LiveTraVeL:
Real-time matching of transit vehicle trajectories to transit routes at scale

Georg Osang[1] James Cook[2] Alex Fabrikant[2] Marco Gruteser[2]

***Abstract*— We present LiveTraVeL (*Live Tra*nsit *Ve*hicle *L*abeling), a real-time system to label a stream of noisy observations of transit vehicle trajectories with the transit routes they are serving (e.g., northbound bus #5). In order to scale efficiently to large transit networks, our system first *retrieves* a small set of candidate routes from a geometrically indexed data structure, then applies a fine-grained *scoring* step to choose the best match. Given that real-time data remains unavailable for the majority of the world's transit agencies, these inferences can help feed a real-time map of a transit system's trips, infer transit trip delays in real time, or measure and correct noisy transit tracking data. This system can run on vehicle observations from a variety of sources that don't attach route information to vehicle observations, such as public imagery streams or user-contributed transit vehicle sightings.**

We abstract away the specifics of the sensing system and demonstrate the effectiveness of our system on a "semisynthetic" dataset of all New York City buses, where we simulate sensed trajectories by starting with fully labeled vehicle trajectories reported via the GTFS-Realtime protocol, removing the transit route IDs, and perturbing locations with synthetic noise. Using just the geometric shapes of the trajectories, we demonstrate that our system converges on the correct route ID within a few minutes, even after a vehicle switches from serving one trip to the next.

I. Introduction

As urban populations skyrocket world-wide, transportation agencies aim to promote public transit use, to cope with congestion and the environmental impact of private cars. Among the best ways to boost traveler confidence in public transit is exposing reliable real-time data about the transit system [5], [18]. A growing fraction of public transit agencies are investing in systems that allow travellers to track transit trips online. These systems typically aggregate data from hardware tracking devices mounted on transit vehicles. But such systems require non-trivial capital and operating expenses, both to install and maintain the hardware, and to maintain a reliable software infrastructure to aggregate and serve the tracking data in real time. As of 2019, a public real-time transit tracking feed in GTFS Realtime format [7] remains unavailable for the vast majority of the world's transit agencies.

When real-time transit data is unavailable, several alternatives involve sensing the real-time motion of transit vehicles from noisy and incomplete signal sources. In many transit systems, vehicles service multiple routes, so there is no trivial persistent association between a vehicle and a single route it serves. In this paper, we specifically consider the setting where we sense just the motion of transit vehicles but need to algorithmically label vehicles with the transit routes they are serving in real time (e.g., the "northbound bus #5"). This addresses a common case where the input signals do not include explicit, reliable labels, which can arise in several contexts:

- For some transit agencies, real-time transit tracking feeds may be available but contain errors in transit trip labels, e.g. due to human operator errors. In these cases, replacing the provided route label with one inferred from the trajectory may yield more reliable results, since the trajectory is sensed directly by on-board hardware.
- Vehicles can be detected via public imagery data from municipal sources, which may resolve a vehicle serial number painted on the bus, but not the dynamic headsign identifying the route.
- Vehicle trajectories can be inferred using roadway induction-loop detectors if they are densely placed and instrumented to classify vehicle dimensions.
- Some users may contribute user-generated reports of transit vehicle positions without manually labeling the report with the transit route they are on.

The problem has been studied in the past [16], [2], [19], but past work has either avoided addressing scalability by studying small networks of less than 10 routes, or has not considered the problem of *trip changes*, where a single vehicle can serve a series of different routes and go in and out of service.

Here, we present LiveTraVeL (**Live Tra**nsit **Ve**hicle **L**abeling), which addresses the general problem underlying these various scenarios: *given a stream of noisy reports of fresh transit vehicle positions identified by physical vehicle only, generate a stream of real-time estimated positions of the vehicles on each transit route*.

We abstract away the distinctions between the possible sources of unlabeled vehicle trajectory data by representing each vehicle trajectory as an abstract location time series: a sequence of (vehicle id, observation timestamp, estimated location) tuples. We also assume that a list of routes and their stop locations is available. This static information can be obtained in GTFS format for many more transit agencies compared to realtime data.

Given such a stream of observations, we design a geometric similarity approach that builds on dynamic time warping to label each vehicle observation with a transit route ID, from the vocabulary of all transit route IDs specified by the agency's static GTFS feed. Crucially, we seek to maximize the amount of time during which the vehicle receives a

[1] IST Austria
[2] Google Research

correct route assignment, including time periods when the vehicle switches from serving one transit route to serving another one.

We purposely focus on purely geometrical similarity between the trajectory observations and the static trip shapes, avoiding any dependency on the published temporal schedules of trips. In our experience, transit schedules for many of the world's agencies are often enough misaligned with observed behavior that it is valuable to have this system operate on the assumption that the schedule effectively doesn't exist. We leave to future work the integration of temporal signals into this system.

Our experiments focus on data covering the entire bus system of New York City. We picked this test setting to demonstrate the system's versatility in dealing with large urban environments with complex and densely interwoven transit networks. For these experiments, we generate a "semisynthetic" dataset by adding synthetic noise to reported vehicle trajectories, to model various kinds of measurement noise that depend on the specifics of how vehicle positions are sensed. Specifically, we take a GTFS Realtime [7] feed from an agency that reports physical vehicle IDs in addition to route IDs, hold out the route IDs, and add Gaussian noise to each vehicle observation. This dataset allows us to measure accuracy on a complex city transit network.

