UTILIZING HISTORICAL AND CURRENT TRAVEL TIMES BASED ON FLOATING CAR DATA FOR MANAGEMENT OF AN EXPRESS TRUCK FLEET

Rüdiger Ebendt, Alexander Sohr, Louis Calvin Touko Tcheumadjeu, Peter Wagner[1]

Summary: During the last nine years, a couple of prototype ITS applications based on Floating Car Data (FCD) of taxi fleets have been developed at German Aerospace Center (DLR). A core application is a route guidance and monitoring system based on current and historical road segment travel times. Recently, it has been extended for use in the German funded project SmartTruck, run by a consortium consisting of the logistics key player DHL, DLR and the German Research Center for Artificial Intelligence (DFKI). An important aim of the project was the use of historical and current traffic information for energy-efficient, optimized offline planning and dynamic re-planning of the tours of DHL express trucks in Berlin, Germany.

This paper discusses the architecture of the SmartTruck system and the methodology used to generate historic and current road segment travel times from positional data.

Key words: Floating Car Data (FCD), Global Positioning System (GPS), traffic monitoring, map-matching, traffic guidance, traffic planning.

1 Introduction

In logistics, a great challenge arises from the need for harmonized strategies to achieve two major objectives: on the one hand, customers expect forwarding companies to accomplish their service at a high level of flexibility and adherence to delivery dates. On the other hand, there is the need to optimize the vehicle

[1] German Aerospace Center, Institute of Transportation Systems, Department of Traffic Management, Rutherfordstraße 2, 12489 Berlin, Germany, tel.: +49 (0)30 67055-0, E-mail: {Ruediger.Ebendt, Alexander.Sohr, Louis.ToukoTcheumadjeu, Peter.Wagner}@dlr.de

routes for a high utilization and a resource-saving and energy-efficient execution of the transports.

Previous transportation planning systems worked with daily lists of pick-up and delivery orders and a management by ad-hoc plans and static planning schemes. This approach is based on the assumption of optimal traffic conditions and does not take jams or construction sites into account. There is no reaction to congestion or accidents and a re-planning process (if any) is only started in case of a dynamical change of the current pick-up orders or the arrival of new orders.

In this paper, the basic architecture and important parts of the methodology of a novel transportation planning system are described. It has been successfully deployed during a pilot project called *SmartTruck*, which was accomplished by a consortium consisting of the logistics key player DHL [4,5], the German Aerospace Center (DLR) [8] and the German Research Center for Artificial Intelligence (DFKI) [9]. Besides the classical input such as pick-up and delivery orders and given transportation restrictions, the system also collects vehicle data. This includes the GPS positions of the trucks and its current load, determined by the use of Radio Frequency Identification (RFID) antennas, an RFID reader inside the vehicle, and RFID parcel labels. It also gathers real-time traffic data by use of Floating Car Data (FCD). The FCD are obtained from taxi fleets operating in Berlin, Germany, and from the trucks themselves. Current and historical road segment travel times are determined by a system that has been developed at DLR [8] and has been enhanced and extended for use within the project. The delivery of FCD to the SmartTruck system has been deployed by DLR as a webservice interface. The travel times enable the system to detect congestion and to react to the changed traffic situation by calculating optimized, resource-saving alternative routes. The resulting re-planning process aims at increasing the vehicle load and at avoiding empty trips, thus decreasing the environmental impact.

This paper presents the methodology used to generate the respective current traffic situation as well as the historical travel times for every road segment. This task first includes the matching of the raw positional data onto the road segments of a given digital road map (known as “map-matching”). Therefore a stream of input trajectories (i.e., sequences of GPS positions) is entered into a map-matching algorithm. The GPS positions are affected by errors and noise (caused, e.g., by clouding or multi path signals), known as the *measurement error*. For each position in a trajectory, map matching needs to decide to whom of several eligible candidate segments the current position should be projected onto. It also needs to handle the *sampling error*: depending on the signal frequency, the vehicle may

also pass one or more intermediate segments between every two such reported positions. The lower the frequency, the more passed segments may exist between any such two points, and map-matching needs to determine these segments.

