corresponding to traversing a particular street segment, and the algorithm proceeds.

While what we have described is sound in theory, compromises must be made in practice. First, it is not possible to achieve satisfactory computational performance if we fully convolve the various distributions as in (1). Instead, each distribution is approximated using a handful of points, the approximations are "integrated" as sums, and a new approximation is then constructed from the result.

Incorporation of a heuristic into A* is more important. If $g(n)$ is the cost of reaching some intermediate node n and there is an optimistic heuristic function h that estimates the cost of reaching the goal from n, then the search is terminated by pruning any node for which $g(n) + h(n)$ exceeds a known upper bound on the total route cost.

If g and h are distributions, we can prune if the expected cost $\overline{g+h}$ exceeds the expected cost of some known path from origin to goal. We also need the heuristic h to be uniformly optimistic in that the associated distribution is uniformly to the left of the actual cost function c. In other words, we need $\int_0^x h(t)dt \geq \int_0^x c(t)dt$ for all x.

Finding a heuristic that provably satisfies this requirement appears not to be practical. Instead, we assume – wrongly – that the expected value $\overline{g+h} = \overline{g} + \overline{h}$ and use a heuristic h that is optimistic in expected value only. We are aware that this approach is theoretically unacceptable but know of no practical alternative.

A real-world example in which stochastic A* results in a suboptimal route appears in Figure 1. The top figure shows the shortest-distance route between points A and E (movement is from right to left in the figure), including a slight left turn at B. This is at odds with the route computed by A*, shown in the bottom figure and ignoring the left turn at B. The problem arises because our implementation of A* cannot recognize the fact that the time distributions at the current search node depend conditionally upon the distributions nearer to the root of the search tree.

Examples such as this one notwithstanding, however, A* produces routes that we have found to generally be very rea-

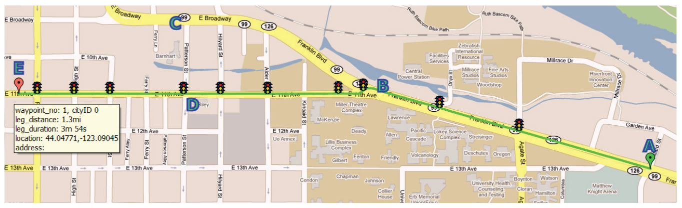
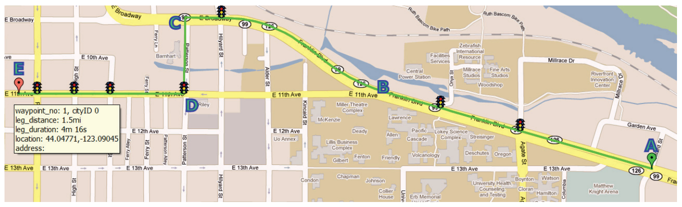

Figure 1: Optimal (top) and suboptimal (bottom) routes, reflecting problems with A*

sonable in the real world; gas mileage for our users appears to increase by some 5%.

Before moving on, let us describe the heuristic h. Euclidean distance does not suffice, since it ignores routing considerations, speed limits, and traffic lights. The number of locations in any particular city is also large enough to preclude the possibility of computing h in advance.

What we do is divide any particular city into small rectangular areas (blocks, as it were, but larger). For any pair (x, y) of blocks, we precompute the minimum time needed to get from block x to block y, assuming that the drivers move at the speed limit and making optimistic (but realistic) assumptions regarding the lights they encounter. Now given a departure point d in cell c_d and an arrival point a in c_a, the time to travel from d to a is at least the time to get from c_d to c_a plus the time needed to get from d to the boundary of c_d and from the boundary of c_a to the arrival point a.

4 Analysis of GPS Data

A smartphone running Green Driver transmits a stream of GPS data as it moves through a road network. In addition to position, a GPS datum can include speed, compass heading, timestamp, expected positional error, and so on. All relevant data are anonymized and saved for analysis; statistics are generated from the aggregate data and returned to the routing engine to improve future routing. Driver trends can be discerned via similar analyses: Where do drivers tend to go, how long do they stay, when and where does traffic congestion occur, and so on. Near real-time analysis of each GPS stream is needed if drivers are to be routed away from emergent traffic problems.

