Proactive Vehicular Traffic Rerouting for Lower Travel Time

Susan Juan Pan, Iulian Sandu Popa, Karine Zeitouni, Cristian Borcea

Proactive Vehicular Traffic Re-routing for Lower Travel Time

Juan (Susan) Pan, Iulian Sandu Popa, Karine Zeitouni, and Cristian Borcea

***Abstract*—Traffic congestion causes driver frustration and costs billions of dollars annually in lost time and fuel consumption. This paper presents five traffic re-routing strategies designed to be incorporated in a cost-effective and easily deployable vehicular traffic guidance system that reduces travel time. The proposed strategies proactively compute individually-tailored re-routing guidance to be pushed to vehicles when signs of congestion are observed on their route. The five proposed strategies are Dynamic Shortest Path (DSP), A* shortest path with repulsion (AR*), Random k-Shortest Paths (RkSP), Entropy Balanced k-Shortest Paths (EBkSP), and Flow Balanced k-Shortest Paths (FBkSP). Extensive simulation results show the proposed strategies are capable of reducing the travel time as much as a state-of-the-art Dynamic Traffic Assignment (DTA) algorithm, while avoiding the issues that make DTA impractical such as lack of scalability and robustness, and high computation time. Furthermore, the variety of proposed strategies allows tuning the system to different levels of trade-off between re-routing effectiveness and computational efficiency. Also, the proposed traffic guidance system can significantly improve the traffic even if many drivers ignore the guidance or if the system adoption rate is relatively low.**

***Index Terms*—proactive driver guidance, traffic load balancing, vehicular congestion avoidance, vehicular networks.**

I. Introduction

Despite significant advances of in-car navigation systems (e.g., Garmin, TomTom), web services for route computation (e.g., Google, Microsoft), and dynamic traffic assignment [9], [24], we are still spending a lot of time in traffic jams. It is predicted that by 2015, the congestion cost will rise to $133 billion and the amount of wasted fuel will jump to 2.5 billion gallons [34]. Hence, finding effective solutions for congestion mitigation at reasonable costs is becoming a stringent problem.

With the deployment of traffic surveillance infrastructure on more roads (e.g., loop detectors, video cameras), we have started to witness web-based services/applications that present the drivers with the current view of the traffic and let them decide which route to follow. However, the usefulness of these applications is limited: (i)They have mostly accurate information about the highways, and thus are not very useful for city traffic; and (ii) they cannot prevent congestions and, at the same time, it is known that no true equilibrium can be found under congestion [9].

Recently, companies such as Google [1] and TomTom [2] have started to use infrastructure-based traffic information to compute traffic-aware shortest routes. However, these solutions do not try to prevent congestions explicitly (i.e., they are reactive solutions) and provide the same guidance for all vehicles on the road at a certain moment as function of their destination (i.e., pull model in which drivers query for the shortest route to destination). Therefore, similar to route oscillations in computer networks, they could lead to unstable global traffic behavior: when it happens, congestion is switched from one route to another if a significant number of drivers use the guidance.

This situation could be avoided by new solutions on dynamic user-optimal traffic assignment [9]. These solutions periodically compute the assignment of traffic flows to routes that lead to user equilibrium. Unfortunately, there is still a significant gap between the theoretical or simulation results and potentially deployable solutions. Some issues are: tractability for large scale road networks, capability of providing real-time guidance, behavior in the presence of congestion, ability to work when not all drivers are part of the system, and robustness to drivers who ignore the guidance.

The time is ripe for building a proactive, intelligent, and real-time traffic guidance system based on the dynamic situations on the road network. In this system, vehicles can be viewed as both mobile sensors (i.e., collect real-time traffic data) and actuators (i.e., change their path in response to newly received guidance). The system is cost-effective and easily deployable because it does not require road-side infrastructure; it can work using only smart phones carried by drivers [1]. Where road-side sensors are available, the system can take advantage of them to supplement the data provided by vehicles to build an accurate representation of the global real-time traffic conditions. When signs of congestions are observed on certain road segments, it computes proactive, individually-tailored re-routing guidance, which is pushed to vehicles that would pass through the congested segments.

Manuscript received October 1, 2012; revised February 12, 2013; accepted April 13, 2013. This work was supported in part by KISS, a research project funded by French ANR Call INS 2011, and the National Science Foundation under Grant CNS-0831753. The review of this paper was coordinated by Dr. P. Lin.