This paper describes the map-matching algorithm used within the project SmartTruck, which is based on routing by an integration of several multi-source variants of Dijkstra's algorithm [6]. A mechanism is described which allows to always select the most appropriate variant in terms of required solution quality and efficiency of the calculation. Next, the paper gives the details of the methodology used to compute historical travel times for every road segment. Instead of a simple calculation of average travel times for every such segment, a more sophisticated method is used. It takes into account the general tendency that a particular road segment has shown in the past.

The structure of the paper is as follows: after Section 1, this introduction, an overview and a brief description of the components of the SmartTruck system is given in Section 2. Section 3 describes the details of the map-matching and routing algorithm. Section 4 introduces the method used to compute historical travel times for every road segment. Section 5 briefly describes how to generate the respective current traffic situation. Finally, in Section 6, the work is concluded with the results obtained.

2 System Architecture

The starting point of the transportation planning process is the download of the pick-up and delivery data for the current day. The next steps are conducted by software components provided by DHL's technology partner infoware [12]: this includes the geo-coding of delivery addresses, the transferal of the geo-positions to the routing server, and the calculation of the transit times between approximately 3000 addresses in Berlin. This is accomplished by use of different travel time matrices for the respective times of the day (thereby distinguishing the rush-hour traffic from that during off-peak times). The travel times are historical travel times computed and periodically updated for every road segment at DLR. The approach that has been chosen for their calculation is described in detail in Section 4.

The transit times are then passed over to the Pick-up and Delivery route planning server which is provided by DHL's technology partner Quintiq [15]. They serve as the basis for the sophisticated optimization process, assigning the addresses to the vehicles in optimized stop sequences.

In contrast to previous planning approaches, the system is capable of considering the current traffic situation. Dynamical re-planning adapts the initial schedule to avoid routes affected by congestion and identified construction sites. The re-planning process is initiated in response to a significant change in the traffic condition: DLR continuously obtains position reports from around 500 taxis in Berlin and computes current travel times for every road segment. They are reported to the traffic server. Whenever a significant discrepancy to the respective historical travel times (which have been transmitted to the server by DLR before) is detected, the traffic server reports this to the route planning server. If appropriate, the route planning server decides to initiate the recalculation of the transit time matrices. In this case the recalculation is based on the current travel times that just have been reported, i.e. they override the historical travel times. Fig. 1 depicts an overview of the architecture of the SmartTruck system with the outlined Dynamic Routing and Dispatch Application (DRADA) as its core.