The GPS analysis is made significantly more difficult because GPS chips in smartphones are generally less accurate than those in dedicated GPS devices. In addition, the GPS readings from smartphone GPS chips are often affected by on-chip processing similar to that found in Kalman filtering (see, for example, the excellent Wikipedia article on this topic). In this context, *filtering* means determining the current position, speed, and heading by combining recent GPS readings and a correction step based on the current reading. Filtering improves positional accuracy when future GPS readings are predictable, such as when a car is traveling at a constant speed on a constant heading. But if a car changes speed quickly or turns a sharp corner, filtering will have an adverse effect on positional accuracy as filtered GPS readings "over-predict" the current position based on previous readings. The sequence of filtered GPS readings for a car making a rolling 90-degree turn at an intersection will often form a jug-handle shape caused by the filtering mechanism wrongly predicting continued straight-ahead and constant-speed travel. (See Figure 2 for a representative example.)

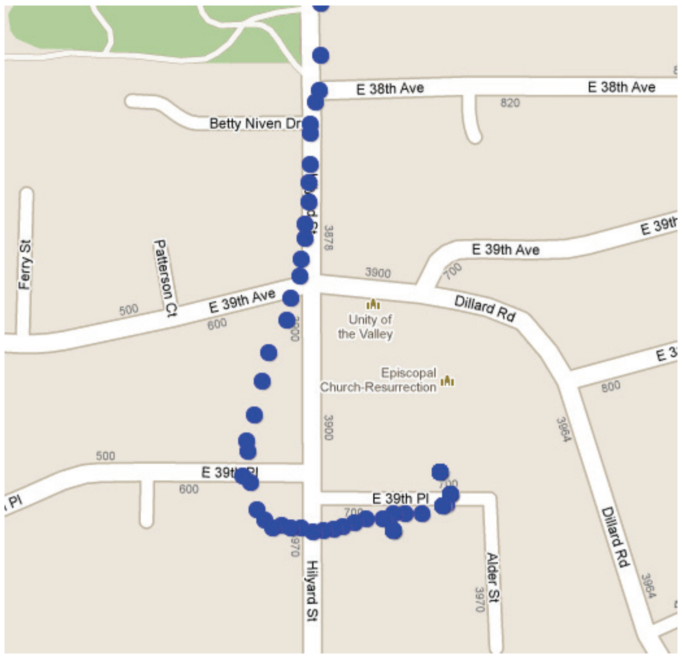

Figure 2: Positional errors in GPS signals for a westbound vehicle turning north

Inaccuracies in map data add to this complexity. If the polylines defining road segments have errors, the allowed transitions in the map can differ from reality. This can happen if the map is old or omits significant recent roadwork, or if some features encountered during travel (driveways, parking lots, etc.) do not appear on the map at all.

Finally, drivers themselves are not entirely predictable. A GPS stream might include data generated by a smartphone in a car performing illegal or otherwise surprising maneuvers, such as speeding excessively, making a U-turn, driving in reverse, stopping briefly to pick up or drop off passengers, and so on. In spite of all these difficulties, we need to accurately locate our users if we are to generate real-time traffic information based on their reported positions, and we need to be able to analyze their paths if we are to provide realistic analyses of their origins, destinations and routes.

Route Discovery with Dynamic Programming: When analyzing a driver's signals, the first thing we do is what we call route *discovery*: Given an atlas (locations of road segments, allowed transitions at intersections, speed limits, etc.), and given a sequence of GPS readings produced by a single device, we need to find the legally drivable route through the map that best matches the data provided.

Two considerations simplify this problem. First, because the time between successive GPS readings is usually smaller than the time needed to traverse a single road segment, we can assume that our desired route is "drivable" in that consecutive GPS readings will be generated either on a single segment or on adjoining segments. Second, GPS readings are error-prone, but they are generally close enough to allow us to eliminate most road segments from consideration when mapping GPS readings to segments.

We solve this problem using dynamic programming (DP) to find the lowest-scoring path through the road network for a given sequence of GPS readings. If we call the directed segments "states" and the GPS readings "observations", and if we observe that state transitions are limited by our drivability assumption, then the number b of transitions from a given state is approximately four (one for each possible compass direction from the current position, including remaining on the current segment). Given N states and T observations, the time needed for a dynamic programming approach to find the best scoring path will be $O(bNT)$ (Cormen et al. 2001).