Our core contributions are:

- A new similarity measure, *ordered matching score*, for comparing a sequence of noisy vehicle observations to the shape of a transit route it might be serving. (§IV-D)
- A fast algorithm for matching vehicle observation sequences to routes, using an efficient *retrieval* step to reduce the number of costly similarity measurements that need to be computed. (§IV)
- An empirical demonstration of our algorithm's ability to identify the route a vehicle is following, to quickly adapt when a vehicle goes out of service or starts serving a new route, and to decide when it is not confident enough to make an accurate prediction, based on a large-scale semi-synthetic dataset. (§V)

II. Related work

Past work has used several approaches to match transit vehicle trajectories to the routes they are serving. Thiagarajan et al. [16] use a greedy algorithm to minimize a least-squares metric, with a preprocessing step to remove outlier locations. EasyTracker, by Biagioni et al. [2], models the routes as hidden states in an HMM and uses the Viterbi algorithm to find the most likely route. Zhou et al. [19] use cell tower sightings instead of GPS location reports, and use a local sequence alignment algorithm. Shah et al. [14] use Fréchet distance, but only to classify the mode of transportation rather than the specific route.

We are interested in handling *trip changes*, i.e. classifying the evolution of a vehicle's route assignment as it serves a sequence of multiple trips, sometimes going out of service. Of the listed methods, the only one besides ours that handles trip changes is EasyTracker [2]. The other prior results all assume that each trace will only cover time when the vehicle is in service and covering a single route. EasyTracker's HMM addresses trip assignment evolution using an additional "Unknown" state in the HMM. However, EasyTracker was evaluated on a 6-route dataset. Directly comparing EasyTracker to LiveTraVeL would be difficult, since we operate on a vastly larger dataset of 1241 New York bus routes, requiring us to carefully consider the computational cost. The running time of the Viterbi algorithm as used in EasyTracker scales with the product of the number of edges in the route map with the number of observations to be mapped.

Previous work also covers some related problems that we do not study here, including detecting when a mobile device is travelling in a transit vehicle [16], [19], [14], and inferring the transit route map from a collection of trajectories [2], [8] in the case that the static route map is not explicitly provided ahead of time.

Ferris et al. [5] found evidence that real-time data is helpful to users: after providing new tools giving transit users access to existing real-time bus data, they surveyed users and found that users were more satisfied with public transit and were able to use the system more efficiently.

An important ingredient in our method for matching vehicles to routes is a time series similarity measure: a way to evaluate how similar two ordered sequences of locations are. We explore two different time similarity measures: the well-known Dynamic Time Warping (DTW) algorithm, and a custom variation on DTW which we call *ordered matching*. Many other time series similarity measures have been studied. Several surveys address the options and tradeoffs: [13], [9], [17]. There is work on improving the runtime of both exact and approximate DTW [15], [1], [12], and Ceccarello et al. [4] show how to use locality sensitive hashing to implement approximate similarity search for Fréchet distance.

In this work, we reduce the computational cost by inserting a retrieval phase that decreases the number of routes to be scored using the similarity measure.

III. Problem Definition

The input to our system is a real-time stream of reports of transit vehicle locations, which we call "observations". Each of these observations consists of a vehicle ID, a (noisy) timestamp and a (noisy) estimated location. We also use a database of static transit data from GTFS. This includes transit stop locations and transit route geometries. GTFS allows a transit agency to also specify schedules for each transit trip. However, many of the world's transit systems show frequent large discrepancies between scheduled and actual trip timings, so we forego the schedule information entirely for the purposes of this paper, and leave the integration of noisy schedule data to future work.

We define a transit *route* to be a sequence of stops which are routinely served by transit vehicles in that order. For example, the stops served by "southbound bus #5" would constitute a route, while the reverse sequence of stops served

by "northbound bus #5" would constitute a different route by our definition. Transit route specifications typically include a polyline describing the full trajectory of a vehicle serving this route, which our algorithm can use to improve precision. A *trip* is the trajectory of a specific transit vehicle doing one complete pass of a transit route.

Given the static data and observation sequences of each vehicle, we aim to match these sequences, or substrings of them, to transit routes, allowing us to infer which route a vehicle is serving at any given time, and how far along the route it is. The goal is to get an overview of current locations of transit vehicles, which can be served to transit users directly or used for further user-facing inferences such as delay forecasting and alerting.

We consider three settings for matching vehicle observations to a route:

Offline. In the easiest variant, we have a sequence of observations that come from a single trip, not necessarily spanning the entire trip. We wish to predict which route the vehicle is serving. **Online.** Observations come from a single trip, but arrive in real-time, and at any given time we would like to predict which route the vehicle is serving. **Trip changes.** We have a sequence of vehicle observations from a single vehicle arriving in real-time. The sequence might include depot runs and trip changes.

IV. Route Matching Pipeline

We provide a pipeline for matching vehicle observations to transit routes that can be adapted to all three settings. The pipeline consists of two phases: a fast *retrieval* phase which uses a geometrically indexed data structure to quickly narrow the set of routes that the observations might match (§IV-C), followed by a *scoring* phase that ranks the retrieved routes (§IV-D). We use the scores to determine the likeliest route, and to decide when we are confident in our prediction by comparing the highest and second-highest scores.