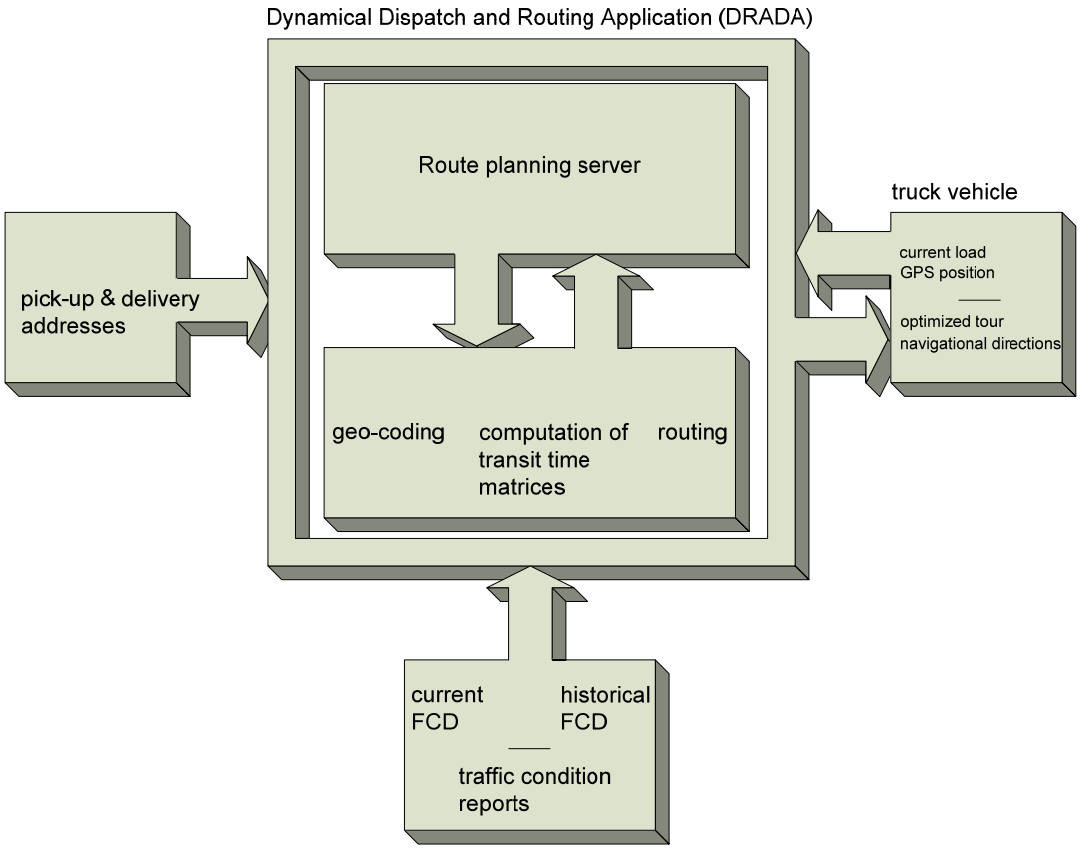

Fig. 1: architecture of the SmartTruck system

The trucks are equipped with on-board units with an integrated navigation system. Once the optimized routes for the day have been transferred by the route planning server over wireless links, it gives directions to the drivers. By this, the times for familiarization of drivers on routes that are new to them can be reduced dramatically. Each parcel has an RFID label. After each pick-up and delivery stop, the information about the current truck load is updated via RFID antennas and an RFID reader inside the truck. Additionally, each driver is equipped with a mobile device with an integrated scanner. Via the on-board unit and wireless links, the updated load status is reported back to the planning server. The on-board computers are supplied by Motorola [13].

Besides the outlined system structure, there are also application servers that implement additional services like e.g. an online address correction and a service that sends SMS messages for pick-up to customers. They are coordinated via an intelligent process management server which is is provided by DFKI.

3 Map-Matching and Routing

The map-matching problem is that of relating vehicle tracking data (i.e., trajectories) to the road network. It has to deal with the sampling error as well as with the measurement error (see Section 1). To tackle the problem of the sampling error, i.e. of unknown intermediate segments passed during the vehicle's transit between two position reports, the approach followed here uses a *routing* algorithm. By this it also ensures that the computed vehicle path is consistent with topological and other constraints due to the particular connection of road segments. For example, the path cannot jump from a highway to a local road unless a suitable connection between the two, such as an exit ramp, exists and, for this reason, a respective routing can be found.

The approach followed here combines an *incremental* with a *global* strategy. First, an incremental position-by-position, edge-by-edge strategy is applied forwards: a trajectory is represented by an (ordered) sequence of transits between a pair of positions subsequently reported (i.e., the "from-position" and the "to-position"). Perpendiculars are dropped to all candidate segments that are within a certain distance (the "matching radius") to the first position of a given pair of position reports. Each candidate segment is split into two auxiliary segments at the foot of perpendicular, where an auxiliary vertex is established. Each auxiliary vertex serves as starting and end point of the augmented auxiliary segments, respectively, and is an eligible starting point of a routing for the transit between the given two reported positions (i.e., a "from-node"). Next, the same is done for the

second reported position. The resulting auxiliary vertices are the eligible end points for the desired routing (i.e., the "to-nodes").

