EasyTracker: Automatic Transit Tracking, Mapping, and Arrival Time Prediction Using Smartphones

James Biagioni jbiagi1@uic.edu
Tomas Gerlich tgerli2@uic.edu
Timothy Merrifield tmerri4@uic.edu
Jakob Eriksson* jakob@uic.edu

Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607, USA

Abstract

In order to facilitate the introduction of transit tracking and arrival time prediction in smaller transit agencies, we investigate an automatic, smartphone-based system which we call EasyTracker. To use EasyTracker, a transit agency must obtain smartphones, install an app, and place a phone in each transit vehicle. Our goal is to require no other input.

This level of automation is possible through a set of algorithms that use GPS traces collected from instrumented transit vehicles to determine routes served, locate stops, and infer schedules. In addition, online algorithms automatically determine the route served by a given vehicle at a given time and predict its arrival time at upcoming stops.

We evaluate our algorithms on real datasets from two existing transit services. We demonstrate our ability to accurately reconstruct routes and schedules, and compare our system's arrival time prediction performance with the current "state of the art" for smaller transit operators: the official schedule. Finally, we discuss our current prototype implementation and the steps required to take it from a research prototype to a real system.

Categories and Subject Descriptors

H.4.2 [**Information Systems Applications**]: Types of Systems—*Logistics*

General Terms

Algorithms, Design, Experimentation, Performance

*This material is based upon work supported by the National Science Foundation under Grants CNS-1017877 and DGE-0549489, as well as the Transportation Research Board of the National Academies contract Transit IDEA #T-65.

Keywords

Public Transit, Transportation, Bus, Real-Time Tracking, Smartphone, GPS Trace Processing

1 Introduction

Transit information has come a long way from the printed paper schedules of decades past. Today, virtually every transit service in the developed world, regardless of size, makes their schedule available on the web in one form or another. However, for smaller operations, this is often where it ends. More advanced services, such as integration with Google maps, automatic transit directions, or real-time tracking and arrival time prediction, are typically reserved for large transit agencies who have the necessary expertise in-house or can afford to have it done for them.

Through these advanced services, transit agencies can dramatically improve the transit user experience. Real-time tracking and arrival time prediction are particularly powerful: transit riders who would otherwise be enduring a potentially frustrating wait can now adjust their travel plans to minimize waiting time. Even where buses typically run on time, real-time tracking can improve rider confidence in the transit service, allowing users to schedule closer connections with less built-in margin of error.

For these reasons, real-time transit tracking is a highly desired feature among transit riders. That said, the implementation of a full transit tracking system, including complete and accurate digital route maps, in-vehicle tracking devices, an online tracking website, up-to-date schedules, and accurate arrival time predictions, can be a daunting and costly process for smaller transit operators. Commercial providers exist, with NextBus [5] and Clever Devices [1] being the two main vendors. However, use of these services incurs substantial initial and recurring fees.[1]

In this paper, our goal is to reduce the cost and complexity of offering these services by creating EasyTracker, an automatic system for transit tracking, mapping, and arrival time prediction. The system consists of four main components: (a) an off-the-shelf smartphone, installed in each bus or carried by each driver, functioning as an automatic vehicle lo-

[1]The Chicago Transit Authority budgeted $24M to install bus tracking in 1000–1500 vehicles.

cation system or tracking device, (b) batch processing on a back-end server which turns stored vehicle trajectories into route maps, schedules, and prediction parameters, (c) online processing on a back-end server which uses the real-time location of a vehicle to produce arrival time predictions, and (d) a user interface that allows a user to access current vehicle locations and predicted arrival times.

Using EasyTracker, a transit agency can implement a sophisticated bus-tracking and arrival time prediction system by simply purchasing a number of smartphones and downloading the bus-tracking app to each phone. EasyTracker has considerable advantages over the current state of the art. First, the use of standard smartphone hardware reduces both the one-time and recurring costs involved in establishing a real-time transit tracking system. Second, since the system automates the process of route map and schedule creation, cost and required user input are dramatically reduced. Third, due to its automated nature, our system is able to adjust the published routes and schedules in response to road construction or predictable congestion events.

The scientific contributions of this paper are as follows:

- An algorithm for deriving the set of serviced routes from a collection of unlabeled GPS traces, requiring no driver interaction or other user input.
- An algorithm for determining the locations of transit stops along these routes.
- A means of automatically producing an estimate of the route schedule, describing hours of operation and intended arrival times for arrival time prediction.
- A thorough evaluation of our system using five months of data across seven routes from two transit agencies.

Below follows additional background and motivation (§2), followed by a high-level overview of our proposed system (§3). We then describe and evaluate each technical contribution separately (§4-6). In §7, we evaluate the arrival time prediction performance of our system, and §8 provides a discussion of the limitations of EasyTracker, and how we propose to address these in a production system. Finally, we provide a brief overview of the related work (§9) and offer our conclusions (§10).

2 Background and Motivation

The minimum requirement for a real-time transit tracking system is an in-vehicle device (sometimes referred to as an Automatic Vehicle Locator, or AVL unit) and a back-office component. The in-vehicle device determines the vehicle's current location using the Global Positioning System (GPS) and communicates this via a wireless link (typically cellular service) to the back-office. The back-office component is a central server that processes the incoming time-ordered sequences of locations (location traces) and typically provides a live tracking website for the public as well as status monitoring for dispatch purposes.

This type of vehicle tracking, which simply reports the locations of all active vehicles, is widely available today. While this is a useful service, its utility for transit applications is somewhat diminished by a lack of sufficient navigation metadata: what route is each bus driving, and at what time will it arrive at my stop? State of the art systems provide this metadata by means of an in-vehicle device which accepts driver input, such as the current route, as well as by estimating arrival times based on current vehicle location, past travel times, and the official route schedule.

In order to make arrival time predictions, these navigation-enabled systems require the following additional information:

1. A route shape file containing the road segments traversed by each route, for matching a vehicle's current GPS location to a location along the route.
2. A list of stops for each route in traversal order, for producing trip directions.
3. The planned schedule for each route and stop, to handle corner-cases such as the first and last trip of the day.
4. The route driven by each active vehicle at all times, to know where each vehicle is going next.

Manually collecting this information can be a time-consuming and complex task for many transit agencies. The authors have personal experience from working with four different transit agencies, which serve between 1,000 and 500,000 trips per day. Anecdotally, one such agency, despite an annual budget of $250M, lacks the resources to produce route shape files for their existing bus routes. As a consequence, their routes do not appear in Google Maps [4] and other trip planning services.

2.1 Back-end Processing

What primarily sets EasyTracker apart from the current state of the art is the aim to require no manual input. Using EasyTracker, items 1-3 are automatically derived from recorded GPS traces. This not only reduces the amount of manual labor required, but also enables agencies that lack the necessary technical expertise in-house to set up a transit tracking system without seeking external assistance. Item 4, the current route of a vehicle, is automatically determined based on its recent movements and the known route map (item 1). This automatic classification allows drivers to focus on safety, and avoids the need for additional driver training.

2.2 In-vehicle Device

For the purpose of this paper, the type of in-vehicle device used is of relatively lesser significance: the quality of GPS coordinates provided by a smartphone GPS unit are not dramatically different from those provided by a dedicated vehicle tracking device. However, the initial and recurring costs can differ dramatically, given the benefits of mass-production of smartphones. In urban environments, where vehicles often travel in the GPS shadow of tall buildings, a sophisticated vehicle tracking device may use inertial navigation to augment the outage-prone GPS sensor. While inertial navigation itself is outside the scope of this paper, modern smartphones are equipped with several suitable sensors (accelerometer, gyroscope, electronic compass, cellular and WiFi radio) which may be leveraged to improve GPS accuracy in challenging environments. Any technique that improves GPS localization will only improve the performance of the system.