A. Preprocessing

We store a sequence of *key points* for each transit route. The sequence includes the stops along the route, as well as equidistant sub-sampled points along the polyline connecting each pair of consecutive stops. (This means the distance between points will vary slightly between different pairs of stops, but ensures that whenever two routes share a consecutive sequence of stops, we extract the exact same key points for those stops, giving both the exact same score if all observations come from the overlapping segment.) The parameter determining the (approximate) distance between consecutive key points is called the *refinement length*. Ideally, it should be chosen small enough to ensure all routes have approximately the same density of key points (and thus chosen smaller than the distance between consecutive stops), as higher densities could give scoring advantages.

We then build a *geometric index* which can be queried for all key points within a given disc, to be used in the retrieval phase. This data structure is provided by the S2 spherical geometry library. [11]

B. Time complexity

The purpose of the retrieval phase is to reduce the number of routes that need to be considered by the slower scoring phase. It uses the geometric index to quickly narrow down the set of routes that the observations might match. This step runs in time $O(kn_{\text{near}})$, where k is a fixed parameter and n_{near} is the number of key points the most recent k observations pass near to. In particular, the runtime of the retrieval phase does not depend on the total number of transit routes.

After the retrieval phase, the slower scoring phase is applied to each retrieved route. Scoring a single transit route against an observation sequence takes time $O(|O||S|)$, where $|O|$ is the number of vehicle observations and $|S|$ is the number of key points along the route's geometry. Note that the refinement length thus carries a performance/runtime trade-off, as finer subsampling improves the precision of the scorer, but increases its runtime due to higher $|S|$. In the online setting, when one new observation comes in, we can update the score of a route in $O(|S|)$ time rather than reprocessing the full sequence O of observations.

C. Retrieval

To retrieve the set of candidate routes, the retrieval component identifies routes with many segments near the observation locations, as follows. Given a sequence of observations, the retrieval component computes a *query region*, which is the union of a disc of radius r around each observation location, for a given parameter r. It uses the aforementioned geometric index to return all transit routes which have at least k key points within the query region, where k is a parameter referred to as the *query cover*. Note that the ideal parameter choice for r and k depends on various factors such as the refinement length, the number of observations in queried observation sequences, average distance between consecutive observations and noisiness of the observation data.

D. Scoring

The scoring component computes a similarity measure between an observation sequence and a route, and updates it in an online fashion for each new incoming observation. For that purpose, it tries to match observation locations to key points along the route. We introduce a new similarity measure called the "ordered matching score". The idea is similar to dynamic time warping (DTW), which is a common approach used to give a measure of similarity for time series data. We will briefly outline both approaches and compare them.

The high level idea common to both approaches is that they try to find an alignment of two ordered data sequences that minimizes or maximizes a sum of point-wise scores. In our setting the two data sequences are the observations $O = \{o_i \colon 1 \leq i \leq |O|\}$ and key points $S = \{s_i \colon 1 \leq i \leq |S|\}$. An alignment is a set of pairs (o_i, s_j) respecting their orderings, i.e. if (o_i, s_j) is in A, then for $k > i$ the pair (o_k, s_l) can only be in A if $l \geq j$. We call such an alignment a *DTW-matching* if it contains each observation and key point *at least* once, and we call it a *matching* if

it contains each observation and key point *at most* once. If each pair $(o, s) \in A$ has a score $\text{score}(o, s)$, then the score of an alignment A is simply the sum of the point-wise scores of its pairs, i.e.

$$\text{score}(A) = \sum_{(o,s)\in A} \text{score}(o, s).$$

In dynamic time warping, this point-wise score $\text{score}(o, s)$ is simply the distance between o and s, $\text{dist}(o, s)$. DTW finds the DTW-matching between O and S with the minimum score, which is then called the similarity score between O and S. A smaller score indicates higher similarity. The algorithm to compute the minimum possible score runs in $O(|O||S|)$ using dynamic programming. It can be adapted to align a sequence of observations to a contiguous substring of key points rather than the entire sequence [10]. We use that adaptation in our experiments.

We introduce a new similarity measure which we call the *ordered matching score*. It differs from DTW in that the alignment has to be a matching, i.e. may contain each observation and key point at most once, and the aim is to maximize the sum of the point-wise scores of each pair. The score of an individual pair (o, s) is

$$\text{score}(o, s) = \max(0, R - \text{dist}(o, s))$$

for some constant radius R. The ideal choice of R depends on factors like the noise level and sampling frequency of the observation data. Like DTW, our measure can be computed via dynamic programming, yielding the same complexity. However there are some advantages. Due to not having to match every observation or key point, it automatically disregards outliers and can match substrings of the key points to the observation data. In fact, it can also naturally match a suffix of O to a prefix of S, which is useful when considering trip changes, while DTW does not easily extend to this problem. Also note that the ordered matching approach has a notion of "best possible score". It is achieved if every observation o is matched to a key point s at distance 0, yielding a total score of $R \cdot |O|$. Thus by dividing by this value, scores can be normalized into a range between 0 and 1, with 1 indicating a perfect match.

