Delay Estimation for Shared Rides From GPS Data

Sepehr Samavati[1] and Alexander Nemirovskiy[1] and Matteo Rossi[2]

***Abstract*— Ride-sharing is one of the most innovative topics in the new era of intelligent transportation. RIDE2RAIL is a European initiative that aims to design a system that connects multiple shared rides to one another or to other modes of transportation. Accurate time planning and high user satisfaction are key factors for the success of ride sharing schemes, which entails that accurately estimating the actual duration and delays of rides is essential. This paper proposes a method to estimate delays in which shared rides incur during their execution. The proposed mechanism relies on link-based route time estimation: It first makes an initial estimation based on a set of parameters related to the city and the situation (e.g., time and day of the ride); then, it dynamically adjusts the estimation based on GPS data that is continuously received during the ride. The mechanism has been implemented and tested using data collected from rides taken in the cities of Milan and Tehran. The results of the evaluation, which has been performed using several error criteria, showed that the proposed approach has a good level of accuracy.**

I. Introduction

The advent of widespread geo-location and communication technologies has fundamentally altered the transportation environment. Thanks to GPS systems and the internet, new mobility concepts and enterprises have been created. One of these services is *ride-sharing*, which is ”a way for multiple riders to get to where they're going by sharing a single vehicle, like a car or van, that's going in their direction. This vehicle makes multiple stops along a route to pick up and drop off passengers, reducing the need for multiple cars on the road.”[1]

RIDE2RAIL[2] is a European project that aims to combine ride-sharing services with other forms of transportation. The goal of the RIDE2RAIL project is to provide users with a variety of options for their trips.

Every ride-sharing situation involves a driver and one or more passengers. Since passengers must typically wait for the ride to arrive at some point, the service's timing is critical. Timing is also crucial to facilitate the integration of ride sharing into a wider, multi-modal transport ecosystem. For example, the shared ride could be used to reach a transportation hub such as a train station or an airport, or it could be followed by another shared ride. In this scenario, planning and monitoring the timeliness of the ride (i.e., estimating the delay, if any, affecting the ride) is crucial since the ride, for all intents and purposes, operates like a minibus connected to a wider transportation network, hence it must meet its appointed target times (at the final destination, but also at intermediate pick-up points). Existing tools (e.g., Google Maps®) that allow users to compute the duration of a trip taking into account traffic conditions could be used also for delay estimation. However, exploiting such tools would typically require issuing many requests to—and possibly overload—online services to keep the duration estimation updated, especially if drivers change path frequently or make stops (for example, to pick up new passengers).

This paper introduces a mechanism to detect delays affecting a shared ride using GPS data of the vehicle in an urban environment. The proposed algorithm relies on simple computations, which could, in principle, be executed even locally, on the user's mobile device. Indeed, delay information is key to, on one hand, inform passengers of disruptions impacting their trips and, on the other hand, trigger the re-planning of the trip if the delay negatively impacts the ability to make the necessary connections. The GPS data is retrieved from the driver's mobile device, which provides the delay estimator (which could be run locally or remotely) with the driver's position information.

The paper is structured as follows. Section II briefly describes related works and highlights the differences of our work with respect to them. Section III introduces some necessary knowledge and background. Section IV presents the delay estimation method and workflow. Section V describes the data set used to evaluate the proposed method and Section VI discusses the results of the evaluation. Finally, Section VII concludes.

II. Related Works

In many situations time management is a challenging task, and even more so in urban transportation, where many variables affect the timing of trips (the type of vehicle, the driver abilities, the congestion of the roads, etc.). In the literature, many techniques have been proposed to estimate the trip time/delay. They can be separated in two categories: *link-based* and *path-based* [1]. Link-based methods divide the path into segments and calculate the result based on the estimation computed for each segment. Path-based methods, instead, consider the whole path and do not divide it into segments. The majority of available methods statistically estimate route travel time from huge amounts of vehicle data. [2] and [3], for example, use different observations

*Work supported by Shift2Rail and the EU H2020 research and innovation programme under grant agreement No: 881825 (RIDE2RAIL)

[1]Sepehr Samavati and Alexander Nemirovskiy are with Dipartimento of Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy sepehr.samavati@mail.polimi.it, alexander.nemirovskiy@polimi.it

[2]Matteo Rossi is with Dipartimento di Meccanica, Politecnico di Milano, Milan, Italy matteo.rossi@polimi.it

[1]www.remix.com/blog/ride-hailing-vs-ride-sharing-the-difference-explained

[2]ride2rail.eu

to estimate route travel time. They are also based on low-frequency Floating Car Data (FCD), and try to guess the travel time for the whole path, not for every link in the path. Similarly to [3], [4] (which focuses on short time estimation) and [5] also use statistical data-driven methods to find the travel time for the whole network. However, unlike [2] and [3], they use link-based methods, rather than path-based ones. In addition, [5] establishes a relation between travel durations in neighboring areas, a result that we exploit in our work. [1] proposed a method to estimate travel time using a large data set of taxi drivers' GPS data; in this case, the data set is large enough that it can be used to build a reference model, instead of relying on a specific statistical model. [6] focuses on bus planning and delay estimation; in particular it estimates the bus's delay from its location using a basic algorithm and a mobile application and then uses the result to create bus schedules.