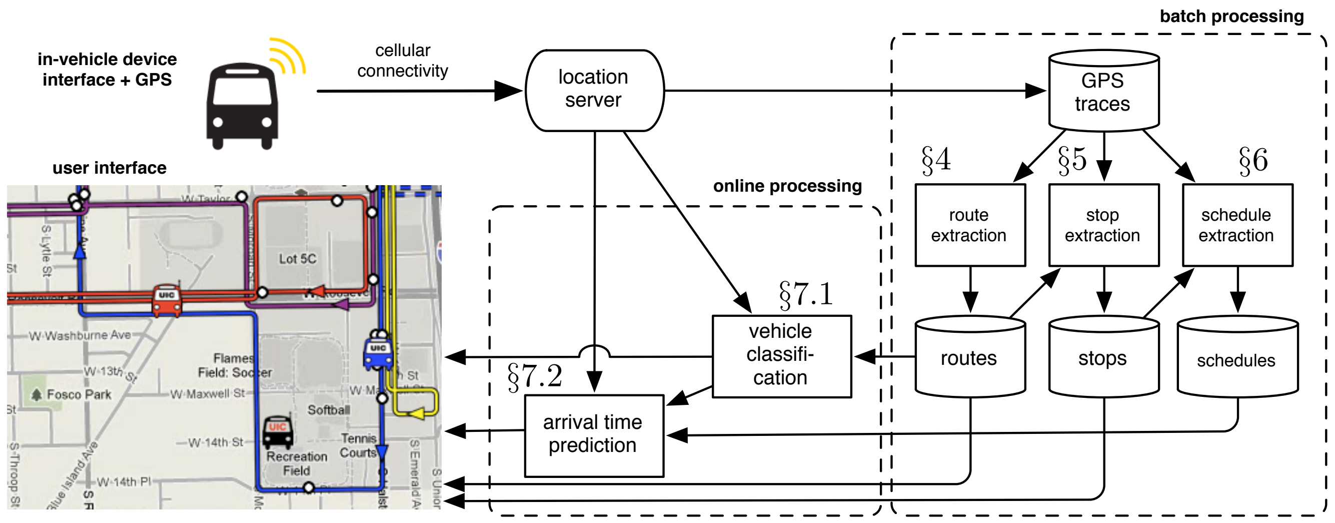


Figure 1. Architectural overview of the EasyTracker system. Data produced by in-vehicle devices are passed through batch and online processing, yielding route shapes, stop locations, vehicle classifications, and arrival time predictions, which are displayed through a user interface. Relevant paper sections indicated.

3 System Architecture and Overview

We call our system EasyTracker, for the ease with which it allows transit agencies to add transit tracking to their list of services. As illustrated in Figure 1, the EasyTracker architecture consists of a data collection unit in each vehicle, a number of algorithms for online and batch data processing, and one or more user interfaces for the transit user. The primary function of the in-vehicle device is to periodically transmit its GPS coordinates to a central location server, using a cellular data link. The in-vehicle device can either be permanently installed in the vehicle, or may be carried by the driver. A central location server receives location updates from all in-vehicle devices.

The received records are put through batch- and online processing. In batch processing, a large set of recorded GPS traces are processed to produce route shapes, stop locations, and schedules. Online processing matches vehicles to routes, and performs arrival time prediction.

3.1 Batch Processing

Our **route extraction** algorithm (§4) is based purely on GPS traces, and does not make use of an existing road map. This design is based on three observations:

- A sufficiently accurate digital road map may not be freely available for the area of interest.
- Since transit routes do not necessarily follow public roadways, a road map may be misleading.
- Our route extraction algorithm performs more reliably when using an automatically generated map.

Given the route shapes, the recorded traces are separated based on the shape they follow. Each set is fed to our **stop extraction** (§5) algorithm, which produces a set of bus stops based on the vehicles' movements along the shape.

Given the stop locations, we determine the raw arrivals: the exact time that each trace arrived at (or passed by) each stop along the route. These are processed by our **schedule estimation** (§6) algorithm, to estimate the planned schedule of the route.

3.2 Online Processing

Automatic **route classification** (§7.1) classifies vehicles as "in-service", serving a particular route as determined by the route shapes, or "out-of-service" if the recent location trace does not match up with a known route.

Once a vehicle is deemed "in-service", **arrival time predictions** (§7.2) are made using the recent location trace of the vehicle and the relevant route schedule. Batch processing is described in more detail below.

3.3 Prototype User Interface

Shown in the bottom-left corner of Figure 1 is a cropped image from a prototype system built by the authors, currently in use by their campus shuttle service. Vehicles are matched to routes by their color, and clicking on a shuttle stop or vehicle brings up the arrival time predictions. Given route shapes, stop locations and predicted arrival times, a number of variations on this interface are easily constructed.

3.4 GPS Traces and Ground Truth Data

The majority of this evaluation is based on recorded GPS traces from our campus shuttle service, as well as captured data from the publicly available Chicago Transit Authority (CTA) real-time bus tracker feed [2].

For the campus shuttle, we use one month of GPS traces recorded from our fleet of thirteen campus shuttle buses, operating four routes (six including minor variations). Campus shuttle GPS traces have the following characteristics:

- They are labeled only with a vehicle ID number—these numbers do not correspond to routes, as every vehicle can be serving any route at any given time.

- Locations are recorded any time the ignition is on—the shuttles frequently take trips to locations off the official routes, for mechanical service, outreach operations, or chartered University outings.

We also manually collected the ground truth location of each campus shuttle stop, and the exact route followed by the campus shuttles to serve as ground truth for route extraction.

For the CTA data, we use 100 days of traces from a single route. The CTA bus tracker system provides bus locations only once every sixty seconds—we interpolate these traces down to one second intervals, to allow for uniform processing. For the CTA, their official General Transit Feed Specification (GTFS) [3] feed, which serves as our ground truth, specifies both stop locations and route shapes.

4 Route Extraction

Route extraction is the process of turning unlabeled GPS traces into the set of service route shapes followed by instrumented transit vehicles. These route shapes can then be used to label real-time GPS traces and classify transit vehicles as belonging to a particular route. Additionally, route-labeled traces are used to perform stop extraction (§5), and the route shapes themselves may be used in drawing a user interface.

Since many transit vehicles travel on public roads, it may seem natural to base a route extraction algorithm on an existing road map. While this is a reasonable initial approach, it comes with several drawbacks:

- A completely accurate road map may not be freely available for the service area. While a free map such as [6] may visually appear accurate, errors in turn restrictions and connectivity are (anecdotally) common. This can result in significant errors in the routes produced.
- Because transit vehicles may use limited-access service roads, or exclusive right-of-way transit lanes, we cannot necessarily rely on existing digital road maps to supply us with the unique road features that may be used by our transit vehicles.
- A complete road map contains many roads not typically traveled by the transit vehicles. As we describe below, this makes a route extraction algorithm based on a full road map susceptible to GPS noise, which may introduce spurious map edges into the extracted routes.

As an alternative, we also evaluate the use of a map inference algorithm [15] to generate our own model of the road network from the GPS traces produced by our vehicles. This approach allows new road segments to be added on-demand: as soon as GPS trace data is available from transit vehicles, the portion of the road network that is newly utilized may be added to the graph.

Figure 2 illustrates the complete automated route extraction process at a high level. Starting with an input of raw GPS traces (Figure 2(a)), a kernel density estimate of the full set of traces is computed (Figure 2(b)). A road map is extracted using [15], which is subsequently smoothed using the Douglas-Peucker algorithm [16] (Figure 2(c)).

The generated road map is used to map-match the raw GPS traces, turning each trace into a series of discrete road segments. These road segment series are then analyzed to find frequently repeated sequences, which are output as the set of official transit routes. Figure 2(d) illustrates one of several extracted routes. Below, we describe the route extraction process in more detail.

4.1 Raw Data Pre-Processing

The first step in our process is to clean up and organize the raw data. Each GPS location is accompanied by a MAC address (identifying the vehicle) and a timestamp. We call a sequence of time-ordered GPS locations with the same MAC address a *trace*. Each trace is broken up into several *drives*, separated by long (10 minute) intervals without location reports. Such intervals typically indicate a parked vehicle, making them a natural delimiter.

Depending on the frequency with which the in-vehicle device is configured to record location data, we may collect a large amount of location points that are very close to each other, when a vehicle is stopped or moving slowly. For the purpose of route extraction we prefer a sparse representation of the traveled path, as extra points along an edge afford us no advantage, and incurs additional computational overhead. Therefore, we thin the trace to produce a linear density of locations in each direction of one point for every 20 meters. This value was selected empirically, to balance between sufficient data density and reasonable runtime.