Then, for the two sets of eligible starting and end points, multi-source variants of Dijkstra's algorithm [6] are used to determine a set of candidate routings for the given transit. This problem reduces to SPSP (single-pair shortest path) which is solved by solutions to the classical SSSP (single-source shortest-path) problem. Thus, a fast method for SSSP is invoked subsequently on the consecutive transits: the to-position of the previous transit becomes the from-position of the next transit, the to-nodes become the next from-nodes. The best paths to them are preserved. Their path costs are passed over to the next routing as starting costs for the new from-nodes. By this it is guaranteed that the routing process remains aiming at finding a minimum cost path for the whole trajectory.

Second, a backwards global strategy follows that combines the routings for the subsequent transits to a globally optimal routing for the whole trajectory. Instead of a fixed-depth look-ahead or a limited back-tracking to evaluate the quality of different paths (e.g. see [3]), the method minimizes over all possible paths in the road network that are matching a given trajectory. By construction, it suffices to simply backward-chain the minimum cost paths, starting at the best end-node of the last routing, iterating the backward chaining until it ends at one of the eligible start nodes of the first transit of the trajectory.

To address the problem of measurement errors, various quality and error measures can be incorporated into the cost function (e.g., the weak Fréchet distance, e.g. [1]). To account for the sampling error, the cost function also targets at shortest/fastest paths between the reported positions. The routing algorithm also makes use of historic travel speeds as obtained by off-line map-matching of historic trajectory data (for the details, see Section 4). For on-line map-matching, the trajectory develops in real time and is not available at once. Therefore, a small number k of recently received position reports and the corresponding unconfirmed matches are maintained in a buffer. They can be used to decide whether an earlier match should be altered, based on the newly arrived track points. In this, matched trajectory data can be passed on to other parts of the FCD processing long before the trajectory is available completely, thus reducing the experienced delay to k reporting intervals only.

Of note is that all data are filtered for bad GPS signals and implausible values in the data sources and the calculated results. This includes the exclusion of non-

representative data. These occur when taxis are allowed to use special bus lanes or the taxis are picking up / dropping off a passenger (see also [16]).

A component of the map-matcher dispatches the routing request for a transit to one of several available Dijkstra-variants. The dispatcher's choice is based on the air distance between the from-position and the to-position. The average air distance again depends on the frequency of position reports. The frequencies can differ significantly: e.g. for the express trucks in Berlin and the taxi fleets in Hamburg the system gets a position report every 30 seconds, whereas there is only one report every two minutes for some of the taxi fleets in Berlin.

The dispatcher speeds up the routing process at zero or a very small loss in quality: for Hamburg, Germany, the method with the largest average loss, a novel integration of weighted A* of Pohl [14] into the algorithm A_ε of Ghallab and Allard [10], has an average loss in quality of less than four ‰. The available variants differ in the base algorithm (plain Dijkstra-algorithm [6], A* [11], weighted A* or the novel modified A_ε) or in the implementation of the priority queue of open nodes which is based on linked lists or on a Fibonacci-heap [7].