The algorithm for computing the ordered matching score uses dynamic programming. Let $s_{i,j}$ be the ordered matching score for matching the sequence of the first i observations to the sequence of the first j key points. Then $s_{i,j} = \max(s_{i,j-1}, s_{i-1,j}, s_{i-1,j-1} + \text{score}(o_i, s_j))$. The ordered matching score for the two sequences O and S then is $s_{|O|,|S|}$, and can be computed in $O(|O||S|)$. We summarize the algorithm in pseudocode:

```
for i in {0,...,|O|}:
  s[i, 0] = 0  # matching 0 key points
for j in {1,...,|S|}:
  s[0, j] = 0  # matching 0 observations
  for i in {1,...,|O|}:
    s[i, j] = max(s[i,j-1], s[i-1,j],
                  s[i-1,j-1] + score(o[i],s[j]))
final_score = s[|O|, |S|]
```

Notice that when computing $s_{i,k}$ for all $k \leq |S|$, we do not need the values of $s_{l,k}$ for $l < i-1$, and thus the computation can be done in linear space. This can be exploited in the online setting where observations come in one at a time: We only cache $s_{i,k}$ for all $k \leq |S|$ where i is the index of the most recent observation. Then when observation $i+1$ comes in we can still compute $s_{i+1,k}$ for all k.

If we track observation sequences for a long duration, we might want to have more recent observations contribute more to the score than observations further in the past. For this purpose we adjust the point-wise score of a pair (o, s) to be $\text{score}(o, s) \cdot e^{-c\Delta t(o)}$ with c being a decay rate parameter and $\Delta t(o)$ denoting the time difference between observation o and the most recent observation. Ordered matching scores can be computed as before, with the difference that before computing $s_{i+1,k}$ for all k, we scale the cached values of $s_{i,k}$ with a factor of $e^{-c\Delta t(o_i)}$.

Note that we still have a "best possible score" in this setting and thus can normalize scores. As we have $\text{score}(o, s) = \max(0, R - dist(o, s)) \cdot e^{-c\Delta t(o)} \leq R \cdot e^{-c\Delta t(o)}$, the best possible score is $\sum_{o\in O} R \cdot e^{-c\Delta t(o)}$. We can compute this score in an online fashion. If we cache the best possible score f_j for the sequence of the first j observations, we can get the best possible score for the first $j+1$ observations as $f_{j+1} = f_j e^{-c\Delta t(o_j)} + R$.

V. Results

This evaluation aims to study the route labeling accuracy of our algorithm and its ability to recover when a transit vehicle switches to a different route. We begin by studying components separately and in simplified scenarios, building up to the evaluation of the complete system, which handles trip changes and filters low-confidence predictions.

A. Dataset

Our experiments use a semi-synthetic dataset. The static route information that the algorithm needs as input, such as locations of transit stops, which sequence of stops a route is serving and polylines describing the trajectories between consecutive stops, comes from a static GTFS feed [6]. Note the distinction between static GTFS data and GTFS realtime data. In contrast to realtime data, the static GTFS map data used here is provided by many more transit agencies.

To obtain realtime vehicle observation data with ground truth labels for evaluation we added noise to vehicle position data reported by one of the few transit agencies that provides such a real-time data feed (in GTFS realtime format). We perturbed the latitude and longitude values with Gaussian noise to simulate noisy location data. The noise represents position reporting errors that may be introduced with different sensing strategies in real settings. Since this dataset is already labeled with routes, we hide these labels from our algorithm and evaluate the algorithm in terms of classification accuracy: that is, the rate at which the route labels generated by the algorithm match the realtime feed labels.

Specifically, the observation dataset we use in this section comes from the GTFS feed describing the buses of New

York City, as provided to Google Maps as of July 2018. This dataset was chosen as an example of a large, dense urban bus system, which has provided, in our experience, reliable static and real-time transit data, and reliably labeled distinct physical vehicle IDs in the real-time data. The NYC bus network in our dataset consists of 1241 bus routes. Many of them have a few stops in one district, followed by a long stretch without stops, such as along a highway, with more stops following in the destination district. The real-time bus data consists of observations that have a vehicle ID, a time stamp and an estimated location, and is implicitly labeled with a route ID which we use for evaluation. While serving a trip, each bus usually has a sampling rate of roughly one observation per minute. Gaps tend to be longer in between trips, though occasionally also during trips. During manual inspections, we found that occasionally trips or part of trips were mislabeled with the wrong route ID in the original data.

B. Outline of experiments

In our first experiment (§V-D), we show performance of the retrieval component for different parameter choices, giving us an educated choice for later experiments. We independently validate our scoring algorithm and compare it to DTW in an idealized transit network where routes never overlap (§V-E). Here we can aim for perfect labels, given enough observations to overcome the noise in vehicle positions. We then proceed to evaluate the full framework in a real transit network without trip changes (§V-F), providing a baseline for our labeling performance goal once trip changes are included. We then compare how trip changes affect our performance, as well as that of DTW (§V-G). To be useful in practice, the algorithm must make few mistakes. Finally, in (§V-H), we trade coverage for accuracy by filtering out predictions where the algorithm is not "confident".