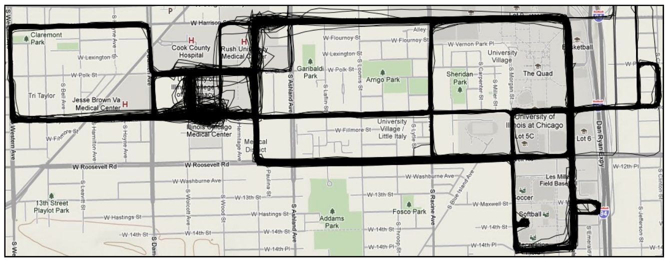
4.2 Map Generation

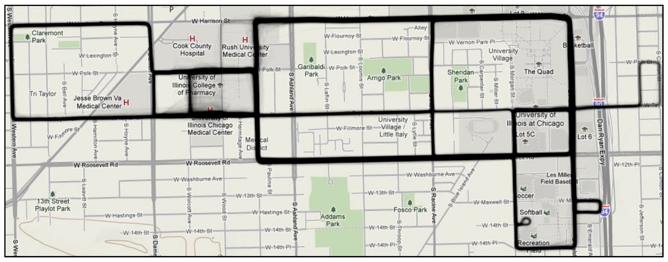
Here, we briefly summarize the map generation scheme proposed by Davies, Beresford, and Hopper [15]. The literature on map generation from GPS traces describes at least eleven distinct algorithms for map generation. We have implemented and evaluated three representative algorithms [18, 15, 12], and found [15] to produce the best results for our route extraction purposes.

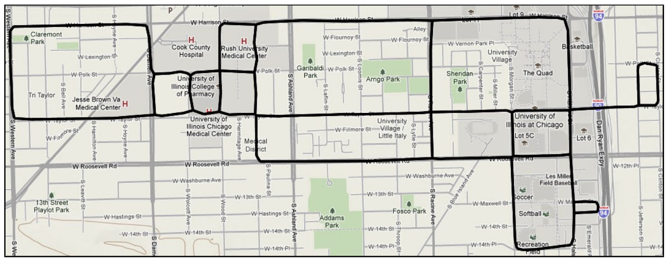
The process starts by computing an approximate, grid-based kernel density estimate [27] of the traces provided, over the area of interest. For each edge between consecutive GPS points, cells incident on the edge are incremented, creating a two-dimensional histogram. An anti-aliasing [22] method is used to account for trace edges straddling grid cells, followed by a gaussian smoothing filter [21] to produce the desired density estimate.

This estimate is passed through a threshold function, to produce a binary image of the underlying roads. A contour follower [36] is used to extract the road outlines, and the road centerline is found by computing the Voronoi graph [9] of points evenly spaced along these outlines, followed by the removal of edges that fall outside the contour or that are of insufficient length.

This produces a very jagged road map. We address this by smoothing each road segment, defined as a series of edges with no intersections, using the Douglas-Peucker algorithm [16]. In another departure from [15], we explicitly make each edge bi-directional, to improve the robustness of the algorithm. While this does not necessarily produce a correct road map (some roads are one-way), it is of no consequence to the route extraction algorithm described below.


(a) Raw GPS traces drawn as a separate thin black lines.


(b) Kernel density estimate as a gray-scale overlay.


(c) Road map extracted from all traces, after smoothing.

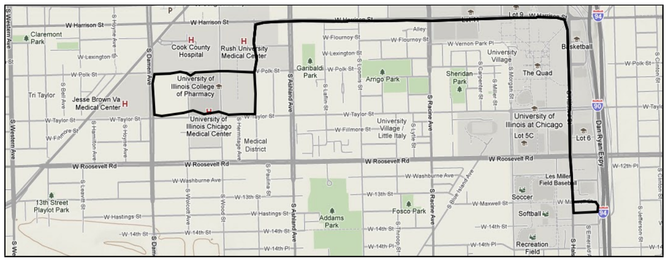

(d) Single extracted route, out of several.

Figure 2. High-level overview of the route extraction process, overlaid on a screenshot of the local road map.

4.3 Route Extraction

To identify the routes followed by our transit vehicles, we first map-match our original drives onto the edges of our extracted road map. This is done using the Viterbi algorithm, as described in [32, 24]. The output of the Viterbi algorithm provides us with the maximum-probability sequence of edges traversed. This sequence of edges is processed further to identify our set of routes.

We iterate through the edge-sequence sequentially until a repeated series of edges of a minimum sum length (we use a distance of 100 meters, or half a block) is encountered, taking direction of traversal into account. Note that we detect a series of edges with a minimum sum length, rather than a single repeated edge. This helps avoid problems where repeated traversals of the same intersection or circulation point may otherwise trigger the repetition detector.

The detected repetition conceptually completes a circuit in our graph, and the resulting subset of edges is considered a *route candidate*. The route candidate is stored, and processing continues from the first repeated edge onwards through the rest of the edge-sequence data. Once this process is complete, we have produced a collection of edge sequences representing all of our route candidates. Note that we assume that each route is cyclical, i.e. it eventually repeats. If a transit system were to contain non-cyclical routes, a different heuristic would be required for detecting and separating route candidates.

In order to identify the true routes from among all of the possible candidates, we count the number of instances of each route candidate. Figure 3 shows the traversal count for each identified route from one month of shuttle data. Here, the black bars show the counts when using an existing map [6], and the gray bars represent the counts when using a map generated as described above. Starting with the route with the highest count, we incrementally add routes to the set of

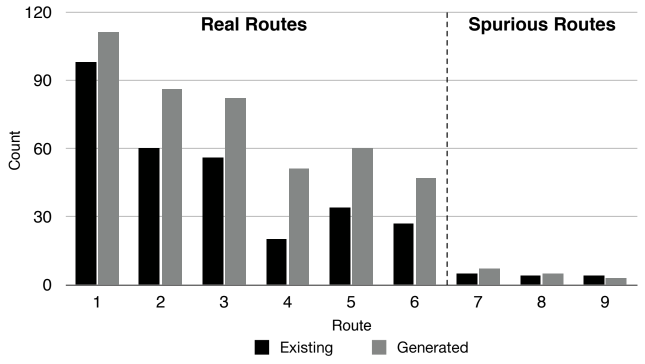


Figure 3. A collection of real and spurious routes, and their corresponding count values (based on existing and generated road maps) using one month of data from the UIC campus shuttles.

official routes. For each added route, we use Welch's t-test [34], to ensure that the new set of route counts is statistically different from the remaining route candidates. Once the t-test fails, the last added route is removed from the set, completing the route extraction process.

In Figure 3, the six routes on the left with the highest counts were manually verified to coincide with the actual campus shuttle routes, whereas the three routes on the right are the three spurious routes with the highest counts, with many more left out. Spurious routes either represent noise in the underlying trips, or GPS noise. Trip noise may be one-off charter trips, and trips to and from the bus depot that do not represent actual campus shuttle routes. GPS noise on the other hand can produce spurious edges during a normal trip, resulting in a unique route. When using the existing

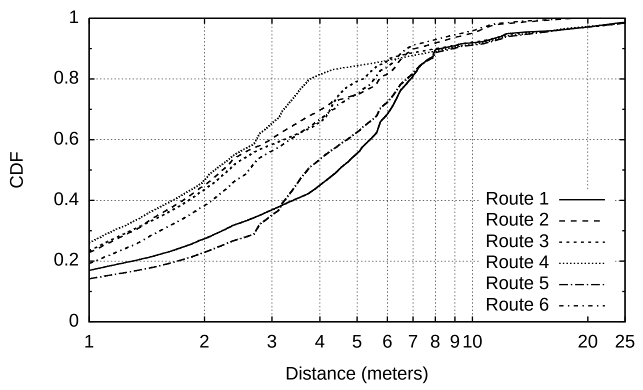


Figure 4. CDF of distance between generated route and ground truth, per route.

Route	Spurious	Mean error
1	1.4 %	5.0 m
2	0.0 %	3.3 m
3	1.6 %	3.9 m
4	1.5 %	3.6 m
5	1.6 %	5.1 m
6	0.0 %	3.4 m
All Routes	1.0 %	4.1 m

Table 1. Route extraction performance. Spurious parts of a route are $>$ 25 m from the ground truth.

map, high GPS error samples occasionally introduce spurious edges, resulting in unique, and spurious, routes. This explains the relatively lower counts on the real routes when using the existing map.

Using Welch's t-test, the routes identified using the generated map have a p-value of 0.0011, considered very statistically significant, whereas the routes identified using the existing map have a p-value of 0.0123, considered (merely) statistically significant. From this, we conclude that our route extraction algorithm performs significantly better when using a generated map.

4.4 Route Extraction Performance

To evaluate the accuracy of our route extraction algorithm we compare our set of extracted routes against the ground-truth routes provided to us by the campus shuttle team, which are based on the OpenStreetMap [6] dataset. The comparison is performed as follows: along the length of the extracted route, select locations at one-meter intervals. For each location, measure the distance to the closest ground truth edge.

Figure 4 shows the CDF of the resulting distances, for each of the six extracted routes. For all routes, the 90th percentile is below ten meters. The ground truth is based on a route map which describes a four-lane road (plus parking lanes) using a single centerline. Considering that the average lane width in the United States is approximately 3.35 meters (11 feet), an error below ten meters falls within the typical road. Hence, a difference of ten meters is not unreasonable for a bus route, which tends to stay on the right-hand side of the road. Some error in addition to this may also be expected, due to GPS noise.

