De l'(in)utilité du temps-réel pour le calcul d'itinéraire dans les réseaux routiers

Amine Mohamed Falek, Antoine Gallais, Cristel Pelsser, Sébastien Julien, Fabrice Theoleyre

De l'(in)utilité du temps-réel pour le calcul d'itinéraire dans les réseaux routiers

Amine Mohamed Falek[1,2] Antoine Gallais[2] Cristel Pelsser[2]
Sebastien Julien[1] Fabrice Theoleyre[2]

[1] *Technology & Strategy group, 4 rue de Dublin, 67300 Schiltigheim, France*
[2] *ICube Lab, CNRS / University of Strasbourg, Pole API, Boulevard Sebastien Brant, 67412 Illkirch Cedex, France*

La planification d'itinéraire est devenue un défi majeur avec un impact significatif sur l'économie, la sécurité, et le climat. Elle consiste à fournir à chaque utilisateur une route présentant le plus faible temps de parcours, même si les conditions de circulation évoluent. Ainsi, une telle stratégie requiert de reconsidérer la route à prendre en continu, les conditions évoluant. Cependant, prendre en compte ces données temps-réel présente un impact élevé sur les ressources en calcul nécessaires. Nous quantifions donc ici le gain apporté par des données temps-réel. Nous comparons les routes obtenues à l'aide de données statistiques, versus temps-réel. Nous fournissons également une borne inférieure du temps de trajet, avec un algorithme qui serait capable de prédire parfaitement le futur. Nos résultats qui s'appuient sur un jeu de données réelles montrent de façon surprenante que le temps-réel est en réalité peu utile.

Mots-clefs : planification d'itinéraire; réseau routier; temps-réel; dynamique, prédiction

1 Introduction

Road network traffic congestion is a well known worldwide cause of anger and frustration, with a growing climatologic and economic impact. Thereby, route planning algorithms are heavily relied upon to improve traffic conditions by computing for each user, an optimal itinerary i.e. an itinerary minimizing travel time.

In practice, road networks are dynamic: travel time is time-variant as roads may suddenly be congested or even closed. In such conditions, computing optimal itineraries become significantly more challenging. In this paper, we investigate the benefits of real-time traffic data for route planning.

Since congestion is time-variant, a vehicle may change its route after having left its point of attachment. However, re-computing the best route for each crossroad is computationally expensive.

Similarly, exploiting only static travel times for each road segment allows implementing a preprocessing approach, reducing the computational cost [H. 16].

In this paper, we propose to model each route planning strategy, from the less agnostic (static data) to the most accurate (real-time) approach. We also propose an algorithm serving as a lower bound, able to identify the best route, by exploiting an ideal prediction on the travel time for each road segment. We exploit a real dataset in several cities (e.g. London, NYC) to evaluate quantitatively the gain of using real-time data.

2 Route planning strategies with static vs. real-time data

We use a dynamic directed graph $G_T = (V, E, W_T)$ to represent the road network, where $v \in V$ is a vertex representing a physical intersection of two or more road segments, $e_{ij} \in E$ a directed edge from vertex i to j and $w_{ij}(t) \in W_T$ the weight assigned to e_{ij}, as a function of time $t \in T$.

We use the term dynamic to refer to a graph whose structure remains the same (no edge deletion/insertion) but the weights assigned to each edge change over the time period T. For each route planning strategy, given a query, the goal is to compute a route with minimal travel time. Formally, we define a query as $q = (s, d, t_s) | s, d \in V$ and $t_s \in T$, where s, d and t_s represent the starting point, the destination and the departure time respectively. Likewise, a route $R = \{s, u, .., w, d\}$ is defined as an ordered list of vertices.

To solve a query, we first compute a route R using a given route planning algorithm alg, then, we compute the actual travel time $TTime[q, alg]$ by emulating a moving vehicle along route R. Here, are the 4 algorithms that we use to evaluate the need for real-time data in route planning:

Static Route Planning: We assume that the system has no knowledge of current traffic conditions. Thus, it uses the speed limit of each road segment to compute the fastest route to its destination. Such a strategy might be used in the case of a mobile navigation system with no access to the Internet, and cannot retrieve real-time data. We use Dijkstra's algorithm to compute the route with the shortest travel time.