The work presented in this paper takes a unique approach to estimate trip time and delay. It is not based on detailed statistical data collected from rides throughout the city. Instead, it distills that information into a set of parameters to first establish a general guess on travel time, then it adjusts the guess dynamically during the ride. It also assumes that GPS data is available with higher frequency than what is assumed by other works, and proposes a link-based approach.

III. Preliminaries

This section provides a brief overview of the concepts and technologies on which the present work is based. First, it defines the addressed problem and some related terminology. Then, it describes the principles underlying this work, such as map services and map matching. Finally, it introduces various error criteria that will be used in the final evaluation.

A. Problem Definition

The goal of this work is to develop a mechanism to *estimate the delay that a ride-sharing driver will accumulate at their destination, using the following inputs:*

- Geographical coordinates of the origin and destination of the ride.
- Intermediate way points, which we assume to be available through some direction API; in the rest of this paper, we call this route the *direction* of the ride.
- Expected ride duration, as returned by the direction API, which is called T_D; the expected duration may or may not take into account traffic data.
- GPS samples retrieved during the ride; we assume that each GPS sample has the format $\langle \text{latitude, longitude, speed, timestamp} \rangle$.

The delay is defined as $Delay = T_R - T_D$, where T_R denotes the actual ride duration. We call T_E the output of the delay estimation procedure. We aim to make T_E a good estimation of T_R according to the criteria discussed in III-D.

B. Map Services and Open Street Map

A map service is a way to digitally represent a region or city map. It features several real-world models, such as a graph that shows roads and their crossings. Map services also offer users a variety of tools, such as area information (places, distances, traffic, and so on) and practical tools (best route direction, travel time, etc.). These services frequently include an API made available to developers. Many firms, such as Apple®, Google®, and Microsoft®, offer map services, each with its own set of tools and features. There are also many open-source projects, such as Leaflet[3], Modest Maps[4], and Polymaps[5], that have developed map services.

Open Street Map[6] (OSM) is one of the most important open-source map services. It has a thriving developer community, and many additional libraries based on OSM are available, in a variety of programming languages. This work uses OSM, and in particular the OSMNX[7] Python package, to obtain the graph of the region map through which the user is driving.

C. Map Matching

Using GPS data is usually challenging due to the considerable imprecision of these data. GPS data must be projected on the map graph to be used, and this operation is called *map matching*. A variety of techniques have been devised in the literature to solve the map matching problem [7]. In this work we use the Fast Map Matching (FMM) method, and in particular the open-source Python FMM library [7].[8]

D. Error Criteria

To evaluate the performance of the delay detection mechanism presented in this work, we consider different notions of error, which are described in the following. Figure 1 shows a visual representation of these criteria.

- **Root Mean Square Percentage Error (RMSPE):** as defined in [8], the RMSPE is given by Equation (1). It captures the average of the relative error through all estimations T_{Ei} for each sample.

$$RMSPE = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} \left(\frac{T_R - T_{Ei}}{T_R}\right)^2} \cdot 100 \qquad (1)$$

- **Error Margin:** this is defined as the maximum acceptable value of the estimation error. For example, in Figure 1 the error margin is depicted by the green horizontal lines, which highlight the maximum desired difference of the estimation with respect to the actual trip duration T_R. A good estimator should keep T_E in the vicinity (percentage-wise) of T_R.
- **Confidence Time Percentage:** The *confidence time* (T_{Conf}) is defined as the elapsed time from the start of the trip after which the estimation gets within the specified error margin (see Figure 1 for a graphical depiction: the confidence time is the point where the

[3]https://leafletjs.com
[4]http://modestmaps.com
[5]http://polymaps.org
[6]https://www.openstreetmap.org
[7]https://github.com/gboeing/osmnx
[8]https://fmm-wiki.github.io/

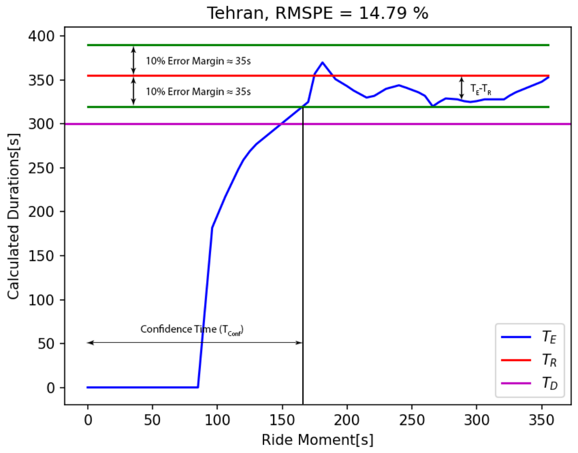

Fig. 1: Error criteria.

estimation enters the region between the green horizontal lines). The *confidence time percentage* is the ratio of the confidence time to the trip duration computed by the direction API (i.e., T_D).