However, for a small number of locations, we observe significantly larger errors. In Table 1, we refer to errors in excess of 25 meters as "spurious" parts of the route. We note that four out of six routes contain some spurious parts, all at or below 1.6% of the total route length. Through manual inspection, we found that these four routes include a street in the medical district of campus that is well characterized as a so-called urban canyon, with tall buildings on both sides interfering with GPS signals. In this area, the extracted roads are significantly offset from those in the ground truth, explaining the problem. Thus, the "spurious" parts are not strictly spurious, but actual parts of the ground-truth route with an unusually large displacement due to biased GPS error in the area. Based on the mean distance measured, we find that the extracted routes on average fall within two lane widths of the ground truth, the desired result. The largest error observed across all routes was 39.5 meters, or about one-fifth of a standard Chicago block.

5 Stop Extraction

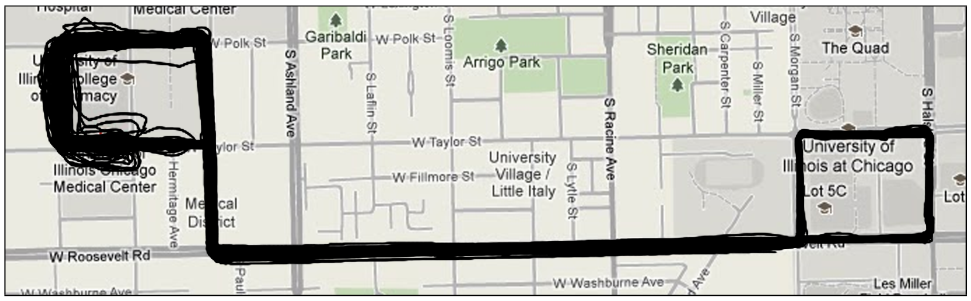
Stop extraction is the process of turning a set of raw GPS traces belonging to a given route, into a small set of coordinates indicating the locations of transit stops. The generated locations are used for producing arrival times for schedule extraction and arrival time prediction, and for drawing stop locations on a route map.

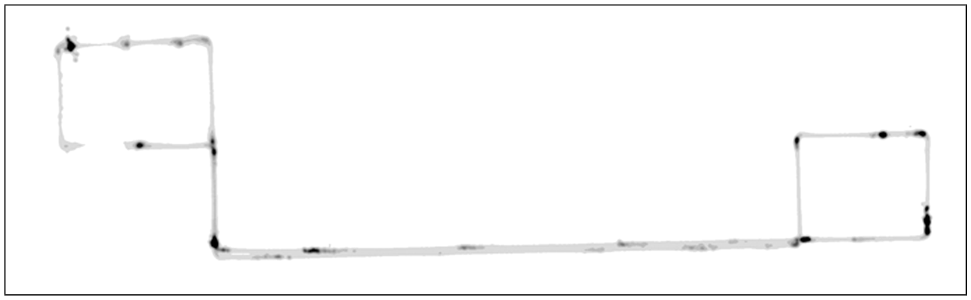
For schedule extraction and arrival time prediction, perfect accuracy of stop extraction is not required as long as most actual stops are represented. However, any omissions will result in the inability to predict arrival times for that stop. Furthermore, inaccuracies in stop locations when drawing the map can lead to missed buses and upset users. Therefore, our goal is to find as many of the actual stops as possible, while minimizing the number of spurious stops reported.

This is a challenging problem, as the movement pattern of a bus at a true bus stop is deceivingly similar to the behavior at traffic signals and stop signs. Moreover, error in GPS location introduces noise in the traces which complicates the identification of stopping events.

An example route with raw GPS traces drawn is depicted in Figure 5(a). Intuitively, buses spend more time at bus stops than in other locations. An estimate of the proportion of time spent in any location along the route is produced using kernel density estimation [27], using 1x1 meter cells. Figure 5(b) shows the computed density estimate along the example route, where the darker parts of the figure show places with a higher density of GPS points—these are the locations where the bus spent the most amount of time.

To extract a set of possible stop locations a minimum density threshold is applied, where any cell with a density at or above the 80th percentile of non-zero density cells is regarded as a potential stop. Using standard image processing techniques, the area of each connected group of cells is calculated, and groups below a minimum area are removed as a noise reduction step. Finally, the centroid of each remaining group is regarded as a detected stop. Figure 5(c) shows the detected stops along our example route as black disks. Disks marked by a dashed circle indicate spurious detections, whereas those without a circle are correct detections.


(a) Raw GPS traces from a single route. Raw traces reveal little about amount of time spent.


(b) Kernel density estimate of raw GPS points. Vehicles spend more time in the darker areas.

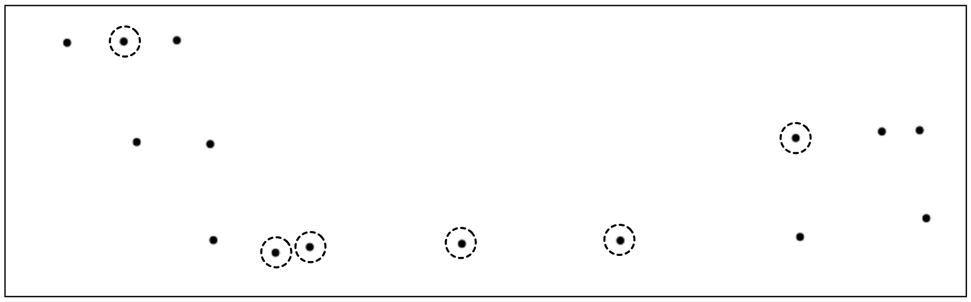

(c) Detected stop locations after applying threshold and computing centroid. Points surrounded by circles are spurious stop detections, points without circles are correctly detected stops.

Figure 5. Stop candidates are distilled from raw GPS points through density estimation, applying a minimum density threshold to identify stopping areas and computing the centroids of remaining connected components.

5.1 Stop Extraction Performance

To evaluate the performance of the stop extraction algorithm, data from six different campus shuttle bus routes over several weeks was used. We used one month of data for the six routes to train the threshold parameter to yield the best performance. Performance was not sensitive to small variations (±5%) in the threshold setting. The chosen cut-off of 80% aims to achieve maximal recall given acceptable precision. The performance of the algorithm was then evaluated on a separate dataset. Figure 6 illustrates the performance of the stop extraction algorithm in terms of precision and recall. Weekday, weekend, and an express route were all included, exhibiting different stopping behavior characteristics. The precision metric captures the fraction of detected stops that were true bus stops with respect to the total number of detected stops. On the other hand, the recall metric determines the fraction of correctly detected stops with respect to the total number of ground truth bus stops. As can be seen from

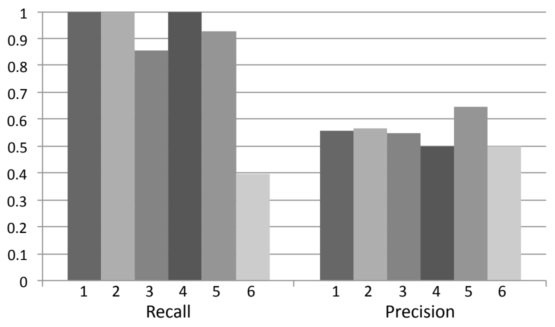


Figure 6. Precision and recall performance of the stop extraction algorithm on six routes, using an 80th percentile cut-off.

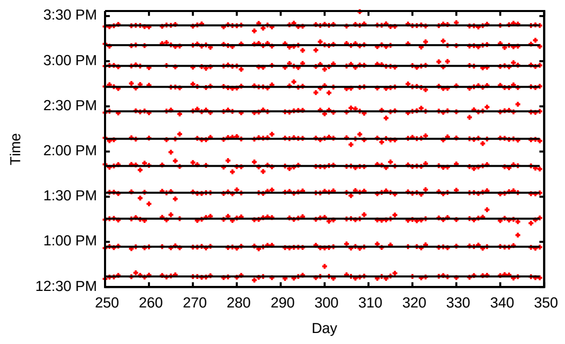


Figure 7. Arrival times for the first stop on a route. The horizontal lines indicate *k*-means computed clusters centers at this stop for each daily trip.

the graph, the stop extraction algorithm achieved between 85% and 100% recall, with around 50% precision across five of the six routes. The sixth route presented an unexpected challenge which we did not yet address: here, the bus was frequently parked for long periods of time in an area of high GPS error. The cells around this parking location dominated the density distribution for this route, making the 80th percentile a poor cut-off. Filtering out long-duration stops may be a good solution to this problem.