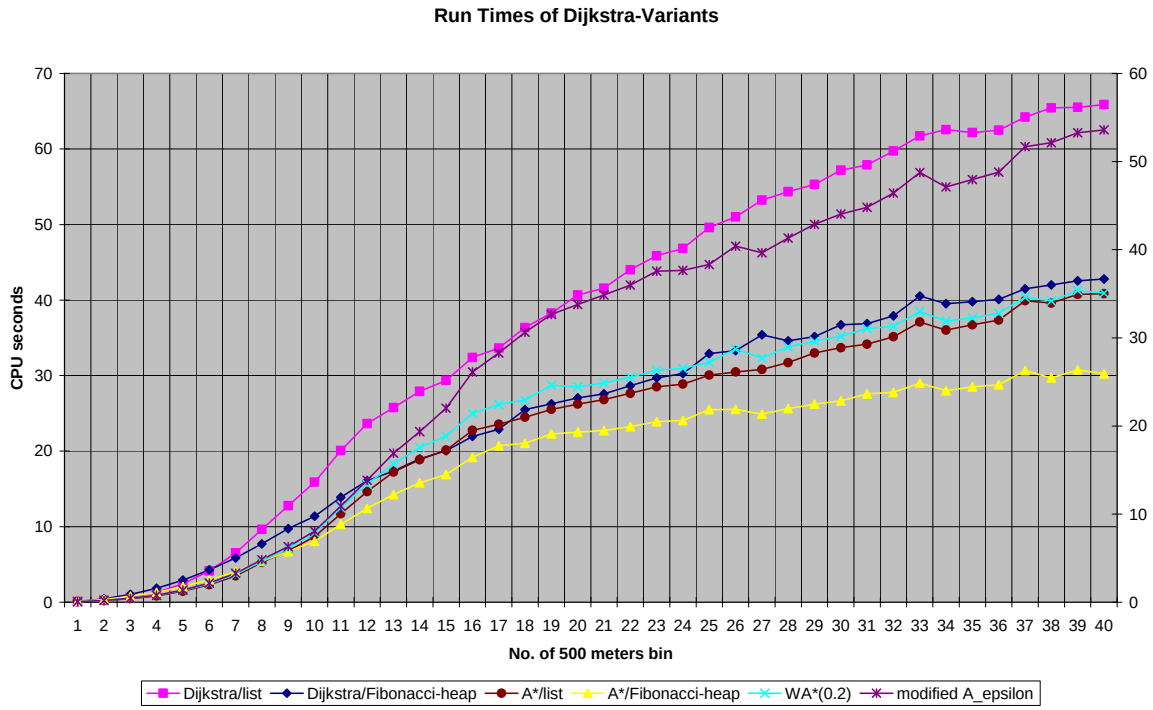

Fig. 2: run times of several Dijkstra-variants for longer distances

Figure 2 shows the average run time[1] of the respective Dijkstra-variations when applied to randomly generated origin-destination pairs in Hamburg. The pairs have been binned with respect to their air distance, one bin for every 500 meters, and such that 10.000 pairs resided in every bin from 0 to 20 km.

When looking at the longer distances, the implementations using a Fibonacci-heap outperform the list-based ones, and A* outperforms plain Dijkstra. Weighted A* (applying a weight of $1+\varepsilon=1.2$) performs very fast, despite its list-based implementation of the priority queue used here, and list-based A_ε, while faster than list-based Dijkstra, is not outperforming the other variations. Summarized, the results for the longer distances are not surprising.