No re-Routing Route Planning: With this strategy, the system has a precise knowledge of the travel time of each road segment in the network before leaving its departure location. The route planning decision is made only once, right before the departure. Consider a cloud-based infrastructure which adopts a connection-less strategy. The device receives a route to follow, and no subsequent tracking is implemented. Thus, for a given query $q = (s, d, t_s)$, we use Dijkstra's algorithm with the graph weights $w_i(t_s) \in W_T$ set at departure time.

Continuous re-Routing Route Planning: Continuous re-routing consists of continuously using the most up-to-date information. After a vehicle has left its departure point, the route can be updated if faster alternative routes exist. We use a dynamic version of Dijkstra's algorithm taking into consideration that edge weights are updated at regular time intervals Δt. Given a query $q = (s, d, t_s)$, when an update occurs at time t, we compute the edge e_{uv} where the vehicle is currently located and issue a new query $q' = (v, d, t)$ to update the route from the upcoming crossroad v.

Ideal Predictions Route Planning: This algorithm is one of the contributions of the paper. Given a query $q = (s, d, t_s)$, we achieve an *ideal* prediction by departing a vehicle in the past and *replaying* the recorded measurements *a posteriori*. Thereby, all updates occurring at $t > t_s$ are known in advance. We present a time-dependent version of Dijkstra's algorithm where the weights W_T are updated at regular intervals of Δt. During the traversal of the graph, we evaluate the travel time $TTime[e_{uv}]$ of an edge e_{uv} by emulating a vehicle along e_{uv} and denote its position by $P(t \geq t_u)$.

From the set of already processed vertices, we retrieve the triplet $[u, t_u, t_{update}(u)]$ where t_u is the smallest arrival time to reach vertex u from s and $t_{update}(u)$ is the remaining time before a new update occurs when located at vertex u. Then, for each of u's neighboring vertices $v \in Neighbors(u)$, we compute the travel time $TTime[e_{uv}] = dist_{uv}/speed_{uv}(t_u)$ along edge e_{uv}:

1. If $TTime[e_{uv}] \leq t_{update}(u)$ i.e. the vehicle traverses e_{uv} before an update occurs, we compute at v, the remaining time before the next update given by $t_{update}(v) = t_{update}(u) - TTime[e_{uv}]$.

2. If $TTime[e_{uv}] > t_{update}(u)$ i.e. the update occurs before reaching v, we temporarily set e_{uv} travel time to $TTime[e_{uv}] = t_{update}(u)$ and compute the new position of the vehicle $P(t = t_u + t_{update}(u))$. From there, we recursively compute the time required to reach vertex v by computing the remaining distance to v and resetting the time for a new update to occur to Δt.

Finally, we compute $t_v = t_u + TTime[e_{uv}]$, push the triplet $[v, t_v, t_{update}(v)]$ into the set of processed vertices and repeat the whole process until vertex d is reached.

Proof of correctness: We must prove that the prediction-based algorithm, does indeed compute an *ideal* route for a given query $q = (s, d, t_s)$ i.e a route with minimal travel time when departing vertex s at time t_s.

The FIFO property, also called the non-overtaking property states that during the traversal of an edge e_{uv}, a vehicle A leaving u at time t should always reach v before another vehicle B that departs from u at $t' > t$. Computing shortest paths on a graph that fulfills the FIFO property is polynomially solvable [KS93] while shown to be NP-hard otherwise [OR90]. In our implementation, when computing the travel time of an edge e_{uv}, we do not freeze its weight at t_u but rather update it during its traversal. Hence, the FIFO property is maintained, that is, a delayed departure from vertex u would never lead to an earlier arrival at v.