J. Pan and C. Borcea are with the Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102-1982 USA (e-mail: jp238@njit.edu; borcea@njit.edu).

I. S. Popa is with the Department of Computer Science, University of Versailles Saint-Quentin-en-Yvelines, Versailles 78000, France, and also with Inria Paris-Rocquencourt, Le Chesnay 78145, France (e-mail: iulian. sandu-popa@prism.uvsq.fr).

K. Zeitouni is with the Department of Computer Science, University of Versailles Saint-Quentin-en-Yvelines, Versailles 78000, France (e-mail: karine.zeitouni@prism.uvsq.fr).

Digital Object Identifier 10.1109/TVT.2013.2260422

[1] In the future, once vehicular embedded systems become widespread, they could be used instead of smart phones.

This paper introduces such a cost-effective and easily deployable vehicular traffic guidance system that reduces the effect of traffic congestions. Then, evaluates five re-routing strategies designed to be incorporated in this system: 1) Dynamic Shortest Path (DSP), which assigns to each vehicle the current shortest time path to destination; 2) A* shortest path with Repulsion (AR*), which modifies the A* shortest path algorithm [17] by considering both the travel time and the paths of the other vehicles (as a repulsive force) in the computation of the shortest path; 3) Random k Shortest Paths (RkSP), which computes kSPs [2] for each re-routed vehicle and randomly assigns the vehicle to one of them; iv) Entropy Balanced kSP (EBkSP), which computes kSPs for each vehicle and assigns the vehicle to the path with the lowest popularity as defined by the path entropy; v) Flow Balanced kSP (FBkSP), which computes kSPs for each vehicle and assigns the vehicle to the path that minimizes the impact of traffic flow in a network region. Specifically, our contributions are as following:

- We propose five novel re-routing strategies and extensive evaluate them through various simulations settings over two medium-size urban road networks. All our strategies result in significantly lower average travel time compared to a "no re-routing" baseline. Among the proposed strategies, AR* has the lowest average travel time, but the highest computation cost. EBkSP and FBkSP can achieve comparable travel times as AR*, while demanding lower CPU times for the re-routing computations.
- We employ a tool implementing a state-of-the-art DTA algorithm for traffic optimization to compare it against our strategies. Compared to DTA, our strategies obtain similar travel times at (much) lower computation costs. Besides, our strategies are much more scalable with the number of vehicles than DTA.
- We measure the robustness of the system by investigating compliance and penetration rate. The results show our strategies are still effective in alleviating congestion even if many drivers ignore the guidance or if the system adoption rate is relatively low, which is important in facilitating the adoption of the system at a large scale.

The rest of this paper is organized as follows. Section II discusses related work. Section III describes the system model and our assumptions. Section IV introduces the five re-routing strategies and the DTA algorithm used as baseline. Section V presents the experimental results and their analysis. We conclude and give directions for future work in Section VI.

II. Related Work

Projects such as Mobile Millennium [42], [20], CarTel [11], JamBayes [19], Nericell [30], and surface street estimation [16] use vehicle probe data collected from on-board GPS devices to reconstruct the state of traffic and estimate shortest travel time. The proposed research moves beyond this idea: instead of investigating the feasibility and accuracy of using mobile phones as traffic sensors, we are focusing on using that information to recommend routes more intelligently, thus, achieving better efficiency in terms of avoiding congestion and reducing travel time.

Services such as INRIX [3] provide real-time traffic information at a certain temporal accuracy, which allows drivers to choose alternative routes if they are showing lower travel times. Systems such as Google Maps and Microsoft's Bing are able to forecast congestion and its duration by performing advanced statistical predictive analysis of traffic patterns. Additionally, short term non-recurrent congestions can be predicted as well [4]. Based on such information, according to Wardrop's first traffic equilibrium principle [40], drivers could be able to reach a user-optimum traffic equilibrium. It is known, however, that no true equilibrium can be found under congestion [21]. Even more important, the usefulness of such services is limited by their reactive nature: they cannot prevent congestions. Our solution moves one step forward by providing effective methods for proactive re-routing when congestion is predicted based on real-time traffic information.