While a 100% recall metric is desirable, this was not possible to achieve on our routes, as some of the official bus stop locations are not typically used by any transit riders. Hence, the bus stops very rarely at these locations, leaving little evidence of it being a true bus stop. On the other hand, it is also difficult to achieve a perfect precision metric due to the strong similarity of bus stopping behavior at bus and non-bus stop locations. Intersections, traffic signals, traffic congestion, and stop signs can easily be confused with true bus stops. We investigated several methods based on both stop time and spatial distributions. Unfortunately, any performance improvements on training data came at the cost of over-training to specific cases. Thus, while our goal is to create a fully automatic system, and while the system does a great deal to whittle down the number of possible stop locations, it does to some extent fall short in the case of stop detection. In §8, we discuss means by which these weaknesses can be addressed through manual intervention.

6 Schedule Extraction

Schedule extraction is the process of turning raw arrival times at stops along a given route into a service schedule for each stop. While the final derived schedule may be displayed to end users, its primary purpose is to support the internal arrival time prediction system (see §7). The schedule provides a fall-back alternative for predictions far into the future, at the beginning or end of the day, or when vehicle tracking data is unavailable or unusable. Automating the schedule extraction process helps to reduce deployment overhead and may in some cases be helpful if the transit agency lacks a formal schedule. Additionally, using an extracted schedule from GPS trace data can correct inaccuracies in the pre-existing schedule or detect undocumented changes in behavior.

Figures 7–8 provide a graphical illustration of the challenges involved in deriving a static schedule from recorded arrival time data. In Figure 7, dots mark arrival times (y axis) at the first stop of Chicago Transit Authority route 157 in the interval 12:30-3:30pm over a span of 100 days (no service on weekends). Here, the underlying schedule is evident from the horizontal bands of arrivals at approximately the same time every day. The lines indicate the result of *k*-means clustering [20] these raw arrival times (discussed below), producing a high-quality schedule estimate. In contrast with the first stop, Figure 8 shows arrival times at the last stop. Here, while some banding is evident, a schedule is very difficult to discern, and clearly beyond straight-forward clustering.

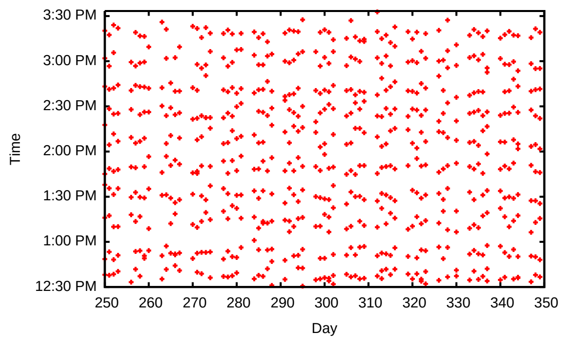


Figure 8. Arrival times at the last stop on a route. The data is too noisy to use the same (clustering) approach used in Figure 7.

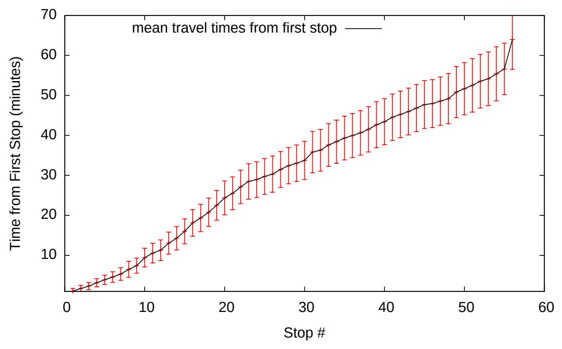


Figure 9. Per stop mean travel times from the first stop on a route, bars show standard deviation. Travel time variance increases with distance from the first stop.

Figure 9 further illustrates this difference in regularity: while travel-time uncertainty is present throughout the route, variance gradually builds as the bus travels away from the first stop, resulting in the disorganized arrival times shown in Figure 8. This invariably means that any schedule at the end of a route is going to be somewhat unreliable. For the route shown in Figure 9 the standard deviation at the last stop is roughly 10x greater than at the first stop.

Due to the high variance at later stops, we cannot rely on *k*-means clustering alone in order to produce a good schedule estimate for every stop along the route. Below, we define the problem formally, and present our solution. In short, we use *k*-means clustering on data from the first stop to determine its schedule, and compute downstream schedules from a combination of the first-stop schedule and the estimated travel time from the first stop at a given time of day.

6.1 Problem Description

The input to the schedule extraction algorithm is a set of stops $S = \{s_1, ..., s_{|S|}\}$ and a set of trips $\mathbf{T}_D = \{T_1^D, T_2^D, ..., T_{|T_D|}^D\}$ on day D along a route, where each trip T_i^D contains a series of arrival times, one per stop along the route $T_i^D = \{a_1^i, a_2^i, ..., a_{|S|}^i\}$. When considering only a single trip, we simplify our notation for arrival times to a_s where s is the stop.

The *schedule extraction problem* is given $\mathbf{T}$, produce a set of schedule times $K_s^{D'} = \{k^s{}_1, k^s{}_2, ..., k^s{}_{|T'_D|}\}$, where k_i^s is the i'th scheduled arrival time at stop s, and D' is either the day of the week, or one of the set $\{weekday, weekend\}$. For example, Figure 7 plots a_1^i (the arrival time points) for a range of i over all D, and k_i^1 (the horizontal lines) for the same range of i.

6.2 Estimating the First-Stop Schedule

As mentioned above, we produce only the first-stop schedule K_1 from the raw arrivals T_1^D. Schedules $K_2...K_{|S|}$ are derived from K_1 and the mean trip times for the time of day, as described in the next subsection.

To extract the intended first stop arrival times, we use k-means clustering over all arrival times a_1^i in all trips T_i. We choose our value for k to be $median(|T^D|)$, the median number of trips observed on this route per day. The choice of initial values has a significant effect on k-means performance. After experimenting with several initialization methods from [30], we settled on the max-min approach described in [23]. This approach initializes k seeds incrementally, choosing the next seed from a_s^i which is furthest away from the current collection of seeds. After we compute the k-means clusters, we have an accurate estimate of K_1, as shown in Figure 7.

6.3 Deriving $K_{2..|S|}$ from K_1 and Mean Times

For two stops i, j and trip t, we define

$$travel\_time(i,j,t) = \frac{1}{|D|}\sum_D a_j^t - a_i^t, \qquad (1)$$

the mean arrival time difference for trip t between stops i, j over all days in D. Given the schedule for the first stop, K_1, we can then compute the schedule for a later stop s as $K_s = \{k_1^s ... k_{|T^{D'}|}^s\}$ where

$$k_i^s = k_i^1 + travel\_time(1, s, i) \qquad (2)$$

As illustrated in Figure 9, the travel time variance increases as the bus travels further down the route, meaning the schedule will be increasingly inaccurate. Unfortunately, this is the nature of bus travel during traffic congestion—our goal is simply to produce the best schedule estimate.

Figure 10 illustrates an example output from the solution described above. Here, the black lines represent the schedule estimate computed from the first-stop departure schedule and the mean travel time from the first stop. The resulting schedule shows reasonable arrival time intervals and a good fit with the raw data.

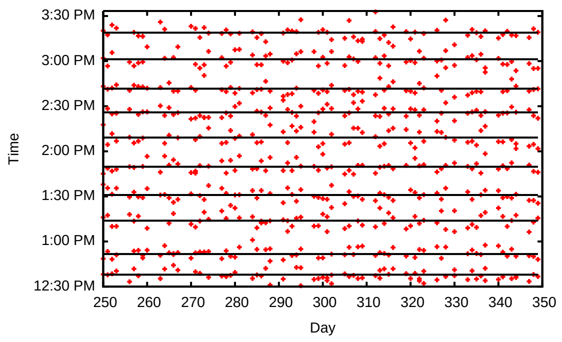


Figure 10. The schedule (horizontal lines) for the last stop based on the first stop schedule and mean travel time.

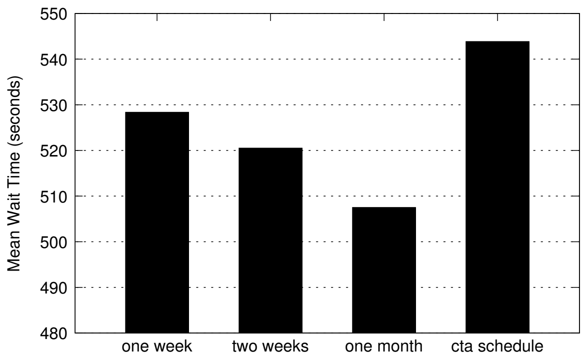


Figure 11. Mean wait times for a selected bus route for several training set sizes and the official CTA schedule.