C. Algorithm parameters

Due to long trip stretches without stops, subsampling of static trajectories for the purpose of scoring, as described in §IV, is crucial. As transit stops tend to be at least 0.1 km apart, we choose a refinement length of 0.1 km for all the routes in the network. We add Gaussian noise with $\mu = 0, \sigma = 0.001^\circ \approx 0.1\,\mathrm{km}$ to the latitude and longitude of each observation (except in §V-E, where we assess the effect of noise on the scoring). For the radius parameter R in the ordered matching scorer we used a value of 0.003°, i.e. only matching edges of spherical length up to $0.003^\circ \approx 0.33\,\mathrm{km}$ have a positive contribution to the overall score. The DTW scorer has no parameters that need to be set. For exponential decay, we use a half-life time of 10 minutes: observations 10 minutes in the past factor into the score with half the weight of fresh ones. Based on the first experiment, in the later experiments on the full framework, the retrieval query range r is chosen as 0.2 km and the query cover k as 10.

D. Retrieval

To understand the effect of parameter choices on the effectiveness of the retrieval component, we measure the number of candidate routes retrieved (out of the 1241 total routes) and whether the candidate set includes the correct route. Ideally, the retrieval component would return a small candidate set that still contains the correct route to decrease runtime of the following scorer without introducing errors.

Table I shows the performance of the retrieval component for different choices of the query radius r and query cover k. Each parameter choice was tested with the same sample of 1000 observation sequences, each sequence containing 30 observations. The results show that even conservative parameter settings can provide a tenfold reduction in input candidates while introducing very few errors. For all later experiments using the retrieval mechanism, we use $r = 0.2$ km and $k = 10$.

(a)

r \ k	10	12	15	20	25	30
0.1	14.3	11.7	8.9	6.2	4.0	1.8
0.2	32.8	27.1	21.4	15.9	12.4	10.1
0.3	49.6	40.8	32.1	21.0	17.7	14.2
0.4	72.0	56.4	43.6	31.3	23.1	18.2
0.6	101.5	94.5	73.3	50.1	38.3	29.2
0.8	116.7	110.9	102.1	74.9	54.8	42.6
1.0	129.1	124.1	115.7	101.4	76.0	57.9

(b)

r \ k	10	12	15	20	25	30
0.1	0.5%	0.7%	1.0%	3.6%	18.7%	56.2%
0.2	0.4%	0.4%	0.4%	0.7%	1.6%	2.2%
0.3	0%	0.3%	0.4%	0.6%	1.1%	2.0%
0.4	0%	0%	0.3%	0.4%	0.9%	1.8%
0.6	0%	0%	0%	0.4%	0.6%	0.9%
0.8	0%	0%	0%	0%	0.6%	0.8%
1.0	0%	0%	0%	0%	0.1%	0.6%

TABLE I: Retrieval performance as a function of query radius r and query cover k: (a) Average number of candidates returned. (b) % instances with correct route not retrieved.

E. Scorers in an Ideal Setting

We start the scoring evaluation with a simplified route network where transit routes may intersect, but not overlap (share route segments). Otherwise the best achievable prediction success would depend on the degree of overlap of the transit routes in the network, giving us no clear baseline. The simplified network is a subset of the New York City bus network consisting of 216 routes that might intersect, but have no overlapping segments.

Fig. 1 shows the labeling accuracy of the ordered matching and DTW scorers. For each scorer, noise level, and position along a 30-observation sequence, we show a *rank skyline*: the fraction of 100 trials where the correct label was always ranked in the top 1 (or 5) from that observation onward. We use the same pessimistic "skyline" metrics in Fig. 2 and 3. The two scorers perform comparably, and only need a few observations to reach nearly 100% accuracy. Predictably, with more noise, we need more observations to reach high accuracy.

F. Labeling Accuracy in a Real Setting

On networks with significant route overlaps, we cannot expect high accuracy. Fig. 2 shows the labeling accuracy of

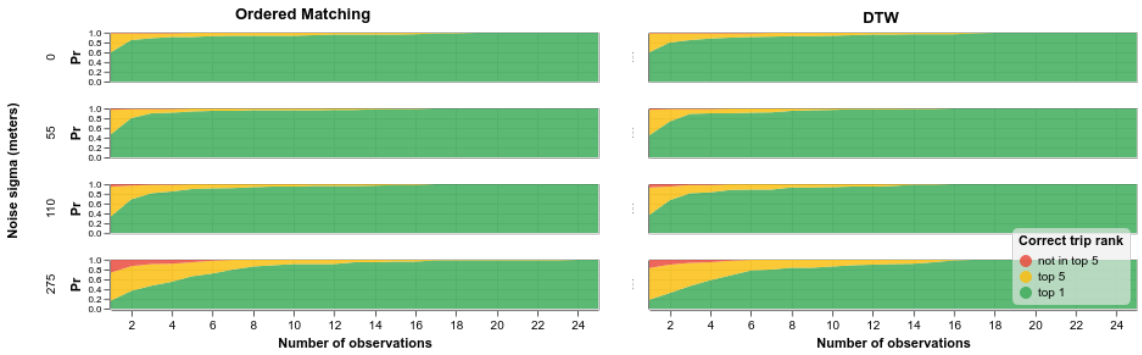

Fig. 1: Scoring performance in ideal setting by synthetic noise level.

the ordered matching scorer used together with the retrieval component. This time, we evaluate on the full New York City bus network. Each observation sequence belongs to a single trip, for observation window sizes ranging from 1 to 30. For each size (number of observations used in scoring), we used a sample of 250 trips of this length.