Let us now use induction to prove that the algorithm does indeed, compute the shortest route for a given query q. In figure 2, let us assume that the shortest route to each vertex within the settled set was correctly

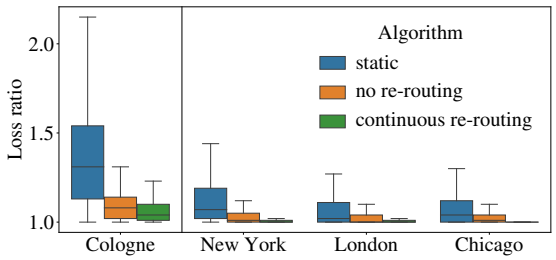


Fig. 1: Loss ratio of each routing strategy on the simulated (left) and the real (right) datasets.

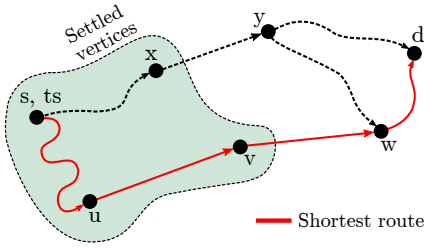


Fig. 2: Correctness of the ideal predictions- based strategy.

identified. Supposing that the algorithm chooses to settle vertex w next, our goal is to prove that the route $R = \{s,..,u,v,w\}$ is indeed the shortest route to vertex w:

1. the base case is satisfied as s is settled during initialization with $t_s = t_s$ (departure time);
2. since w is to be settled next, then by definition $TTime[R] = t_v + TTime[e_{vw}] \leq t_x + TTime[e_{xy}]$;
3. Hence, for any route $R' = \{s,..,x,y,..,w\}$ other than R, we know that $TTime[R'] = t_x + TTime[e_{xy}] + TTime[\{y,..,w\}]$;
4. Thus, $TTime[R] < TTime[R']$, proving that the algorithm is correct as t_w is indeed the shortest time to reach w from s.

3 Evaluation of the need for real-time data

Using all four route planning strategies, we first measured the distribution of travel times for a large set of queries on both the simulated and real datasets. We only measured significant variability in travel time among the different strategies during rush hours at (6:00-9:00) and (14:00-18:00). Hence, in all upcoming subsections, all the experiments are solely conducted during rush hours when routing strategies yield different results. Running all our experiments required more than 50,000 computational hours that we distributed among the 380 nodes of our HPC.

Datasets: We first exploit a **Real Travel Time Dataset**, obtained via HERE's API (`https://www.here.com/`). It corresponds to actual travel times for a large collection of road segments (New York, London, and Chicago), sampled every 1 min, and collected over 3 months.

Since a few road segments are not monitored by HERE, we also rely on the well-known **Simulated Dataset (TAPAS trace)**. [UTCFBO14]. It focuses on a small German City (Cologne), during a weekday (24 hours). OpenStreetMap was used to construct the road network and SUMO to simulate traffic using a large set of emulated vehicles.

Impact of Route Planning Strategy on Travel Time: We compare the travel time achieved by each strategy compared with the ideal one (which provides by construction the lower bound). The loss ratio for a query $q = (s,d,t_s)$ is defined as: $L[q] = \frac{TTime[q,alg]}{TTime[q,ideal]}$, where $TTime[q,alg]$ denotes the travel time for the query q using the algorithm alg.

By construction, the loss ratio is larger than 1.0 (Fig. 1). In the simulated dataset (Cologne), the travel times are significantly higher compared to the real dataset. Simulation tends to exacerbate the variability, where redirections significantly increase travel time.

The static algorithm does not exploit real-time data and provides the highest travel times. However, the no-rerouting strategy (only using data right before the vehicle leaves its point of attachment) computes the shortest travel time in most situations. The difference is only significant for a few queries (usually long itineraries). Finally, the continuous re-routing strategy, which identifies the best instantaneous route at each crossroad, provides quasi-optimal travel times: prediction seems useless here.

Finally, we measured the impact of the sampling rate, and verified that the route planning strategy can accommodate even low rates (up to 5 min). This limits the computation cost without impacting accuracy.