IV. Delay Detection

The delay detection procedure proposed in this paper has two parts: *delay estimation* and *deviation recognition*. The rest of this section describes the main structure of the procedure and its parts, which are depicted in Figure 2.

At startup, the system takes as input a set of coordinates corresponding to the "direction" of the ride (as returned by the direction API), calculates the map boundaries surrounding these points, and uses a map service to retrieve the graph of the requested region. Then, the map-matcher module projects the direction points onto the map graph and obtains the path of the direction.

After the initial step, the system continuously receives GPS samples throughout the user's ride. It concatenates each new sample with a predetermined number of previous positions and sends the result to the data filtering and refinement block. The latter corrects—if necessary—the new sample, and discards it if it is determined to be faulty data. The modified sample is stored for the subsequent iterations. The map-matcher module then projects the new point on the map, along with previously cached points, to establish the actual path of the driver. The deviation detection block receives the direction path, the path inferred from the latest samples, and the new sample data, and determines if the user deviated from the intended direction. In the latter case, the deviation detection module notifies a suitable component, which recalculates the direction and adjusts the delay accordingly.

In the next step, the delay estimator block computes the travel time/delay based on the inputs listed above. First, it makes an initial guess on the trip time based on the received inputs and on prior knowledge. Second, it adjusts the estimate by applying a suitable factor (called λ) and by detecting and discounting stops made by the driver along the path, until it outputs a final estimate for the delay. The parameters of the delay estimator block are determined from prior information, and are updated as new data points arrive.

The rest of this section provides some details concerning the delay estimation and deviation recognition blocks.

A. Delay Estimation

The delay estimation process is made of several steps, which are outlined in the following.

a) Initialization: In the first phase, the algorithm requires some information about the direction edges from the map service. More precisely, these data indicate the amount of time it takes to travel through each edge on the path. The map service determines this value, which we call the edge's *time contribution*, by dividing the length of the edge by the average speed of vehicles on that map segment. The total duration of the ride is given by the sum of the time contributions of all direction edges, as defined by the following formula:

$$T_{Total_{Map}} = \sum_{e \in E_D} TC_e \tag{2}$$

where TC_e is the time contribution of edge e and E_D is the ordered set of direction edges. However, $T_{Total_{Map}}$ is usually different from T_D (i.e., the total time returned by the direction API, as explained in Section III-A) because the direction API may take into account various factors (e.g., the traffic along the route) of which the map service is not aware. To match T_D (which does not consider the time it takes to go through the single links) and $T_{Total_{Map}}$, the algorithm multiplies each edge's time contribution by a ratio, as captured by Equation (3).

$$\forall e \in E_D : TC_{e_{new}} = TC_e \cdot \frac{T_D}{T_{Total_{Map}}} \tag{3}$$

b) Hour/Day Ratio: Based on the information computed above, the algorithm then makes an initial guess for the actual time of arrival T_R. The ride time, as is well known, varies depending on the day of the week and the hour of the day, and different cities typically display traffic patterns. The goal of this stage is to create an initial guess based on information regarding these variations. We do so by retrieving, through Google Direction[9], the expected travel time for a predefined set of "test" paths and considering different starting times and days of the week; a set of "test" paths must be defined for each city in which we want to deploy the system, since different cities have different traffic patterns during the week. Essentially, we assume that the Google Direction API, in its estimations, considers many factors (e.g., traffic conditions) based on a very large set of real data; hence, we use it to determine certain necessary parameters without having to execute actual experiments ourselves. Figure 3 shows the outcome of this experiment for what concerns the city of Milan. The travel time computed

[9]https://developers.google.com/maps/documentation/directions

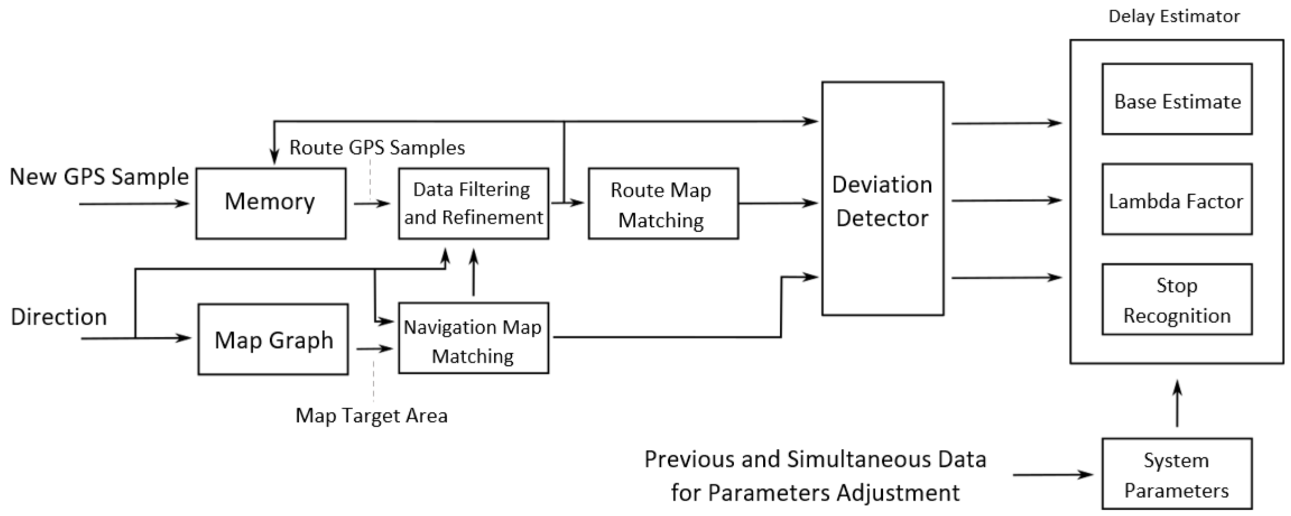

Fig. 2: System main structure.