A large body of existing route planning research focuses on fast generation of (k)shortest paths [35], [26] in highly dynamic scenarios with frequent traffic information update. In particular, [35] presents transit-node routing and highway-node routing to reduce the average query time and memory requirements. The work in [26] proposes two new classes of approximation techniques that use pre-computation and avoidance of complete recalculations on every update to speed up the processing of continuous route planning queries. However, current instantaneous shortest paths are not necessarily equal to time-dependent shortest paths. These algorithms calculate shortest paths based only on the snapshot of current traffic conditions without considering the dynamic future conditions.

One of the essential properties of the road network is the time-dependency of the travel time. Computing shortest paths in a time varying spatial network is challenging since the edge (i.e., road segment) travel time changes dynamically. In this case, the computation not only considers the instantaneous travel time in one single snapshot of the traffic graph but also the relationship among the consecutive snapshots across time. George et at. [15] demonstrated a fast greedy time-dependent shortest path algorithm (SP-TAG) by using a Time Aggregated Graph (TAG) data structure instead of the time-expanded graph. SP-TAG saves storage and computation cost by allowing the properties of edges and nodes to be modeled as a time series instead of replicating nodes and edges at each time unit. While algorithms such as SP-TAG provide insights into the dynamics of traffic networks, two obstacles remain besides increased computational cost. First, it is impractical to assume the system knows the exact travel time series of every single road segment given the traffic dynamics. Second, these algorithms do not help with switching congestion from one spot to another if all the drivers are provided the same time-dependent shortest path.

An alternative to our work could be the research done on dynamic traffic assignment (DTA) which leads to either system-optimal or user-optimal route assignments. DTA research can be classified into two categories: analytical methods and simulation-based models. Analytical models such as [13],

[2] We use kSPs for short for k shortest paths in the remainder of this paper.

[29], [28] formulate DTA as either nonlinear programming problems, optimal control problems, or variational inequalities. Although they provide theoretical insights, the computational intractability prevents their deployment in real systems [32].

Simulation-based approaches [9], [39], [25], [14] have gained greater acceptability in recent years, in which the time-dependent user equilibrium is computed by iterative simulations. The simulations are used to model the theoretical insights that cannot be derived from analytical approaches. This process computes the assignment of traffic flows until the travel times of all drivers are stationary. Unfortunately, there are still a number of issues associated with these approaches that make their deployment difficult: tractability for large scale road networks given the computational burden associated with the simulator, capability of providing real-time guidance, effectiveness in the presence of congestion, and behavior of drivers who ignore the guidance. For example, they assume the set of Origin-Destination (OD) pairs and the traffic rate between every OD pair are known. This information is highly dynamic especially in city scenarios, leading to frequent iterations of computationally expensive algorithms even when not needed from a driver benefit point of view. Additionally, the OD set is large, and the DTA algorithms may not be able to compute the equilibrium fast enough to inform the vehicles about their new routes in time to avoid congestions. Our system, on the other hand, is designed to be effective and fast, although not optimal, in deciding which vehicles should be re-routed when signs of congestion occur as well as computing alternative routes for these vehicles.

The complexity of DTA systems has led scientists to look for inspiration in Biology and Internet protocols. In [36], Wedde et al. developed a road traffic routing protocol, BeeJamA, based on honey bee behavior. Similarly, Tatomir et al. [38] proposed a route guidance system based on trail-laying ability of ants. Inspired by the well-known Internet routing protocols, Prothmann et al. [33] proposed decentralized Organic Traffic Control. However, since they employ ad hoc networking, these approaches have only a partial view of the traffic conditions, which may lead to less accurate re-routing. Also, simply treating vehicles as packets which always listen to the guidance ignores the nature of human behavior. Furthermore, these systems react to real-time data without insight into future conditions, thus introducing greater vulnerability to switching congestion from one spot to another.

III. System Model

Our traffic guidance system is composed of: (1) a centralized traffic monitoring and re-routing service (which can physically be distributed across several servers), and (2) a vehicle software stack for periodic traffic data reporting (position, speed, direction) and showing alternative routes to drivers. Vehicles run this software either on a smart phone or an embedded vehicular system. Vehicles are equipped with GPS receivers and can communicate with the service over the Internet when needed. When starting a trip, each vehicle informs the service of its current position and destination; the service sends back a route computed according to its strategy. It is assumed that the service knows the road network as well as the capacity and legal speed limits on all roads.