A direct dynamic programming approach remains impractical, however. As an example, the road network for Lane County, Oregon, has roughly 57,000 directed segments, and a smartphone in a car can easily generate a thousand GPS readings during travel, so that $4 \times 57,000 \times 1000 = 228,000,000$ score updates will need to be computed for a single route. We simplify the calculation via the "close enough" assumption, ignoring state transitions where the successor state is more than a specified bound from the position reported by the GPS. This eliminates most paths from consideration, and usually gives good results, although a single or small number of inaccurate GPS readings will occasionally produce a non-optimal path or fail to find a path satisfying the bound at each time.

The details of the per-observation score used for dynamic programming will not be presented here, but it is basically a distance measure: nearness between GPS reading and road segment contributes to the score, and so does a term comparing how far a car would actually travel between consecutive GPS readings (using speed as reported by the GPS device) and how far it would have to travel along road segments if a given state transition occurred at that time.

Route Scoring with Hidden Markov Models: The second phase of analysis is what we call *scoring*. We take the GPS stream and legally drivable route from the dynamic program as input, score the GPS readings and route versus a model, improve the associations of the GPS readings with road segments and intersections on the route, and annotate the route in whatever ways seem helpful. This is important because we need to know much more than the overall route that the vehicle took as the readings were produced; we want to know exactly where it was at each datum, how quickly it was moving, and so on.

The goal in the scoring phase is to assign probabilities to events as a car traverses the different elements of the route, primarily road segments and intersections. For instance, if GPS readings indicate that a car's speed varies substantially from the expected speed along a segment, then a low probability should be assigned to the car's traversing that segment. A similarly low probability should be assigned to a car's stopping at an intersection that has no stop sign or traffic light. In this context, scoring can also be used to show anomalous driver behavior.

By improving the associations of GPS readings with route elements, we are able to correct for small errors in the GPS data, including those caused by filtering. This correction is guided by standard traffic control rules, in that control devices such as speed limit signs, traffic lights, and stop signs cause drivers to behave in predictable ways. If, for example, a sequence of GPS readings indicates that a car passed through an intersection having a stop sign and then stopped near the beginning of the next road segment, we will conclude that it is far more likely that the car stopped in accordance with the stop sign and then continued moving.

By annotation, we mean labeling individual GPS readings to indicate the car's status as it traverses the route. In the previous example, we might annotate the relevant GPS readings with "Stopped for stop sign at intersection." The annotations give the story of the car's progress along its route.

We solve this problem by using a hidden Markov model (HMM) (Rabiner 1989, or many others). Continuing to think of individual GPS readings as observations, we discretize the route by dividing road segments into subsegments of roughly equal transit time and grouping these with intersections to form states. Our HMM defines allowed state transitions based on state proximity in the road network, with transition probabilities based on distance between states and expected segment speeds. An observation probability, given a state, depends on the distance between GPS-reported position and a representative point on the subsegment or intersection. All probabilities are assumed to be normally distributed, with observation variance based on the reported error in the current GPS reading. Our HMM also encodes, at least approximately, the logic of traffic. For instance, assume that the transition between subsegment states A and B is feasible, but intersection state C is between them with a traffic light; then the $A \rightarrow B$ transition can only occur when the light-status at C is green.

The states, state transitions and associated probability distributions, observation probability distributions, and logic of traffic comprise our hidden Markov model. Using an HMM forward pass on this model, we can efficiently compute the observation probability at each time, given the model and preceding observations, as well as the probability for the entire sequence of observations versus the model. An HMM backward pass can then be performed to find the sequence of most probable states, given the observations and the model. The observation probabilities are the scores mentioned earlier, the most probable states are the improved associations, and we can annotate the GPS stream based on the most probable state for each reading and relevant traffic controls.

One might wonder why intersections are included in our set of HMM states, since subsegments would suffice to track a car's progress along a route. One can think of an intersection as a dwell state, that is, a state that can transit back to itself with high probability, and where the ability to transit away from this state might depend on observable conditions, such as the status of the corresponding traffic light.