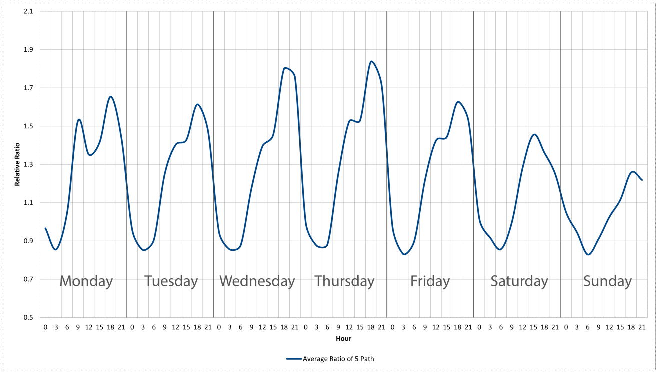

Fig. 3: Hour/Day ratio curve relative to the city of Milan.

considering as starting time Sunday at midnight is used as reference, and the other times are shown as a ratio with respect to it. Next, the algorithm uses least square regression to fit the curve in Figure 3 to training data-sets (see Section V). Finally, it updates the time contribution of each edge using Equation (4).

$$\forall e \in E_D, TC_{e_{new}} = TC_e \cdot Ratio(Day, Hour) \qquad (4)$$

The sum of these time contributions results in the initial guess for the whole ride duration.

c) Map Matching: Following the initial phase, which produces the initial estimate of the travel time, the algorithm goes through a series of steps for each GPS sample it receives. It uses the map-matcher module (which employs the FMM method, as mentioned in Section III-C) to find the location of the new sample on the map as the first step. It then determines the edge of the map graph on which the vehicle is, and the distance already covered on that edge. If this edge belongs to the direction being followed, the λ factor (explained later) is applied. Otherwise, the samples are filtered and refined, es described in the following.

d) Filtering and Refinement: Due to measuring noises and inaccuracies, GPS samples can have offsets, which can cause the map-matcher module to place them on the wrong edge. In this scenario, the algorithm locates the direction's point that is closest to the sample coordinate. All geometric elements of roads, such as turns, are considered in this step. If the distance between the point and the direction is smaller than a predefined threshold (which varies per city), the algorithm replaces the sample's coordinates with the nearest direction point. This causes the samples to be attracted by the direction. Points that do not exist or that are over the threshold are not considered when the λ factor is applied. If, later, the deviation recognition block determines that these points are not indicative of a deviation being taken by the vehicle, the algorithm will discard them.

e) Stop Recognition: Next, the algorithm examines the input sample for the presence of a halt in the ride. This step is necessary because several steps in the delay estimation process consider the traffic flow, and a car stop that is unrelated to traffic adds an offset value to the total duration of the ride that should be discounted when projecting the final delay (while still being counted in the total duration of the trip). A stop in this context refers to a situation where the car halts for reasons unrelated to driving rules. For example, short traffic stops and red lights are not considered "stops", and they should be considered in the initial guess. On the other hand, this feature covers situations in which traffic accidents occur: this step will recognize the unusual stop and add the stop duration to the final value.

For stop recognition, the algorithm uses the speed values in GPS samples. These values first go through a low-pass filter whose definition is shown in Equation (5).

$$V_F = 0.7 \cdot V[n] + 0.2 \cdot V_F[n-1] + 0.1 \cdot V_F[n-2] \qquad (5)$$

where V_F is the filtered speed and V is the raw speed. The time constant of this filter is 15 seconds, so after about 15

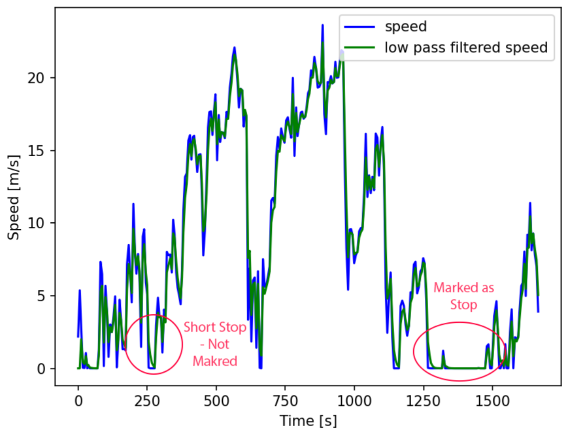

Fig. 4: Speed filtering for deviation recognition.

seconds of having zero raw speed, V_F will be almost zero. The filter helps ignore instant zero speeds or short traffic pauses. Figure 4 shows the raw and filtered speed of a sample ride. After filtering, the algorithm sums the duration of stop intervals. This sum is called T_{Stop}. A stop interval is defined as the range between the point whose filtered speed is less than a predefined threshold $V_{\text{Stop\_Threshold}}$ and the next point.

f) Lambda Factor: The algorithm starts with a guess for T_R, but each ride's condition differs from the others. As a result, it should have an adaptive component that corrects the estimation during the ride.