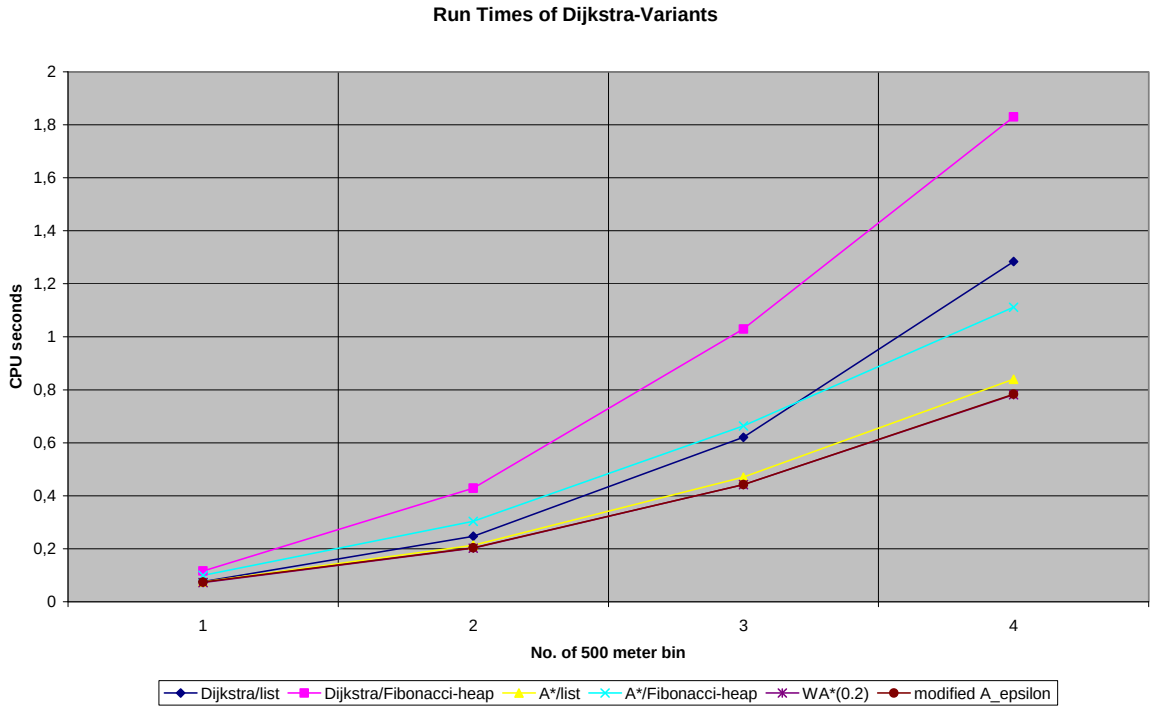

Fig. 3: run times of several Dijkstra-variants for smaller distances

The benefit from the list-based methods and A_ε becomes apparent when looking to the shorter distances. As Figure 3 shows for distances from 0 to 2 km, the list-based methods are significantly faster since the Fibonacci-heap has a high

[1] *The experiments were conducted on a machine with an Intel® Core™2 Duo CPU running at 2.4 GHz*

overhead cost. Modified A_ε performs as good as weighted A*. A_ε performs depth-oriented search as long as it can be guaranteed that the resulting solution cost will not deviate more than by a factor of $1+\varepsilon$ from the minimum solution cost (ε was set to 0.2 in the exeriments). This strategy turned out to be most effective when working with short distances. The trends observed in this experiment with purely synthetic origin-destination pairs can also be observed in reality: as a result of the different signal frequencies, map-matching by A* using a Fibonacci-heap as priority queue (using Pohl's weighted variant, whenever the small loss in quality is negligible) dominates in Berlin, whereas in Hamburg the modified A_ε method even outperforms weighted A* and yields the smallest run times. In Berlin, the outlined dispatching strategy reduces run time of routing by approximately 30%.

4 Obtaining Historic Travel Times from Floating Car Data

In Section 3, the approach chosen to match the GPS position reports to the digital road map has been described. Thereby, sampling errors are encountered: the vehicle may pass one or more intermediate segments between every pair of reported GPS positions. As a consequence, instead of knowing the travel time for a particular segment, usually only the transit time between two reported positions (measured as the lag between the time stamps of subsequent position reports) is known. However, this often is the accumulated travel time for a whole sequence of segments passed by the observed vehicle (the lower the signal frequency, the larger the length of this sequence will tend to grow). Instead of simply passing on the total travel time of such a sequence to its segments (e.g., with respect to their respective segment lengths), an iterative, self-adapting approach is used that takes into account the general tendency that a particular road segment has shown in the past.