While the per-observation HMM scores have value in identifying points where the car behaves in an unexpected way, the overall HMM score for a sequence of observations also has value. For example, we have had difficulty evaluating the offset between GPS timestamps from client phones and timestamps associated with changes in light status. (Phone timestamps are based on cell tower time, which can be inaccurate, and light status information is timestamped at our server a few seconds after a change in status has occurred.) The most effective method for identifying the drift between these multiple timestamps is often to to try several different offsets and compute an overall HMM score for each. The best offset for the entire route is the one returning the highest HMM score. This sort of accuracy is essential in any post-route analysis of our users' actions, since the analysis is impacted so profoundly by the states of traffic lights that change multiple times each minute.

Post-Processing and Aggregation: Finally, there is a post-processing step that uses output derived from the HMM to improve our estimates for the position and speed of the car at each point in time as it travels the route. These improved estimates are then aggregated across routes and used to estimate segment speeds and delays at intersections for future processing of route requests. In the future, we hope to improve these estimates further by clustering routes by features such as time-of-day, proximity to congested regions, and so on.

5 Conclusions and Future Work

Our technical focus in this paper has been on two specific AI-related problems that arise in the development of Green Driver: routing quickly in the presence of stochastic segment costs, and our use of hidden Markov models to analyze the erratic data returned by smartphone GPS's.

There is much to be done here; the techniques we use work well enough as opposed to well in any absolute sense. We know that our implementation of A* produces routes that are suboptimal in at least some cases, but know of no computationally viable alternatives. Our use of hidden Markov models has significantly improved our analysis of the route information provided by our users, but presumably there is much to be done there as well.

Other AI-related problems will also bear on Green Driver. Suppose that we have a great deal of historical data for a particular segment that we know slows down considerably during rush hour, but that we have only sparse data for an adjacent segment (but not adjoining – i.e., 12th Street as opposed to 11th).

What can we conclude from this? Should we argue that the two segments are fundamentally alike, and therefore likely to exhibit similar historical patterns? Should we conclude that the fact we have little data for the second segment is itself meaningful, suggesting that drivers are avoiding this route for a reason? Which segments can reasonably be viewed as "similar", and which cannot? There are many machine-learning and classification issues here, and we have only begun to consider them.

The situation becomes still more complex in the presence of real-time traffic information. Clearly a traffic jam five miles away is not an issue if it will have cleared by the time we get there, but how are we to predict that? Our initial investigations into these issues suggest that straightforward statistical methods may perform comparably to or better than more complex approaches based on modeling traffic as fluid flow. But here, too, much is to be done, and the work will become only more interesting as we collect and understand ever more data provided by our users.

Looking further to the future, the problems become more interesting still. At what point is it meaningful for us to try to load balance road usage by our users? At some point, as well, the lights should be responding to the drivers as much as the other way around; sensors are close to traffic lights only because there has historically been no other way to do it. As we get accurate information from our users regarding their positions and intended routes, there is no reason for us not to inform traffic lights minutes in advance that vehicles are coming. What sort of optimization algorithms are appropriate for such problems?

We look forward to tackling all of these problems in the future. And given how little time we now spend waiting at red lights, we should have plenty of time to do it.

References

Bander, J. L., and White, C. C. 2002. A heuristic search approach for a nonstationary stochastic shortest path problem with terminal cost. *Transportation Science* 36(2):218–230.

Bertsekas, D., and Tsitsiklis, J. 1991. An analysis of stochastic shortest path problem. *Mathematics of Op. Rsch.* 16(3):580–595.

Boyan, J., and Mitzenmacher, M. 2001. Improved results for route planning in stochastic transportation networks. In *In Proc. of Symposium of Discrete Algorithms*.

Cormen, T. H.; E.Leiserson, C.; Rivest, R. L.; and Stein, C. 2001. Dynamic programming. In *Introduction to Algorithms*. MIT Press. chapter 15, 323–369.

Hall, R. W. 1986. The fastest path through a network with random time-dependent travel times. *Transportation Science* 20(3):182–188.

Koonce, P.; Kittelson, M. W.; Rodegerdts, M. L.; and Others. 2008. Traffic signal timing manual. *Technical Report, Federal Highway Administration, U.S. Department of Transportation*.

Rabiner, L. R. 1989. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, 257–286.

Wellman, M. P.; Larson, K.; Ford, M.; and Wurman, P. R. 1995. Path planning under time-dependent uncertainty. In *In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, 532–539. Morgan Kaufmann.