6.4 Extracted Schedule Accuracy

We evaluate the extracted schedule using actual CTA bus arrival time traces. For this evaluation, we repeatedly choose a random bus stop and time, and use our extracted schedule for that stop to predict when the next bus is due to arrive. To simulate the experience of a typical bus rider, we "arrive" at the bus stop 2 minutes before the scheduled arrival time (building in a small margin of error—2 minutes was found to be the best value), and then wait until the next bus actually arrives, based on our recorded bus arrival time traces.

We refer to this as the *wait time*—if a passenger arrived at the stop based on the schedule, this is how long they would actually have been waiting for the bus to arrive. To see how the size of our training set impacts performance, we evaluated schedules generated from one week, two weeks and one month of data. For a given route, we performed this test 5,000 times and recorded the mean wait time for each schedule. As shown in Figure 11, the extracted schedule actually outperforms the official CTA schedule, potentially saving travelers an average of over 30 seconds. However, note the absolute values on the y axis: even our most accurate schedule, based on one month's worth of data, does a terrible job at accurately predicting arrival times. Using the schedule for guidance, the mean time spent waiting for a bus is over 8 minutes.

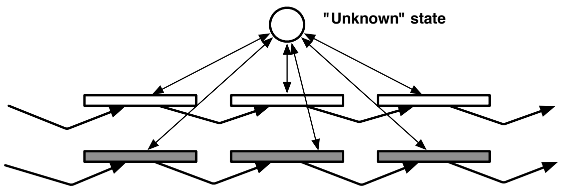


Figure 12. Route-matching Hidden Markov Model. Transitions between routes are only possible through the unknown state.

Given the inadequacy of static schedules in predicting bus arrival times, real-time transit tracking and arrival time prediction is clearly called for. In §7.2, we evaluate the accuracy of a simple real-time arrival time predictor, and compare this to using the static schedule.

7 Online Processing: Route Classification and Arrival Time prediction

The online processing components combine the routes, stops and schedules produced in batch processing with the recent GPS trace of a vehicle, to (a) determine if the vehicle is in service (and if so, on which route) and (b) estimate the arrival time of the vehicle at subsequent stops. Below, we describe how this is done.

7.1 Route Classification

Given a sequence of GPS points $G = \{g_1, ..., g_{t-1}, g_t\}$ recorded up until time t, and a set of candidate service routes R, our goal is to determine whether the vehicle is currently driving route $r \in R$. Assuming a vehicle serves at most one route in each uninterrupted drive, this can be determined by computing, for each route r,

$$dist = \frac{1}{|G|} \sum_{i=1..|G|} dist(g_i, r), \qquad (3)$$

where $dist(g_i, r)$ is the minimum distance between point g_i and any segment of route r. To reflect a more realistic usage model, we need to relax these assumptions as follows:

- Vehicles may serve multiple routes in a single drive.
- Vehicles may change between in-service and out-of-service within a single drive.
- Vehicles may occasionally detour around closed roads or accident sites.

Thus, rather than make a single decision for an entire drive, we need to determine the status of each vehicle in an online fashion, as new GPS points arrive. To do this, we make use of Hidden Markov Model (HMM) map matching [24, 32], with a map constructed from the known routes, as illustrated in Figure 12. Here, the segments of each route are added as *separate* states, creating overlapping road segments where routes coincide, with no direct transitions between routes. Transition and emission probabilities are left unmodified, except for the introduction of a single "unknown" state, representing out-of-service driving and detours, and serving as a place-holder for transition between routes. Transitions are possible from each state to the unknown state, and from the unknown state to every other state, though the probability

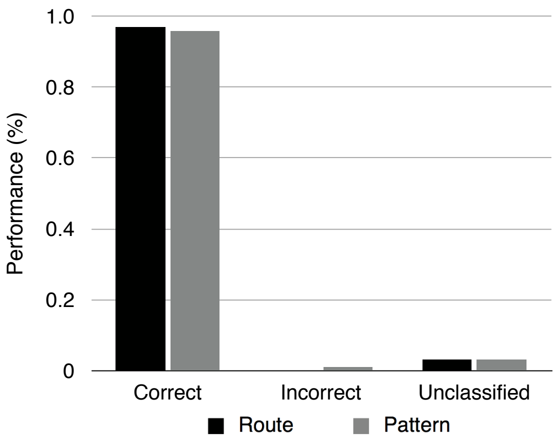


Figure 13. Classification performance of HMM-based vehicle classifier, using one month of labeled data from the UIC campus shuttles.

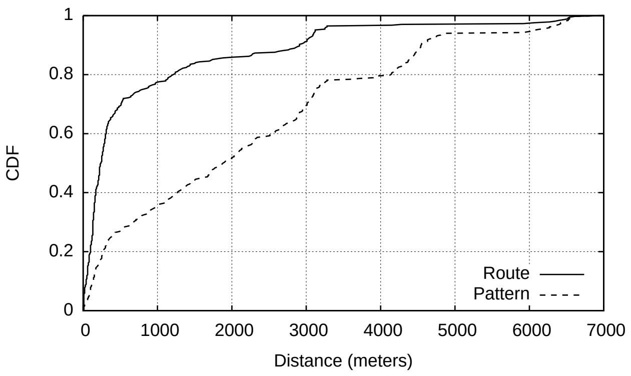


Figure 14. CDF of distance traveled before a correct pattern or route classification was made.

is a small non-zero constant. The emission probability of the unknown state is a small non-zero constant as well.

We use the Viterbi algorithm to decode the most probable sequence of states. As a consequence, once a vehicle is classified as belonging to a given route, it will remain in that state as long as it follows the route somewhat closely, independent of overlaps with other routes. Once it diverges significantly, it will either transition to the unknown state and stay there (if it is now out of service), or transition to the unknown state and then to another route (if it is now serving a different route).

We evaluate our HMM-based vehicle classifier on one month of labeled GPS trace data from the UIC campus shuttles. Specifically, we first split the labeled trace data into individual drives of each route. Then, for each drive we select a random starting point and run through the trace, measuring the distance traveled until a route classification is made. If we travel more than seven kilometers without making a route classification, we treat the route as "unclassified".

Out of our six routes, two are extended versions of two others. The existence of several different patterns belonging to a common route is a regular occurrence in transit networks. Typically, the variation served by a given vehicle depends on the time of the day, or the day of the week, which means they can be distinguished based on time, in combination with the spatial matching discussed here. To give an accurate picture of the route matching algorithm's performance in the face of such variations, we evaluate the accuracy of both unique pattern determination as well as aggregate route determination for these routes. To make the route or pattern classification decision, we analyze the Viterbi state probabilities with every new GPS point. When the most probable route/pattern is $10\times$ greater than the next most probable route/pattern, the algorithm makes its determination.

Figure 13 shows the overall performance of our HMM-based vehicle classifier for route and pattern determination. We see see that 97 and 96 percent of the time we are able to correctly classify the route and pattern, respectively. We can also see that 1 percent of patterns and 0 percent of routes are incorrectly classified, while 3 percent of patterns and routes remain unclassified after 7 kilometers of driving.

From this, we can see that the algorithm is, with high probability, eventually able to accurately determine the route and pattern driven. But how long does it take to make this determination? Figure 14 is a CDF of the distance traveled before a decision was made. As expected, routes are classified more quickly than patterns, as no pattern classification can be made without taking time into account, until the patterns for a given route diverge. In 75% of cases, 500 meters of travel is sufficient to distinguish the route traveled. For routes with patterns that largely overlap (as is the case here), or for routes that overlap substantially, the distance that needs to be traveled can be significantly longer. In conclusion, the performance of route classification depends heavily on the amount of overlap present in the transit network in question, as well as the typical driving patterns of transit vehicles. Once a decision is made, it will remain until the vehicle leaves the route. Hence, the initial classification delay is incurred only at the beginning of each shift.

7.2 Arrival Time Prediction

Arrival time prediction continuously estimates the next arrival time of a vehicle serving route r at stop s_i given a schedule estimate and (when available) the current location(s) of vehicles currently serving the route. Typical circumstances under which the current vehicle location may be unavailable or insufficient include:

- The first trip of the day, before any vehicle has started serving the route.
- The first several stops of the route, when the next vehicle has not yet departed the first stop.
- The last trip of the day. Here, a schedule is needed to predict that the vehicle will subsequently be taken out of service.