Let us assume for a moment, that historical travel speeds/times are already known for every road segment. Since the travel time for a passed road segment, t_{segment}, is not given, it must be estimated. A first idea would be to pass on t_{transit} to the segments passed by the vehicle during a transit with respect to the individual segment length, i.e.

$$\tilde{t}_{\text{segment}} = \frac{l_{\text{segment}}}{v_{\text{transit}}} \tag{1}$$

Here, the average transit speed $v_{\text{transit}} = l_{\text{transit}} / t_{\text{transit}}$. However, the fact that road segments do not only differ with respect to their length but also with respect to their shape, number and width of lanes and parameters of road geometry and construction, is not taken into consideration by Eq. (1). All these factors have an effect on the historical travel time profile of a particular road segment. Therefore, a different formula is used. Let $\hat{v}_{\text{segment}}$ denote the historical segment travel speed and let $\hat{t}_{\text{segment}}$ denote the historical segment travel time. Then

$$t_{\text{segment}} = \left(\frac{l_{\text{segment}}}{\hat{v}_{\text{segment}}} \right) \cdot w_{\text{segment}} \tag{2}$$

where we define w_{segment} as

$$w_{\text{segment}} = \frac{t_{\text{segment}}}{\hat{t}_{\text{segment}}} = \frac{\hat{v}_{\text{segment}}}{v_{\text{segment}}} \tag{3}$$

The weight w_{segment} expresses the ratio of the currently observed travel time and the travel time that would be expected by the observations of the past. Since the weight w_{segment} depends on t_{segment} (or, alternatively, on v_{segment}), i.e., on exactly the unknown quantity that is subject to estimation, the weight w_{segment} must be estimated. For this purpose, the idea of the weight as ratio of current and expected travel time is transferred to the set of segments passed by the vehicle during a transit:

$$w_{\text{transit}} = \frac{t_{\text{transit}}}{\sum_{s \in \text{transit}} \hat{t}_s} \tag{4}$$

Next, w_{transit} is used as an estimator for w_{segment}. This simplification is justified as long as a vehicle driving e.g. twice as fast as would be expected from

history, does this more or less constantly during the whole transit. The final estimation used for the segment travel time then is

$$\hat{t}_{\text{segment}} = \left(\frac{l_{\text{segment}}}{\hat{v}_{\text{segment}}} \right) \cdot w_{\text{transit}} \tag{5}$$

The historical segment travel speeds are computed for every hour of the week (i.e., $7 \cdot 24 = 168$ speed values are computed for every road segment). To accomplish this, GPS trajectories collected during a longer period (e.g. several months) are map-matched (see Section 3). Statistics are computed based upon the estimated segment travel times, each computed with respect to the transit between a pair of consecutive position reports. The following definitions are needed for a formula describing the computation of these statistics: let $\text{transits}(i, j)$ denote the set of transits that pass road segment j within week hour i $(1 \le i \le 168)$. Let $\text{t}_j(t)$ be the travel time for road segment j, estimated as described in Eq. (1) to Eq. (5) with respect to transit t. Further, let $\text{l}_j(t)$ denote the length of the section travelled on segment j during transit t. Finally, $\hat{\text{v}}(i, j)$ denotes the historical travel speed on segment j within week hour i $(1 \le i \le 168)$. Then $\hat{\text{v}}(i, j)$ is computed as

$$\hat{\text{v}}(i, j) = \frac{\sum_{1 \le i \le 168} \sum_{j} \sum_{t \in \text{transits}(i, j)} \text{l}_j(t)}{\sum_{1 \le i \le 168} \sum_{j} \sum_{t \in \text{transits}(i, j)} \text{t}_j(t)} \tag{6}$$

This strategy is based on the assumption that historical travel speeds/times are already given for every road segment. Requiring them in advance is essentially a “hen and egg” problem. As a remedy, the static average speed attribute provided for every segment in digital road maps is used in a first iteration. As soon as a first set of travel times is computed with Eq. (1) to (5), the statistics expressed in Eq. (6) can be based on them to compute the first “real” set of historical travel speeds. The iteration then restarts to improve the current set of historical values and it is terminated as soon as the relative changes in the historical speed values stay below a pre-defined threshold. In practice, this iteration converged to historical speed values that were amenable to a successful validation [2] after only two rounds.