Logically, the traffic guidance system operates in four phases executed periodically: (1) data collection and representation; (2) traffic congestion prediction; (3) vehicle selection for re-routing; and (4) alternative route assignment for each such vehicle and pushing the guidance to the vehicles. Since data collection has been studied extensively in the literature, we do not address this issue and just assume that the centralized service receives traffic data from vehicles and road-side sensors where available. We discuss in detail each of the other phases in this section and Section IV.

A. Traffic data representation and estimation

The road network is represented as a directed, weighted graph, where nodes correspond to intersections, edges to road segments, and weights to estimated travel times. The weights are updated periodically as new traffic data becomes available. Several methods can be employed to estimate the travel time over a road segment. For instance, using vehicle probe data collected from on-board GPS devices to reconstruct the state of traffic is a well-studied topic [42], [24]. We use the Greenshield's model [6] to estimate the travel time since it is used extensively in dynamic traffic assignment models by transportation researchers. The model considers that there is a linear relationship between the estimated road speed V_i and the traffic density K_i (vehicles per meter) on road segment i, as in Equation 1:

$$V_i = V_f(1 - \frac{K_i}{K_{jam}}) \qquad T_i = L_i/V_i \tag{1}$$

where K_{jam} and V_f are the traffic jam density and the free flow speed for road segment i, while T_i and L_i are the estimated travel time and length for the same segment. The free flow speed V_f is defined as the average speed at which a motorist would travel if there were no congestion or other adverse conditions. To simplify our implementation, we consider that the free flow speed is the road speed limit. Basically, K_i/K_{jam} is the ratio between the $current\_number\_of\_vehicles$ and the $max\_number\_of\_vehicles$. The $current\_number\_of\_vehicles$ is obtained from the traffic data collected by the service, whereas the $max\_number\_of\_vehicles = length\_of\_road/(avg\_vehicle\_length + min\_gap)$.

B. Congestion prediction

Periodically, the service checks the road network to detect signs of congestion. A road segment is considered to exhibit congestion signs when $K_i/K_{jam} > \delta$, where $\delta \in [0, 1]$ is a predefined threshold value. Choosing the right value for δ is particularly important for the service performance. If it is too low, the service could trigger unnecessary re-routing; this may lead to an increase in the drivers' travel times. If it is too high, the re-routing process could be triggered too late and congestion will not be avoided. The evaluation in section V-B confirms these hypotheses.

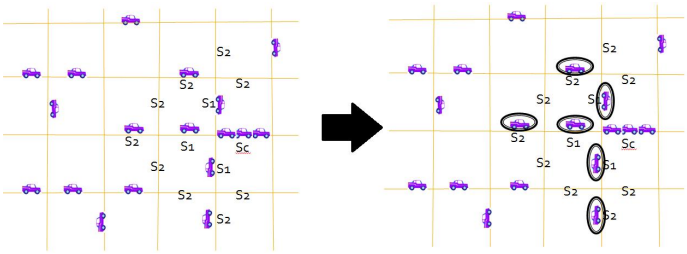


Fig. 1: The vehicle selection process

C. Selection of vehicles to be re-routed

When a certain road segment presents signs of congestion, the service looks for nearby vehicles to re-route. Specifically, we select vehicles from incoming segments (i.e., segments which bring traffic into the congested one). To decide how far from congestion to look for candidates for re-routing, the service uses a parameter L (level). This parameter denotes the furthest distance (in number of segments) a candidate vehicle can be away from the congested segment. In practice, L could be computed as function of the severity of congestion; for example, we can use the "level of service" (LOS) defined in the Highway Capacity Manual [27]. L's value has to be large enough to mitigate congestion. If L is too high, however, more vehicles than necessary will be selected for re-routing, which can have undesired consequences (e.g., creating congestion in another spot). Since our focus is on the re-routing algorithms and the analysis of their performance, we decided to consider L a tuning parameter that is varied during our experiments. We plan to investigate an efficient way for selecting its value in the presence of dynamic traffic conditions in future work.

The service performs a breadth first search (BFS) on the inverted network graph (i.e., the road network graph is directed), starting from the congested segments with maximum depth L and considers all these cars as candidates for re-routing. The process is illustrated in Figure 1. Assuming L=2 and signs of congestion are detected on the segment Sc, the system recursively selects the vehicles situated on the incoming segments of the congested segment in two steps. First, the vehicles located on segments S1 are included in the candidate set, followed by the vehicles situated on segments S2.