The work presented in [5] showed that there is a correlation between the trip times of neighbouring sections. We use this result in our work, and we assume the traffic flows of edges that are connected on the map graph to be correlated. As a result, at any point during the ride, the algorithm will use the most recent observation to update the estimated remaining time of the ride.

More precisely, the algorithm uses a factor, called λ, to regulate the impact of the current observation on future steps. Because the correlation between traffic flows decreases for links that are further away from the current one, this effect should be reduced along the path. The ratio R_U that is used to update the time contribution of each edge e from the current one until the end of the ride is defined, as shown in Equation (6), as the ratio between the actual time spent up to the current point, minus the stops, and the estimated time spent up to this point. All values involved in the definition of R_U are considered in a limited recent window of time, which is defined by the number w of recent points that contribute to the computation.

$$R_U = \frac{\text{Real Time Passed}_w - T_{stop}}{\text{Estimated Time for Passed Edges}_w} \tag{6}$$

The new estimate of the time contribution of each relevant edge e (where an edge is relevant if it is in the recent window w, or if it follows the current one) is computed according to Equation (7), where e_c is the current edge, e_w is the first edge which is inside the defined recent window of size w, and d is the number of edges between e_c and e.

$$\forall e \in E_D \text{ such that } e > e_w,$$
$$TC_{e_{new}} = \begin{cases} TC_e \cdot R_U, & \text{e } \leq e_c \\ TC_e \cdot \big((R_U - 1)\cdot \lambda^d + 1\big), & \text{e } > e_c \end{cases} \tag{7}$$

After updating the time contributions TC_e, the algorithm computes a new estimation for T_E using Equation (8).

$$T_E = \text{Real Time Passed} + \sum_{\substack{e \in E_D, \\ e > \text{current edge}}} TC_e \tag{8}$$

The value of parameter λ is suitably chosen, based on previous data collected in the city of interest, to minimize the estimation error (as discussed also in Section VI).

B. Deviation Recognition

There is a possibility in every ride that the driver deviates from the planned route. This divergence could be slight or significant. A slight diversion occurs when the driver returns to the main path after a short amount of time or distance. A significant deviation is one that is not slight and requires a direction recalculation.

As explained in Section IV-A, the deviation recognition block examines the samples that could not be refined. If four consecutive samples cannot be refined, the algorithm recognizes a deviation. Then, this block uses Dijkstra's algorithm to compute the shortest path to each remaining node of the direction. The algorithm determines the shortest path back to the main direction after considering all feasible routes. The prior adjustment ratios up to this point are then applied to the new route edges. Finally, T_E is re-computed using Equation (9), where $E_{D'}$ is the set of edges which form the detour and e_r is the first edge where the driver returns to the main path.

$$T_E = RealTimePassed + \sum_{e \in E_{D'}} TC_e + \sum_{e \in E_D}^{e > e_r} TC_e \tag{9}$$

V. Data Set for Evaluation

To evaluate the accuracy of the delay estimation mechanism according to the criteria defined in Section III-D, a suitable set of real-world data is needed. In the rest of this section we discuss how such data set was created.

a) Data Collection Application: An application for Android-based mobile devices has been created to collect the assessment data set. The application uses Google Maps as a map service, and the Google Direction API as direction API. Application users declare a start point and a destination, and in turn receive a direction to follow during the ride. After completing the ride, the user should submit the data collected by the application during the ride and indicate whether or not they followed the given directions. The data is then sent to an online cloud server, where it is stored. The collected information covers all ride details, as well as the algorithm

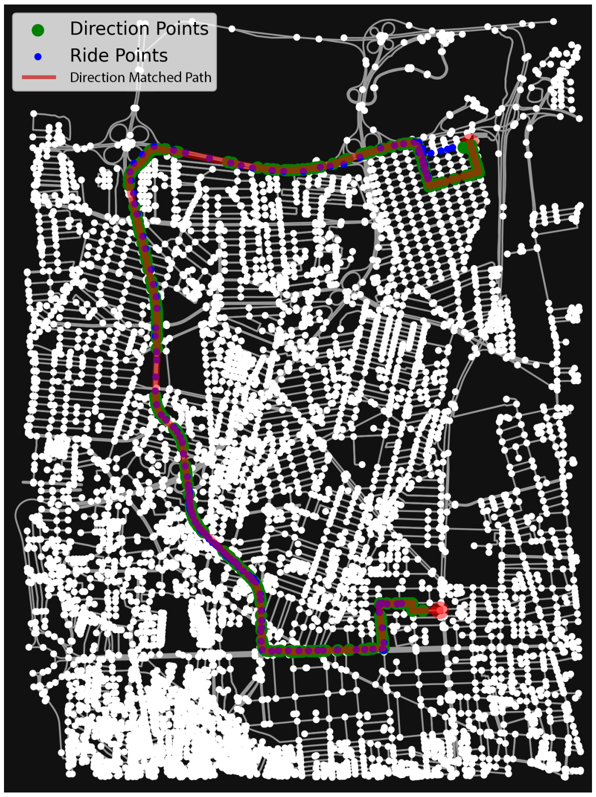

Fig. 5: Ride submission example.