We assume that every vehicle serving the route is equipped with a working in-vehicle device, and is reporting its location periodically. If no vehicle is present on the route preceding s_i, we estimate the arrival time based on the next departure from the first stop s_1 according to the extracted schedule, plus $travel\_time(s_1, s_i, t)$, computed as described

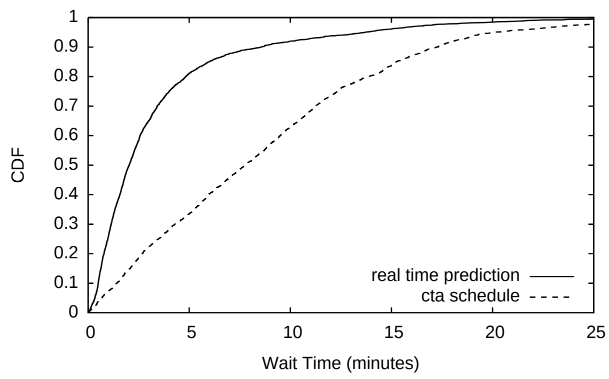


Figure 15. CDF of wait times for 5,000 real time arrival predictions vs. the CTA schedule.

in §6. Conversely, if a vehicle is present, then the time until the its arrival time at s_i is estimated based on the mean pairwise travel time between its current location and s_i, computed as

$$\begin{aligned}\gamma\, travel\_time(s_{prev+1}, s_i, t) + \\ (1-\gamma) travel\_time(s_{prev}, s_{prev+1}, t),\end{aligned} \tag{4}$$

where s_{prev} is the most recent stop served, γ is the fraction of the distance between s_{prev}, s_{prev+1} already traversed, and t is the current trip.

To evaluate the performance of our arrival time predictor, we perform the same experiment described for the extracted schedule. However, instead of consulting the schedule, we use the arrival time predictor, based on the location of the next bus that will arrive at the stop. Figure 15 clearly illustrates the benefit of incorporating real-time tracking when predicting bus arrival times. Wait times are significantly reduced by the use of real time data, bringing the median wait down from over 8 minutes to just 1 minute.

The reason behind this dramatic improvement can be seen in Figure 9. The route used for this evaluation has a mean service interval of approximately 15 minutes. Without real-time tracking, each vehicle spends the majority of its time in the latter parts of the route, where variance is high. With real-time tracking and a 15 minute service interval however, the bus is on average 7.5 minutes away from the stop in question. Assuming the travel time variance is loosely a function of travel time, it is easy to see from Figure 9 that the expected variance (at 7.5 minutes out) is very small. Hence, no matter where the bus is when we consult the predictor, we are likely to get an accurate prediction.

8 From Paper Product to Production System

In this section, we briefly discuss additional steps required to take the proposed system from its current form to production use. We also briefly mention additional features that may be incorporated to further improve the rider experience.

The system described thus far takes no manual input: routes, stop locations, schedules, vehicle classification and arrival time predictions are all based purely on unlabeled GPS traces. As we have shown, our system is able to (with the exception of stop locations and occasional slow vehicle classifications) produce results similar to or better than data entered by hand.

However, EasyTracker cannot produce the kind of transit tracking system users have come to expect without a bit of manual input. In addition to spurious stop locations, and the occasional slow route classification, the system lacks several kinds of annotations, such as stop and route labels, that transit riders would typically expect to see.

From a technical point of view, the stop extraction algorithm tends to produce spurious stop locations at a number of traffic lights and stop signs, and the vehicle classification algorithm is sometimes unable to distinguish between similar routes. These errors need to be corrected, either by additional sensor modalities, or by human intervention.

Below, we discuss two optional system components: an administrative web interface and a driver interface, that provide means of integrating a small amount of manual input into the system to significantly improve the user experience.

8.1 Optional Management Interface

We propose to complement the automatic system with an optional web-based management interface. The purpose of this interface is to allow a dispatcher (or other office personnel) to enter additional, static annotations to the system which cannot be automatically derived from GPS traces:

- Route labels, such as "Lakeshore Drive" or "Route 9A".
- Stop labels, such as "City Hall, or "Roosevelt/Halsted". Reasonable stop labels can be synthesized from a road map, but these may not correspond to the labels on stop signs or in paper schedules.
- Accessibility information, such as "Elevator available" or "Has bike rack".

Through the management interface, the transit operator is presented with the tools to correct mistakes, add route and stop labels, and other relevant annotations. In addition, the management interface may allow an operator that is aware of impending route or schedule changes to proactively "reset" selected routes, to avoid inertia in the acquisition of updated routes and schedules.

8.2 Optional Driver Interface

In addition to the static annotations that may be entered through the administrative web interface, drivers may optionally be trained to provide additional information through an interface on the in-vehicle device. Figure 16(b) illustrates an envisioned driver interface. Here, the driver may manually override the automatic route classification in case of a misclassification. Other data that the driver may be asked to provide includes passenger occupancy, availability of seats, room for wheelchairs/strollers, and bike rack occupancy.

To further improve the passenger experience, the smartphone may be connected to the vehicle speaker system to provide voice prompts to passengers, notifying them of the next stop. Finally, the smartphone interface may be used as a means for central dispatch to communicate with the driver in the form of text prompts when the vehicle is not moving.

8.3 Current Prototype System

Parts of EasyTracker are currently in use on the UIC campus shuttle system. Figure 16(a) is a screenshot of the web interface we provide to campus shuttle riders today. In the current prototype, vehicles are auto-classified as belonging to a red, blue, purple or yellow route using the algorithm in §7.1 and arrival times are predicted using the method in §7.2. Routes, stop locations and schedules from §4–6 were manually corrected to remove any mistakes, and the colored route map was drawn by hand.

9 Related Work

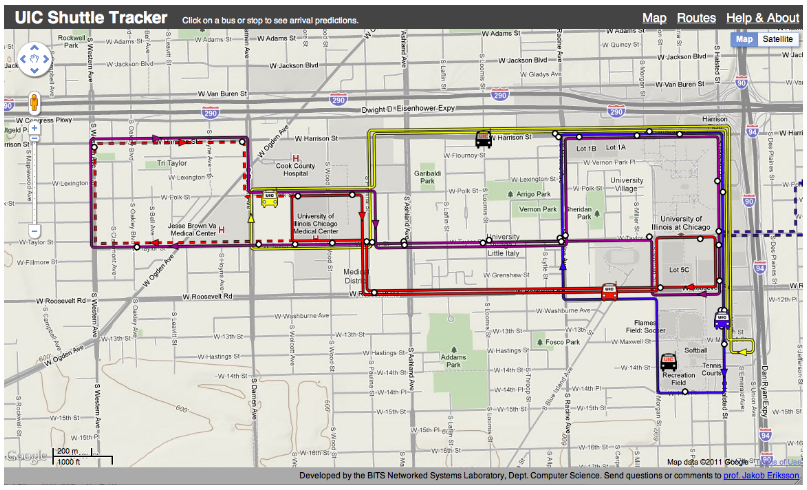
While transit tracking is already a popular service offered by commercial providers, EasyTracker is to our knowledge the first system that automates the entire process, from raw GPS traces to a complete transit tracking and arrival prediction system.

In [31, 10], a system for cooperative transit tracking is described. Here, it is assumed that the routes and schedules are known, but that the transit agency is not willing to install tracking devices. Instead, users cooperatively track transit vehicles through software that automatically reports their location when they are riding a bus or train. Extracting routes and stops through crowdsourcing, in addition to cooperative transit tracking, is an interesting direction, but outside the scope of this paper. TransitGenie [7], developed by the authors, is a transit navigation service for smartphones, which computes route recommendations based on real-time transit information, as opposed to static schedules. TransitGenie is complementary to the transit tracking system described here, in that it makes use of tracking information produced by services like EasyTracker to offer travel advice.

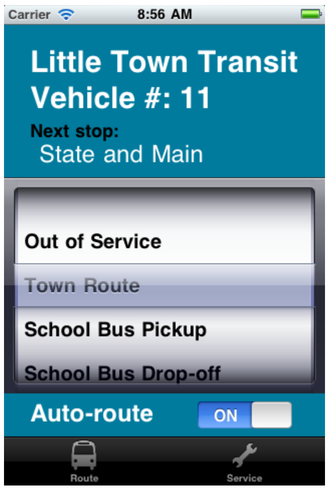
9.1 Map inference from GPS traces

A large body of work exists concerning the problem of inferring road maps from GPS traces, and the methods used can be broken into three general categories: (a) k-means [18, 26, 35, 8], which builds the road map using a series of cluster seeds connected using the raw trace data, (b) trace merging [25, 12], which incrementally merges together trace data that is similar in location and bearing into a road map, and (c) kernel density estimation [15, 13, 28], which first computes a kernel density estimate of the raw GPS data, and then uses image-processing techniques to extract the road map. In this paper, we build upon road maps extracted using [15] to automatically extract accurate transit route shapes.