We select the union of all the vehicles affected by all the congested segments in the whole network within the same level, then perform vehicle ranking (as described in Section III-D), path computations, and path assignments (in Section IV).

D. Ranking the selected vehicles

The selected vehicles need to be ranked and assigned to alternative paths according to their rank for all strategies, except DSP. In this way, the performance of the strategies improve. The impact a congested road segment has on a vehicle's travel time is different depending on the remaining distance to the vehicle's destination. Intuitively, the drivers that are close to their arrival point may have a different perception of the congestion than the drivers that are far away from their destination. Our system uses an urgency function to rank the vehicles that are selected for re-routing. Hence, the vehicles with higher urgency are re-routed first and get relatively better routes.

Definition 1. *Given a set of vehicles $V = (v_1, v_2, v_3, ..., v_m)$ to be re-routed, we define two urgency functions to compute the re-routing priority of a vehicle in V:*

- *Absolute Congestion Impact: ACI=$RemTT$-$RFFTT$*
- *Relative Congestion Impact: RCI=($RemTT$-$RFFTT$)/$RFFTT$*

where $RemTT$ is the remaining travel time, and $RFFTT$ is the remaining free flow travel time for the vehicle.

ACI measures the impact of all the (congested) segments of the remaining journey of a vehicle, since ($RemTT$-$RFFTT$) is the absolute increase of the travel time with respect to the free flow travel time. With ACI, the longer the remaining distance to the destination for a vehicle is, the higher is the probability for that vehicle to get a higher rank (as the difference $RemTT$-$RFFTT$ normally increases with the $RFFTT$). On the other hand, RCI weighs the congestion impact on a vehicle relative to its remaining travel time. Hence, RCI gives a higher priority to vehicles that are close to their destination. Since the further the vehicle is from its the destination, the higher are the number of alternative paths and the potential benefit of re-routing, we expect ACI to perform better than RCI in our system.

IV. Re-routing Strategies

Recent research has proved that real-time traffic flow data and road travel time can be determined based on data reported by vehicles or road-side sensors [19], [42], [30]. The question is how to utilize this knowledge in an intelligent fashion to avoid congestion and reduce the drivers' travel times. This section presents five re-routing strategies that we classify in two categories. The first, presented in Section IV-A, includes two re-routing strategies that compute a single, alternative new path for each re-routed vehicle. The strategies are based on the well-known Dijkstra algorithm and on the A* algorithm with a modified heuristic, respectively. Section IV-B presents the second category consisting of three re-routing strategies that compute multiple, alternative new paths for each of the re-routed vehicles. Then, different heuristics are used to choose the best alternative path to be assigned to a vehicle. Based on the five proposed strategies, we describe the main re-routing process executed by the traffic guidance system in Section IV-C. Finally, Section IV-D presents a Dynamic Traffic Assignment strategy [14] that we use as a baseline to measure the effectiveness and efficiency of the proposed strategies.

A. Single Shortest Path Strategies

1) Dynamic Shortest Path (DSP): DSP is a classical re-routing strategy that assigns the selected vehicles to the path with lowest travel time. However, different from the existing systems, our system takes a proactive approach. Specifically, each time a road segment presents signs of congestion, the service obtains the set of cars whose paths intersect this road segment and computes for each car a new shortest

path based on the current travel time in the road network. Therefore, the path of each car can be periodically updated on an event-driven basis. The advantage of this strategy lays in its simplicity and consequently reasonable computational cost, i.e., $O(E + VlogV)$ [12], where E is the number of road segments and V is the number of intersections of the road network. We expect this strategy to provide good results when the number of re-routed vehicles is low, since in this case the risk of switching congestion from one spot to another is low. Hence, locally redirecting the traffic when congestion happens should be sufficient in this case. On the other hand, when the traffic density is higher, there is an increased risk of switching the congestion from one road to another. Moreover, the re-routing frequency for a driver is likely to increase in this case, which can be annoying to drivers.