inputs specified in Section III-A. It also contains the actual duration of the ride, which will be considered as the value of T_R.

b) Experiment and Data Cleansing: Evaluation data was collected in two cities, Milan and Tehran. A number of volunteers installed the app on their phones and made submissions during their daily commutes. They made a total of 92 submissions over the course of three months.

A data cleansing phase is essential since there are always some flaws and inaccuracies in data submissions. The data cleaner first double-checks the beginning and end of the rides. The ride starts when the car starts moving and it comes to an end when the car reaches a certain distance from the destination. Second, all data have been manually checked and tagged. Figure 5 shows an example of submitted path.

Not all collected rides are suited for our purposes, and some had to be discarded, for example because they were not completed. Table I summarizes the data collected for the evaluation of the delay estimation mechanism.

	Milan	**Tehran**
Participants	2	7
Total Submissions	29	63
Followed Direction	12	30
Minor Deviation	4	3
Major Deviation	8	16
Discarded	5	14

TABLE I: Summary of evaluation data.

VI. Result and Evaluation

The delay estimation mechanism was implemented using the Python 3.7 programming language and it was evaluated according to the criteria mentioned in Section III-D. In the rest of this Section we briefly describe the results of the evaluation according to the different error criteria. Notice that the average execution time for each GPS sample is about $0.05 \sim 0.1$ seconds in normal situations and about $0.2 \sim 0.5$ seconds when there is a deviation. This shows that the mechanism is well-suited to be run locally, even on mobile devices. Given its lightweight nature, the algorithm can be implemented and deployed on a high scale (e.g., on smartphones), thus distributing the computational load across many devices.

a) RMSPE: The result of RMSPE for each city is reported in Table II. Two groups of entries were considered to perform the evaluation using this criterion: rides that followed the directions exactly and those that had minor deviations from the planned path.

	Milan	**Tehran**
Followed Direction	13.01 %	16.21 %
Minor Deviation	12.47 %	23.9 %

TABLE II: RMSPE Report.

As expected, Milan has considerably lower error since Tehran is much more crowded and has a more chaotic traffic flow. Also, time estimations for paths that show a minor deviation are usually less accurate, since the behavior of the driver is less predictable after a deviation—though indeed this was not the case in our data set for the rides collected in Milan.

Figure 6 shows the distribution of RMSPE for the two considered cities. More precisely, about 80% of the errors are in the 5%-20% range for Tehran and in the 5%-15% range for Milan. This shows that the results have reasonable variance, in addition to a good error average.

b) Error Margin: Figure 7 shows an example of error margin plot for a ride in Tehran. Most of the entries show characteristics similar to the plot of Figure 7. The algorithm produces estimations only some minutes after the start of the ride, so initially the curve is flat. The initial guess, which is shown as a jump in the plot, is usually already fairly close to the actual duration of the ride. The plot then shows the variation of the estimation. As the plot demonstrates, the estimation usually stays in the specified error margin after some specific time. However it is possible that in some areas the estimation exceeds the error margin due to harsh changes in the traffic flow.

c) Confidence Time: Figure 8 shows how the confidence time changes with the value of the error margin, for both cities. By definition of confidence time (see Section III-D), after T_{Conf} the estimation stays in the specified error margin relative to the actual value. For example, the purple dashed lines in Figure 8 show that we stay within a 12% error margin in Milan, and 22% error margin in Tehran

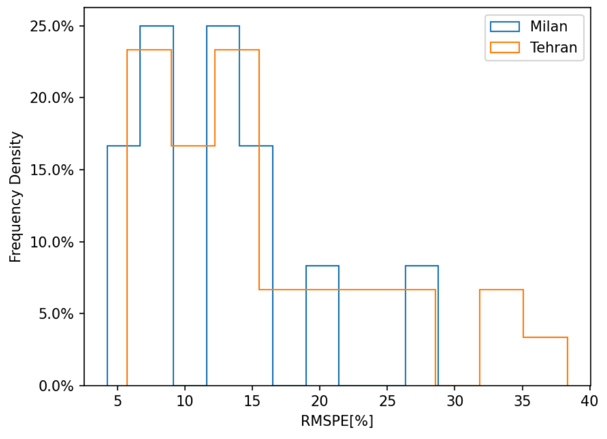

Fig. 6: RMSPE Distribution.