5 Generating a Current Traffic Situation based on Floating Car Data

Section 4 already gave the methodology of calculating estimated travel times for every road segment. The availability of historic travel speeds for every edge based on e.g. the trajectory data of the last few months facilitates the calculation of realistic current travel times. Of course, current travel speeds can be obtained by $v_{\text{segment}} = l_{\text{segment}} / t_{\text{segment}}$.

There is a second aspect where historic travel speeds are important for the calculation of current travel times or speeds: when determining the current travel speeds with FCD, this is done on the basis of the position reports which come in asynchronously. Thus a mechanism must be in place which modifies the previous speed value in response to a new speed value based on the arriving position reports. The methodology for this mechanism includes *aging* of the speed values determined in the recent past: until newly arriving position reports help to „refresh“ the older information about the traffic state on a given road segment, an aging formula will continuously modify the previously determined speed value such that it approaches its historic speed value as time passes by. This is particularly important on side roads where the number of taxi or truck vehicles travelling often is much lower than on arterial roads. Consequently, speed values must be subject to aging for a longer period in time.

After measurement activities with test drivers in Hamburg, Germany, the measured travel speeds have been compared to the current travel speeds obtained by the FCD system. As a result, the current travel speeds generated by the FCD system could be successfully validated [2].

6 Conclusion

We gave the architecture and the basic methodology of a novel transportation planning system for a fleet of express trucks. It has been successfully deployed in Berlin, Germany. The system gathers real-time traffic data by use of Floating Car Data (FCD), as obtained from the express trucks and from taxi fleets. The derived current and historical road segment travel times enable the system to detect congestion and to react to the changed traffic situation by calculating optimized, resource-saving alternative routes.

Reference literature

1. ALT, H.; EFRAT, A.; ROTE, G.; WENK, C. Matching planar maps. Journal of Algorithms 49, 2003, pp. 262–283.
2. BROCKFELD, E.; WAGNER, P.; PASSFELD, B. Validating travel times calculated on the basis of taxi floating car data with test drives. ITS World Congress, 2007, Beijing, China.
3. CHAWATHE, S.S. Segment-based map matching. Proceedings of the IEEE Intelligent Vehicles Symposium, 2007, Istanbul, Turkey, pp. 13–15.
4. DHL Innovation Center, http://www.dhl-innovation.de
5. DHL International GmbH, http://www.dhl.com
6. DIJKSTRA, E.W. A note on two problems in connexion with graphs. Numerische Mathematik 1, 1959, pp. 269–271.
7. FREDMAN, M.L.; TARJAN, R.E. Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM 34, 1987, pp. 596–615.
8. German Aerospace Center (DLR), http://www.dlr.de
9. German Research Center for Artificial Intelligence (DFKI), http://www.dfki.de
10. GHALLAB, M.; ALLARD, D.G. A_{ε}: An efficient near admissible heuristic search algorithm. Proc. Int. Joint Conf. on Artificial Intelligence, 1983, Karlsruhe, Germany, pp. 789–791.
11. HART, P.; NIILSSON, N.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. 2, 1968, pp. 100–107.
12. infoware GmbH, http://www.infoware.de
13. Motorola GmbH, http://www.motorola.com
14. POHL, I. Heuristic search viewed as path finding in a graph. Artificial Intelligence 1 (3), 1970, pp. 193–204.
15. Quintiq B.V., http://www.quintiq.com
16. SCHÄFER, R.-P.; THIESSENHUSEN, K.-U.; BROCKFELD, E.; WAGNER, P. A traffic information system by means of real-time floating-car data. ITS World Congress, 2002, Chicago, USA.