2) A Shortest Path With Repulsion (AR*):* The DSP strategy only takes into account the current view of the traffic when performing re-routing, without considering the impact the re-routing will have on the future traffic. To address this limitation, we propose AR*, which modifies the A* search algorithm to include the prior re-routing decisions into the computation of the current shortest path. A* [17] uses a best-first search and a heuristic function to determine in which order to visit the network nodes (road intersections in our case). Given a node x, a heuristic function $F(x)$ is computed as the sum $G(x) + H(x)$. $G(x)$ is the path-cost from the start node to x, which corresponds to the travel time in our case, while $H(x)$ is a heuristic estimation of the remaining travel time from x to the destination node. In addition, $H(x)$ has to be "admissible" (i.e., it must not overestimate the remaining travel time to the destination) to produce the shortest path between the source and the destination. Therefore, $H(x)$ is computed as the Euclidean distance divided by the maximum speed in the road network.

Future congestion occurs if many drivers take the same road segment within the same future time window. [3] As we assume that the drivers share their route information with the service, it is possible to estimate the future footprint of each driver in the road network.

Definition 2. *A weighted footprint counter, fc_i, of a road segment i is defined as follows: $fc_i = n_i \times \omega_i$, where n_i is the total number of vehicles that are assigned to paths that include segment i, and ω_i is a weight associated with i. $\omega_i = \frac{len_{avg}}{len_i \times lane_i} \times \frac{Vf_{avg}}{Vf_i}$, where len_{avg} is the average road segment length in the network, Vf_{avg} is the average free flow speed of the network, len_i is the length of i, Vf_i is the free flow speed of i, and $lane_i$ is number of lanes of i.*

In the formula, n_i represents the discretized future traffic flow on road i. We decided to use weights in the formula in order to count for different road characteristics. For example, suppose there are two road segments r_i, r_j. Although n_i=n_j, the segments should not be treated equally since r_i has higher capacity (more lanes or longer length), thus the possibility of causing congestion is lower. In other words, the impact of the traffic flow n_i on road r_i is lower than n_j on r_j even though n_i=n_j.

In AR*, we modify the heuristic function $F(x)$ to include the other vehicles sharing the same path as a repulsive force. Specifically, we define the repulsive score $R(x)$ of a node x as the sum of the weighted footprint counters (cf. Definition 2) from the starting node to the node x. Thus, the path-cost function becomes $F(x) = (1-\beta)\times(G(x)+H(x))+\beta\times R(x)$, where $G(x)$ and $H(x)$ are computed as in the original algorithm and β is a weighting parameter. $G(x) + H(x)$ measures the travel time factor, while $R(x)$ reflects the impact of other vehicle traces on the examined path. Since the travel time and the repulsive force use different metrics, we normalize their values and compute $F(x)$ as a linear combination of the two factors. The parameter β allows a variable weighting between the travel time factor and the repulsive force factor. If β is too large, the repulsive force factor becomes predominant and the resulting path can be diverted too far away from the shortest path. Oppositely, if β is too low, AR* will behave similarly to the naive DSP strategy. In our experiments, we determine empirically the value of β that leads to the best effectiveness for the AR* algorithm.

The complete algorithm is presented as pseudo code in Algorithm 1. Starting from the initial node, the algorithm maintains a queue of nodes to be traversed, denoted as the open set (lines 3-5). At each iteration, the node with the lowest $Fscore$ value is removed from the queue (lines 16-19), the values of its neighbors are updated accordingly (lines 27-28), and these neighbors are added to the queue (line 30). The algorithm continues until the end node has been reached or until the queue is empty. The normalization of the travel time and the repulsive force factors is done at line 14. A path is returned at line 18 if found, otherwise an empty path is returned in line 48.

Figure 2 illustrates a simple example of how AR* is used in re-routing. We suppose that vehicles v_1, v_2, v_3 having the same origin and destination, i.e., from ab to ij, need to be re-routed and that $urgency(v_1) > urgency(v_2) > urgency(v_3)$. At the beginning, since no vehicle has been assigned any path, AR* performs normal A* search and assigns the shortest path ab, bc, cd, di, ij to vehicle v_1. When computing the shortest path for vehicle v_2, AR* will find ab, bg, gh, hi, ij. Although v_2 has the same destination as v_1, the path found by AR* is different since it considers the footprints produced by v_1 as a repulsion. Hence, AR* avoids the already assigned paths as much as possible, while still keeping the new path as short as possible. Finally, the procedure is repeated for vehicle v_3 and the path ab, bc, cd, di, ij is obtained for the same reasons.