As expected, we observe a lower error rate when a longer window of vehicle observations is used. While we cannot expect this approach to always return the correct label due to overlap in routes, we would expect the score of the correct route to be close to the best scoring route once we use sufficient observations. The blue line confirms that, for most observation sequences, the score of the correct route rapidly comes within 10% of the top score. In fact, given the long stop sequences shared between routes, multiple routes are often exactly tied for the top score. In that case, all route candidates with the same score are ranked in a random order, decreasing accuracy. Performance of the DTW scorer is almost identical and thus omitted.

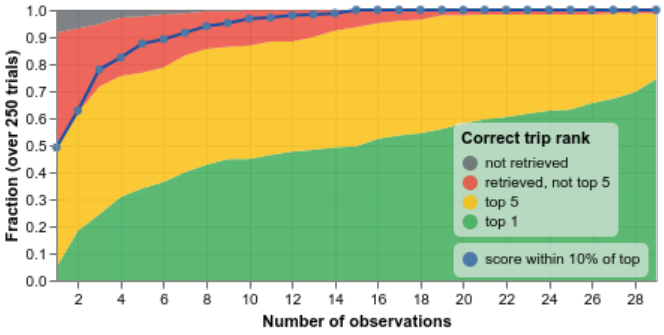

Fig. 2: Ordered matching scorer on the real NYC network.

G. Labeling Accuracy with Trip Changes

We extend the previous experiment to observation sequences where the vehicle switches route within the observation window, so the older observations belong to a different route from the one the vehicle is now serving. If we use the basic scorers that assign the same weight to all observations, we can only expect reasonable predictions if at least half of the observations come from the trip after the change. Thus while longer observation sequences usually mean higher accuracy, here they'd also mean more delay until a trip change is detected. Furthermore, discarding old observations is impossible in the online algorithm without recomputing the score for the full observation sequence.

We address this issue by devaluing older score contributions of observations using exponential decay. This provides relatively quick detection of the new route after the trip change. This also lets us elide observation windows for scoring, since very old observations barely impact the score anyway. Instead, the time needed to correctly predict the new route after a trip change is determined by the *half-life*, i.e. the age of an observation at which it contributes half the weight compared to the most recent observation.

Using exponential decay with a half-life of 10 minutes, we compare the ordered matching and DTW scorers in Fig. 3. As before, the sample size for each data point is 250 observation sequences. We start with 30 observations from the route before the trip change, and add new observations one by one. Notice how the ordered matching scorer tends to achieve the baseline established in Fig. 2 after only about 10 observations. Also notice how the DTW scorer tends to need more observations to detect the route change.

H. Putting it all together: Confidence filtering and accuracy-coverage tradeoffs.

To reduce labeling errors, the system can output no label at all when it has low confidence in the result. This is vital when label correctness matters more than labeling every trip. We first estimate confidence by comparing the scores of the two highest scoring route labels. Intuitively, a large gap implies a lower chance of confusion and higher confidence in the label. Using the same data set as above, Fig. 4(a-b) shows the behavior of confident predictions when a score gap of $1.1\times$ is required to output a label, for the DTW and ordered matching scorers respectively. We use *success rate (SR)* and *error rate (ER)* for probabilities of confidently outputting the right and wrong labels, respectively. Though our *response coverage* ($SR+ER$) is always substantial with this approach — we output *some* label confidently over 30% of the time — ER remains high for a while after a trip change. The scorer is likely still returning the old trip then.

To further reduce errors after a trip change, we refine the confidence criterion with a minimum score threshold. (This

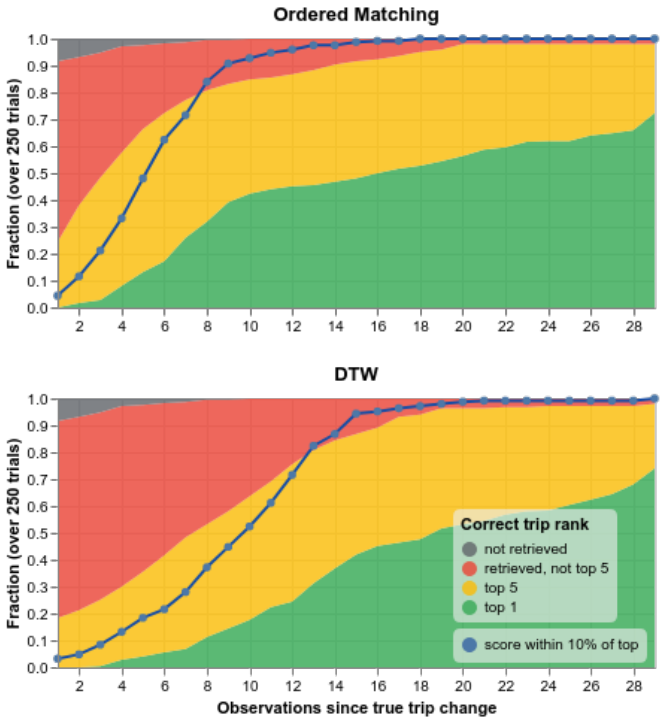

Fig. 3: Trip labeling performance soon after a vehicle switches trips, using exponential decay.