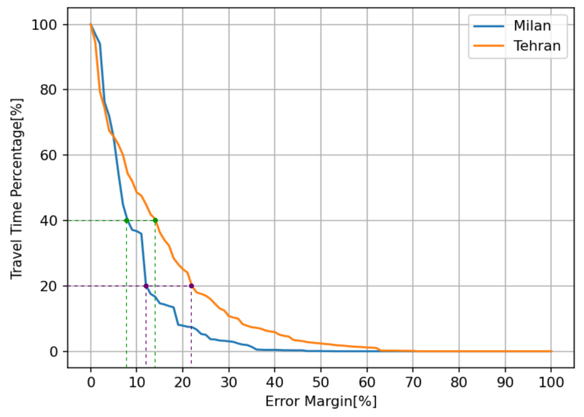

Fig. 8: Confidence time for Milan and Tehran.

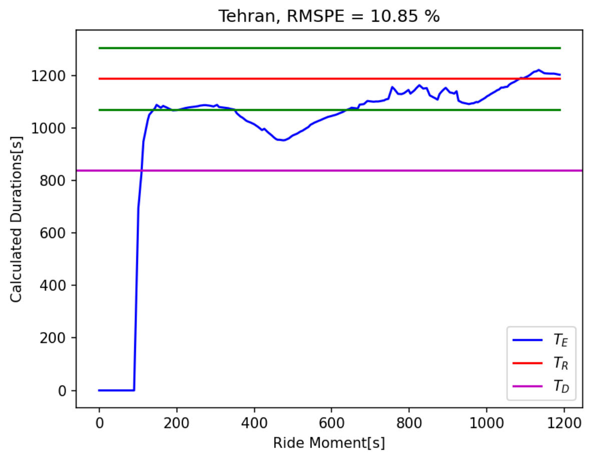

Fig. 7: Error margin plot.

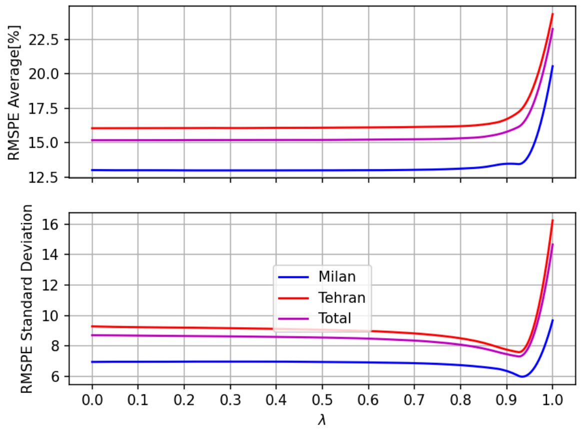

Fig. 9: λ analysis.

after about 20% of time T_D has passed. The error margins decrease to 8% for Milan and 14% in Tehran after 40% of T_D has passed (green dashed lines of Figure 8).

d) Parameter Evaluation: The presented algorithm relies on several parameters, and in particular on the λ value. Figure 9 shows how the average and standard deviation of the RMSPE change as the the value of λ changes. The plots show that the average error remains almost constant as the value of lambda increases, while the standard deviation decreases. The λ factor is designed to correct inaccurate estimations or to act in unpredictable situations. The collected data show that increasing the value of λ decreases the error of the samples that are most inaccurate and increases the error of those that are less inaccurate. That is, tuning the value of λ can provide more robust results with almost the same error. As depicted in Figure 10, we can identify a suitable criterion to choose the appropriate value of λ if we separate the samples that are negatively affected by an increase in λ from those which are instead positively affected. Indeed, Figure 10 shows that, for samples on which λ has a positive effect (i.e., for which the RMSPE decreases as λ increases), the minimum estimation error is reached for around $\lambda = 0.95$, which is also the value for which the standard deviation is lowest. Also, at that value, the increase in the RMSPE for samples that are negatively affected by λ is still rather small (the error is still below 15%). Our experience shows that this minimum point always exist. If the delay estimation mechanism is newly set up in a city and there is no previous data on which this analysis can be performed, the initial value of λ can be chosen based on its value for similar cities (in which the system is already running, hence the parameters have already been tuned). Then, the value of the parameter can be tuned after a initial batch of rides is performed. Notice also that the samples that are positively affected by higher values of λ are those with higher RMSPE (for example, in Teheran they have twice the RMSPE with respect to those that are negatively affected), which highlights how tuning the λ parameter helps make the estimations more uniform while maintaining a good overall precision.

Another parameter used in the delay estimation procedure is w, which indicates the size of the window of recent sam-

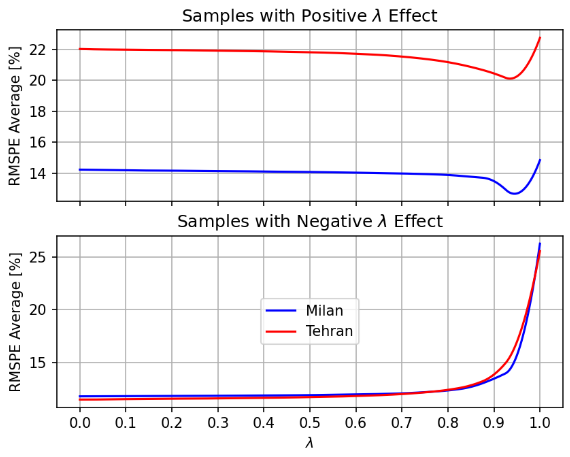

Fig. 10: λ analysis with data segmentation.