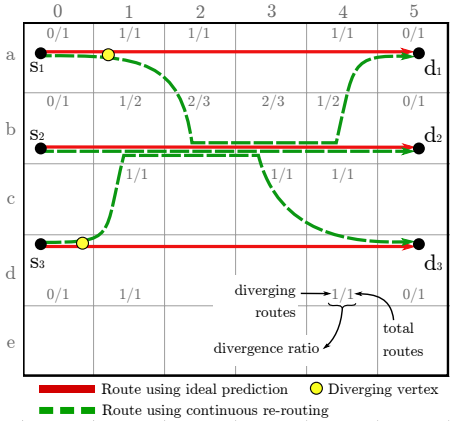
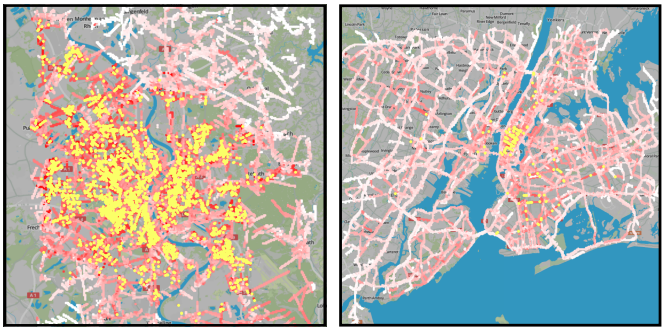


Fig. 3: Computing the divergence ratio



(a) Cologne (simulated) (b) New York (real dataset)

Fig. 4: Diverging vertices (yellow dots) and divergence ratio (red shade) on the simulated (a) and the real (b) datasets.

Divergence Patterns: We now try to identify the diverging vertices in the maps, i.e. where optimal and continuous re-routing routes diverge. Intuitively, they denote the points where a re-computation should be triggered. Additionally, we try to discover the regions in the map where traffic is typically diverged through.

For this purpose, we create a geographic regular grid. Then, we count for each cell the ratio of routes passing through this cell while diverging (Fig. 3). Typically, the ideal (in red) and the continuous re-routing (dashed green) routes begin to diverge in cell (a,1). Hence, we introduce a diverging vertex at that location (yellow dot). Moreover, we measure for each cell traversed by a green route its divergence ratio. Typically, cell (b,2) has a divergence ratio of $2/3$ as only 2 routes diverge among the 3 green routes.

Figure 4 illustrates the divergence using a heat map, also identifying the diverging vertices with a divergence ratio ≥ 0.5 (yellow dots). The simulations exhibit a very high number of diverging vertices, and routes diverge uniformly in the city. On the contrary, we can identify few specific diverging locations for all of NYC, London and Chicago in the real dataset (only NYC is shown in figure 4b). Only the road segments in red exhibit a very dynamic pattern. Hence, the continuous re-routing strategy could significantly be improved to reduce execution time by recomputing a route only when encountering a diverging vertex.

4 Conclusion and Future Work

By knowing the actual speed for each road segment, intelligent systems guide the vehicles through the fastest routes. However, exploiting this real-time data has a cost: the route computation has to be re-executed continuously until the vehicle reaches the destination. Similarly, prediction-based methods may be inaccurate and computationally expensive. In this paper, we asked whether real-time data is important for planning routes with near-optimal travel time. We compared quantitatively the end-to-end travel times obtained through different route planning strategies, using measured and simulated datasets. To reduce the computational cost, we have also identified a set of diverging points in each road network.

In future work, we plan to enhance the use of real-time data. In particular, we are investigating automatic methods to identify the most relevant instants and locations where should be re-computed.

References

[H. 16] H. Bast, et al. *Algorithm engineering*, chapter Route Planning in Transportation Networks, pages 19–80. 2016.

[KS93] David E Kaufman and Robert L Smith. Fastest paths in time-dependent networks for intelligent vehicle-highway systems application. *Journal of Intelligent Transportation Systems*, 1(1):1–11, 1993.

[OR90] Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM (JACM)*, 37(3):607–625, 1990.

[UTCFBO14] Sandesh Uppoor, Oscar Trullols-Cruces, Marco Fiore, and Jose M Barcelo-Ordinas. Generation and analysis of a large-scale urban vehicular mobility dataset. *IEEE Transactions on Mobile Computing*, 13(5):1061–1075, 2014.