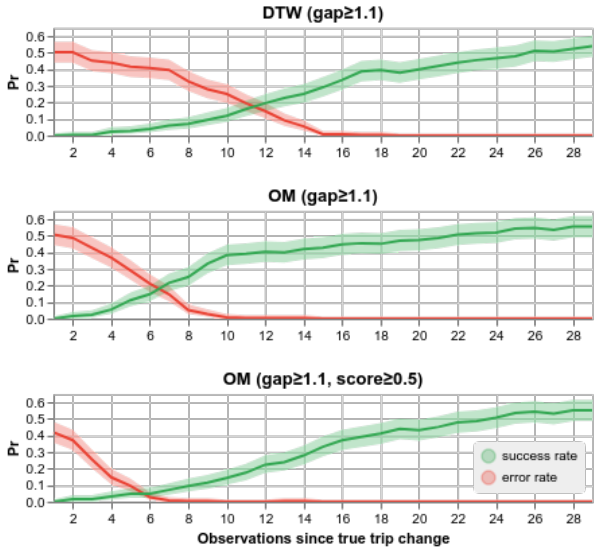

Fig. 4: Success rate and error rate with confidence thresholding based on gap above 2nd-highest score and/or absolute score. Clopper-Pearson 90% confidence intervals [3] are shown.

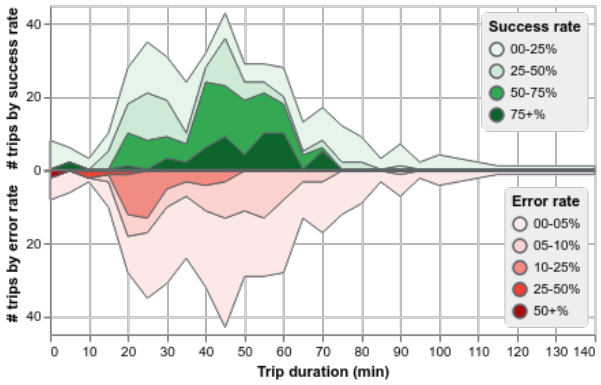

Fig. 5: Distributions of success and error rates for trips from full-day trajectories of 50 vehicles, by trip duration.

is only possible for the ordered matching scorer.) This comes from the observation that while usually the score of the top scoring line variant is well above 0.5, after a trip change it tends to dip, as no route fully matches the observation sequence. Fig. 4(c) shows results of combining the 1.1 gap threshold with a 0.5 minimum absolute score requirement. This decreases the error rate when the trip change was very recent. Overall, confident labels can be returned within 5-10 location observations after a trip change.

To further evaluate the SR-ER tradeoffs in a real-world setting, we used observation data from 50 different vehicles, each for a full day, and after each observation inferred the vehicle's route. Fig. 5 shows SR distributions by trip duration. We only used a relative score cutoff of 10%. SR can be traded against ER with different cutoff parameters, though they are correlated (Fig. 6).

Some trips, especially those above 40 minutes in duration, are correctly labeled for most of the trip. This indicates that with a carefully chosen subset of routes the system can provide higher accuracy real-time information about vehicles serving these routes with very high SR.

VI. Discussion and Possible Extensions

We saw that there is latency before we can detect the new route after a trip change. One approach to decrease this latency is to try to explicitly detect trip changes, and discard all observations that were made before the trip change. Using data from only the new trip reduces, though not drastically, the time needed to detect the route for the new trip: compare Ordered Matching performance between 3 and 2. In most instances, the new trip starts near the previous trip's endpoint (see Fig. 7). We could exploit this by limiting route candidate choices for the new trip to routes starting close to the previous end point, at the risk of occasionally never retrieving the correct route.

Our scoring method allows for various mechanisms to help explicitly detect trip changes. For example, on average, the absolute score of the highest scoring route drops after a trip change. Furthermore, as the scoring algorithm creates a matching between observations and stop locations, a trip change is likely if the matching of the highest scoring route includes the last stop of the route. We implemented trip change detection using the latter method, and constrained candidates for the new route to start close to the end of the previous route. However it did not outperform the simpler scheme using exponential decay as in Fig. 3. A likely reason is that any wrong trip change detection will lead

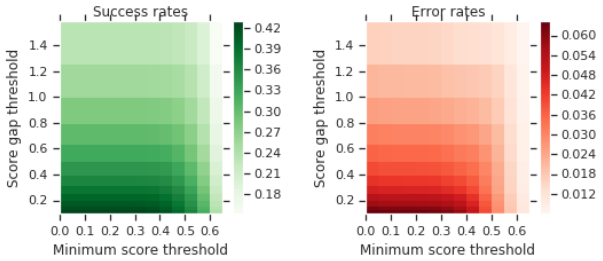

Fig. 6: Effect of score gap and minimum score thresholds on success and error rates for confident predictions, over full-day trajectories of 50 vehicles.

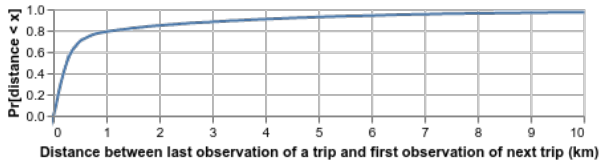

Fig. 7: CDF of trip-to-trip transition distances, from a sample of 5789 trip changes.

to false assumptions and almost certainly cause a wrong prediction, and if multiple routes with different endpoints share a common stretch with the observation sequence, this is likely to occur. Additionally, per Fig. 2, even discarding observations for the previous trip does not lead to an instant good prediction for the new trip.