Many of these and related papers take a more fine-grained approach to map inference than we used here. For our purposes a map with bi-directional edges for roads, and t-junctions for intersections was sufficient. However, many authors [18, 26, 19, 14] focus considerable attention on identifying individual lanes, and re-constructing complex intersection geometry. While simple transit route maps do not necessarily require the modeling of such road features, if there were a need for them, techniques do exist in the literature for extracting and generating such details.


UIC Shuttle Tracker
Click on a bus or stop to see arrival predictions.
Map
Routes
Help & About
Map
Satellite
Developed by the BITS Networked Systems Laboratory, Dept. Computer Science. Send questions or comments to prof. Jakob Eriksson

(a) Screen shot of current EasyTracker prototype, currently in production use on the UIC shuttle bus system.



(b) Envisioned driver interface for optional manual input and communication with dispatch.

Figure 16. User interface of our existing prototype system, and an envisioned smartphone driver interface.

9.2 Arrival Time Prediction

In 7.2 we present a very simple arrival time prediction system that relies on the schedule and mean trip times computed in 6.3. While this technique produces favorable results, more sophisticated bus arrival time prediction methods can be found in [11, 33] and [29].

9.3 Other Related Work

Classical non-parametric density estimation is explained in more detail in [17]. [19] considers the problem of detecting road intersections from GPS data where a shape descriptor is used to represent the distribution of GPS traces around a point. Related work on surface street traffic monitoring is described in [37].

10 Conclusion

We have presented EasyTracker, an automatic system for low-cost, real-time transit tracking, mapping and arrival time prediction. Based on our experience with building a campus shuttle tracking system for our University, we have found out (the hard way) how labor intensive the collection of this data can be. To address this problem, we have demonstrated how high-value data such as routes, stops, and transit schedules, can be computed automatically from simple GPS traces. Our system produces high-fidelity route maps, extracts transit stop locations, and constructs transit schedules that consistently out-perform the official schedules produced by the Chicago Transit Authority. Last but not least, EasyTracker provides accurate transit tracking and real-time arrival time predictions, all without manual intervention.

Acknowledgments

We would like to extend our gratitude to our diligent and enthusiastic shepherd Geoffrey Challen and the anonymous reviewers for their valuable and insightful guidance.

11 References

[1] Clever Devices Inc. `http://www.cleverdevices.com`.

[2] CTA Real-Time Feed. `http://www.ctabustracker.com`.

[3] General Transit Feed Specification (GTFS). `http://code.google.com/transit/spec/transit_feed_specification.html`.

[4] Google Maps. `http://maps.google.com`.

[5] NextBus To Roll-out System Wide In San Francisco. `http://www.nextbus.com/corporate/press/index.htm#muniNew`.

[6] OpenStreetMap, the free Wiki World Map. `http://www.openstreetmap.org`.

[7] TransitGenie Chicago. `http://www.transitgenie.com`.

[8] G. Agamennoni, J. Nieto, and E. Nebot. Robust inference of principal road paths for intelligent transportation systems. *Intelligent Transp. Systems, IEEE Transactions on*, 12(1):298–308, March 2011.

[9] F. Aurenhammer. Voronoi diagrams–a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405, Sept 1991.

[10] J. Biagioni, A. Agresta, T. Gerlich, and J. Eriksson. Transitgenie: A real-time, context-aware transit navigator (demo abstract). In *SenSys*, pages 329–330. ACM, 2009.

[11] Y. Bin, Y. Zhongzhen, and Y. Baozhen. Bus arrival time prediction using support vector machines. *Journal of Intelligent Transportation Systems*, 10(4):151–158, 2006.

[12] L. Cao and J. Krumm. From gps traces to a routable road map. In *GIS'09*, pages 3–12, New York, NY, USA, 2009. ACM.

[13] C. Chen and Y. Cheng. Roads digital map generation with multi-track gps data. In *Education Technology and Training, and International Workshop on Geoscience and Remote Sensing*, volume 1, pages 508–511, December 2008.

[14] Y. Chen and J. Krumm. Probabilistic modeling of traffic lanes from gps traces. In *GIS'10*, pages 81–88, New York, NY, USA, 2010. ACM.

[15] J. J. Davies, A. R. Beresford, and A. Hopper. Scalable, distributed, real-time map generation. *IEEE Pervasive Computing*, 5:47–54, 2006.

[16] D. Douglas and T. Peucker. Algorithms for the Reduction of the Number of Points Required to Represent a Line or Its Caricature. *The Canadian Cartographer*, (2), 1973.

[17] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley, 2001.

[18] S. Edelkamp and S. Schrödl. Route planning and map inference with global positioning traces. In R. Klein, H.-W. Six, and L. Wegner, editors, *Computer Science in Perspective*, volume 2598 of *Lecture Notes in Comp. Sci.*, pages 128–151. Springer Berlin / Heidelberg, 2003.

[19] A. Fathi and J. Krumm. Detecting road intersections from gps traces. In *GIScience'10*, pages 56–69, Berlin, Heidelberg, 2010. Springer-Verlag.

[20] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[21] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[22] W. J. Leler. Human vision, anti-aliasing, and the cheap 4000 line display. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '80, pages 308–313, New York, NY, USA, 1980. ACM.

[23] B. Mirkin. *Clustering for Data Mining: A Data Recovery Approach (Comp. Sci. and Data Analysis)*. Chapman & Hall/CRC, Apr. 2005.

[24] P. Newson and J. Krumm. Hidden markov map matching through noise and sparseness. In *GIS'09*, pages 336–343, New York, NY, USA, 2009. ACM.

[25] B. Niehoefer, R. Burda, C. Wietfeld, F. Bauer, and O. Lueert. Gps community map generation for enhanced routing methods based on trace-collection by mobile phones. In *Advances in Satellite and Space Communications, 2009. SPACOMM 2009. First International Conference on*, pages 156–161, July 2009.

[26] S. Schroedl, K. Wagstaff, S. Rogers, P. Langley, and C. Wilson. Mining gps traces for map refinement. *Data Mining and Knowledge Discovery*, 9:59–87, 2004.

[27] D. W. Scott. *Kernel Density Estimators*, pages 125–193. John Wiley and Sons, Inc., 2008.

[28] W. Shi, S. Shen, and Y. Liu. Automatic generation of road network map from massive gps, vehicle trajectories. In *Intelligent Transportation Systems, 2009. ITSC '09. 12th International IEEE Conference on*, pages 1–6, October 2009.

[29] Y. D. S.I.J Chien and C. Wei. Dynamic bus arrival time prediction with artificial neural networks. *Trans. Engrg.*, 128:429–438, 2002.

[30] D. Steinley and M. J. Brusco. Initializing k-means batch clustering: A critical evaluation of several techniques. *J. Classif.*, 24:99–121, June 2007.

[31] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson. Cooperative transit tracking using gps-enabled smartphones. In *SenSys*, pages 85–98. ACM, 2010.

[32] A. Thiagarajan, L. Sivalingam, K. LaCurts, S. Toledo, J. Eriksson, S. Madden, and H. Balakrishnan. Vtrack: Accurate, energy-aware road traffic delay estimation using mobile phones. In *SenSys*, 2009.

[33] D. Tiesyte and C. S. Jensen. Similarity-based prediction of travel times for vehicles traveling on known routes. In *GIS'08*, pages 14:1–14:10, New York, NY, USA, 2008. ACM.

[34] R. Walpole, R. Myers, and K. Ye. *Probability and Statistics for Engineers and Scientists*. Pearson Education, 2002.

[35] S. Worrall and E. Nebot. Automated process for generating digitised maps through gps data compression. In *Australasian Conference on Robotics and Automation*, 2007.

[36] S. Yokoi, J. ichiro Toriwaki, and T. Fukumura. An analysis of topological properties of digitized binary pictures using local features. *Computer Graphics and Image Processing*, 4(1):63 – 73, 1975.

[37] J. Yoon, B. Noble, and M. Liu. Surface street traffic estimation. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, MobiSys '07, pages 220–232, New York, NY, USA, 2007. ACM.