ples on which λ acts. The optimum value of this parameter is highly dependent on factors such as the length of the trip and the structure of the city roads. In our experiments we set the value of w to 120 (i.e., the last 10 minutes).

e) Discussion: As the results of the evaluation show, the proposed delay estimation mechanism reaches its desired goals. For example, for a 30 minutes ride, the average error would be about 4 minutes in Milan and 4.5 minutes in Tehran. The error can be further reduced if the predictions are produced after an initial delay, rather that right from the start of the ride, since it is at the beginning of the ride that the predictions are less reliable. In general, we deem the performance of the proposed delay estimation mechanism very promising in the context of urban ride sharing.

It must be noted, however, that the evaluation, although promising, has been carried out on a limited number of sample rides, and in two cities. We plan to further evaluate the proposed mechanisms on a larger data set in the future, to increase our confidence in its effectiveness.

As mentioned above, the computation of each new duration estimate takes only fractions of a second. Indeed, to keep the computation time low, our system avoids making repeated calls—which can be costly—to services providing estimates of the trip duration that possibly take into account traffic conditions. Instead, the direction API (for example, Google Maps®) is invoked only once, at the beginning of the trip, to gather an initial estimate (T_D) of the trip duration. If this initial estimate is accurate (i.e., T_D is very close to T_R), for example because it includes precise information about the traffic situation, the successive adjusted duration estimates will stay very close to the initial value. If, instead, it is considerably off-target (e.g., because the driver has an uncommon driving style, much faster or slower than the average, or because traffic conditions suddenly change), then the computed delay estimates will deviate significantly from it. In both cases the mechanism presented in this paper provides valuable information regarding the actual execution of the trip.

VII. Conclusions

This paper presented a new trip time/delay estimation mechanism that can be used to detect if a disruption impacts a shared ride and prevents it from being on time. The procedure produces an initial estimate of the trip time, which is then adjusted as the trip proceeds, based on GPS data samples collected during the ride. The mechanism relies on several parameters, which are fine-tuned depending on the monitored context (e.g., the city in which the rides occur). The approach has been evaluated on real-world experimental data, with promising results.

The approach can be improved in several ways. As discussed in Section VI, the values of the parameters on which the delay estimation mechanism relies (e.g., λ and w) depend on the monitored context. In the future, we plan to define a procedure to automatically adjust and tune the values of the parameters based on a training data set. We also plan to improve the accuracy of the estimation procedure, for example by considering different day/hour ratios for different areas of cities (e.g., city center, peripheral areas). Finally, the mechanisms presented in this paper will be integrated into the RIDE2RAIL ecosystem [9] to live-track the progress of vehicles during the execution of shared rides. The deployment architecture of the trip-tracking subsystem is structured on a set of independent modules, which perfectly fits the stateless nature of the components described in this paper; this allows the system to scale even in presence of heavy loads, depending on the number of final users.

References

[1] K. Lee, A. Prokhorchuk, J. Dauwels, and P. Jaillet, “Estimation of travel time from taxi gps data,” in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2017, pp. 1–6.

[2] M. Rahmani, E. Jenelius, and H. N. Koutsopoulos, “Route travel time estimation using low-frequency floating car data,” in *16th international ieee conference on intelligent transportation systems (itsc 2013)*. IEEE, 2013, pp. 2292–2297.

[3] ——, “Non-parametric estimation of route travel time distributions from low-frequency floating car data,” *Transportation Research Part C: Emerging Technologies*, vol. 58, pp. 343–362, 2015.

[4] E. Jenelius and H. N. Koutsopoulos, “Urban network travel time prediction based on a probabilistic principal component analysis model of probe data,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 2, pp. 436–445, 2017.

[5] M. Xu, K. Guo, J. Fang, and Z. Chen, “Utilizing artificial neural network in gps-equipped probe vehicles data-based travel time estimation,” *IEEE Access*, vol. 7, pp. 89 412–89 426, 2019.

[6] Y. Ishizaki, T. Sasama, T. Kawamura, and K. Sugahara, “Determining location of bus and path planning considering bus delay,” in *Proceedings of SICE Annual Conference 2010*. IEEE, 2010, pp. 2436–2437.

[7] C. Yang and G. Gidofalvi, “Fast map matching, an algorithm integrating hidden markov model with precomputation,” *International Journal of Geographical Information Science*, vol. 32, no. 3, pp. 547–570, 2018. [Online]. Available: https://doi.org/10.1080/13658816.2017.1400548

[8] T. Fomby, “Scoring measures for prediction problems,” *Department of Economics, Southern Methodist University, Dallas, TX*, 2008.

[9] RIDE2RAIL Consortium, “D3.3 – crowd-based travel expert service,” available from https://ride2rail.eu/wp-content/uploads/2022/07/RIDE2RAIL-D3.3-Crowd-based-Travel-Expert-Service.pdf, 2022, accessed: 14/07/2022.