A different information channel that could be used to predict routes after a trip change is past occurrences of trip changes. Certain trip changes might occur more commonly than others, e.g. from a route to its reverse route. We recorded a training set of 4179 trip changes, and for each route, we recorded the set and frequency of routes that were served directly after a trip change from that route. With a network of 1241 distinct routes, this limited data set only gave us 3.4 trip changes per route on average, with many even only observed once. Nevertheless, using for each route the most common followup route after the trip change as a prediction, in the evaluation dataset of size 2094, 66% of route changes were predicted correctly (in 3.6% of the cases, no prediction was made as the route had not been encountered before).

While in general not a reliable predictor, this data could be used as a prior when trying to assign probabilities to routes soon after a trip change.

VII. Conclusion

We introduced the LiveTraVeL real-time transit labeling algorithm, which matches a stream of transit vehicle positions to the transit route the vehicle is serving. To improve its accuracy and runtime in dense areas with many nearby transit routes, the system is separated into a lightweight candidate retrieval phase and an ordered matching route similarity scorer. The former pre-selects a smaller set of route labels to reduce the load on the more computationally expensive scorer. We prototyped and evaluated each component individually to assess parameter choices and the effect of noise and vehicle positions needed, and confirmed that the scoring component is close to perfect accuracy in an idealized setting. We then evaluated the accuracy of the full system on a real-world dataset with over 1500 simulated trips from the full New York City bus system (1241 routes). Here, accuracy degrades to about 60% but this can be addressed by only returning results when labeling confidence is high. In this case, the system can return labels for about 40% of trips with a near-zero error rate. We further find that the event of a vehicle switching to a new route can be detected within 5-10 location updates (assuming 30-60 s location reporting intervals). Overall, these results show promise for automatically generating route labels when they are unavailable or when additional validation of manually generated data is desired.

References

[1] Pankaj K. Agarwal, Kyle Fox, Jiangwei Pan, and Rex Ying. Approximating Dynamic Time Warping and Edit Distance for a Pair of Point Sequences. In *Proc. SoCG*, volume 51, pages 6:1–16, 2016.

[2] James Biagioni, Tomas Gerlich, Timothy Merrifield, and Jakob Eriksson. EasyTracker: automatic transit tracking, mapping, and arrival time prediction using smartphones. In *Proc. Sensys*, 2011.

[3] Lawrence D. Brown, T. Tony Cai, and Anirban DasGupta. Interval estimation for a binomial proportion. *Statistical Science*, 16(2):101–117, 2001.

[4] Matteo Ceccarello, Anne Driemel, and Francesco Silvestri. FRESH: fréchet similarity with hashing. *CoRR*, abs/1809.02350, 2018.

[5] Brian Ferris, Kari Watkins, and Alan Borning. OneBusAway: Behavioral and satisfaction changes resulting from providing real-time arrival information for public transit. In *Proc. TRB*, 2011.

[6] General Transit Feed Specification (GTFS). `https://developers.google.com/transit/gtfs/reference/`.

[7] GTFS Realtime Specification. `https://developers.google.com/transit/gtfs-realtime/reference/`.

[8] How we mapped the world's weirdest streets. `https://medium.com/transit-app/hello-nairobi-cc27bb5a73b7`, 2015.

[9] Nehal Magdy, Tamer Abdelkader, and Khaled El-Bahnasy. A comparative study of similarity evaluation methods among trajectories of moving objects. *Egyptian Informatics Journal*, 19(3):165 – 177, 2018.

[10] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[11] S2 Geometry Library. `https://s2geometry.io`.

[12] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intell. Data Anal.*, 11(5):561–580, October 2007.

[13] Joan Serrà and Josep Lluís Arcos. An empirical evaluation of similarity measures for time series classification. *CoRR*, abs/1401.3973, 2014.

[14] Rahul C. Shah, Chieh-yih Wan, Hong Lu, and Lama Nachman. Classifying the mode of transportation on mobile phones using GIS information. In *Proc. UbiComp*, pages 225–229, 2014.

[15] Saeid Soheily-Khah and Pierre-François Marteau. Sparsification of the Alignment Path Search Space in Dynamic Time Warping. *Applied Soft Computing*, 78:630 – 640, 2019.

[16] Arvind Thiagarajan, James Biagioni, Tomas Gerlich, and Jakob Eriksson. Cooperative transit tracking using smart-phones. In *Proc. Sensys*, pages 85–98, 2010.

[17] Haozhou Wang, Han Su, Kai Zheng, Shazia Sadiq, and Xiaofang Zhou. An effectiveness study on trajectory similarity measures. In *Proc. Australasian Database Conf*, pages 13–22, 2013.

[18] Kari Edison Watkins, Brian Ferris, Alan Borning, G. Scott Rutherford, and David Layton. Where Is My Bus? Impact of mobile real-time information on the perceived and actual wait time of transit riders. *Transp. Res. Part A: Policy and Practice*, 45(8):839 – 848, 2011.

[19] Pengfei Zhou, Yuanqing Zheng, and Mo Li. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In *Proc. MobiSys*, pages 379–392, 2012.