Notice that AR* has to be employed by the re-routing system in an iterative manner. Namely, after the selected vehicles to be re-routed have been ranked based on their urgency, the system calculates sequentially each vehicle's route starting from the most urgent one. Therefore, in the case of AR*, the computation time increases linearly with the number of re-routed vehicles. On the other hand, as explained in the next sections, the rest of the proposed re-routing methods optimize this phase by grouping the vehicles to be re-routed based on their origin-destination, which leads to lower computational

[3]The time window size equals the period used by the system to evaluate congestion.

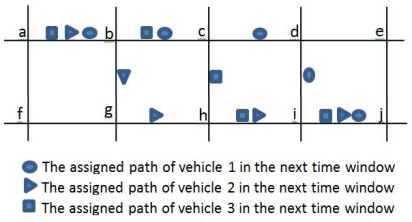


Fig. 2: AR^* re-routing example. All road segments have same weight and $\beta = 0.5$.

Algorithm 1 A Star Shortest Path with Repulsion Re-routing

```
1: procedure AstarRepulsion(start,end)
2: P[start]=empty {the reverse pointer of the path, which is used to re-construct the path}
3: closedset = set() {The set of nodes already evaluated}
4: openset = set()
5: openset.add(start) {The set of tentative nodes to be evaluated, initially containing the start node}
6: Gscore[start] = 0.0 {Travel time cost from start along best known path}
7: Hscore[start] = Euclidean(start, end)/maxspeed
8: Rscore[start] = 0.0
9: Fscore[start] = 1.0
10: while openset is not empty do
11:     sumF=SumFscore(openset)
12:     sumR=SumRscore(openset)
13:     for all node in openset do
14:         Fscore[node]=(1-β)*Fscore[node]/SumF + β*Rscore[node]/SumR
15:     end for
16:     current=getleastFscore(openset)
17:     if current==end then
18:         return (Fscore[current],P)
19:     end if
20:     openset.remove(current)
21:     closedset.add(current) {add current to closedset}
22:     for all edge in current.outEdges do
23:         node=edge.endnode
24:         if node in closedset then
25:             continue
26:         end if
27:         tentative_g_score=Gscore[current] + edge.actualtime
28:         tentative_r_score=Rscore[current] + edge.weight_footprints
29:         if node not in openset then
30:             openset.add(node)
31:             Hscore[node]=Euclidean(node, end)/maxspeed
32:             tentative_is_better=True
33:         else
34:             if tentative_g_score<Gscore[node] then
35:                 tentative_is_better=True
36:             else
37:                 tentative_is_better=False
38:             end if
39:         end if
40:         if tentative_is_better == True then
41:             P[node]=edge
42:             Gscore[node]=tentative_g_score
43:             Rscore[node]=tentative_r_score
44:             Fscore[node]=Gscore[node]+Hscore[node]
45:         end if
46:     end for
47: end while
48: return (0.0,{})
49: end procedure
```

complexity.

B. Multiple Shortest Paths Strategies

The two strategies proposed above compute a single path for each re-routed vehicle. However, the two methods have opposite behaviors. On the one hand, DSP sacrifices effectiveness (since it does not consider the impact of re-routing on the future traffic) to optimize the computational cost (by grouping the re-routed vehicles on their origin-destination). On the other hand, AR* trades efficiency (since it computes an alternative path for each vehicle) for effectiveness (by taking into account the future traffic configuration). In this section, we introduce a new class of re-routing strategies to obtain the best trade-off between efficiency and effectiveness. For these strategies, the re-routing process is divided into two steps. First, k loopless shortest paths are computed for each selected vehicle based on the travel time in the road network, where k is a predefined parameter. Compared to DSP this approach involves a higher computation time, but it still permits to group vehicles on their origin-destination. Therefore, we expect the computation time to be lower than AR*. Second, the vehicles are assigned to one of their k-shortest paths in the order of their ranking. Among the k-shortest paths, the algorithm generally selects the path the least employed by other vehicle traces. Hence, we expect this class of strategies to have an effectiveness similar to AR*. We propose three heuristics for the selection of the best path among the k-shortest paths.

1) Random k Shortest Paths (RkSP): RkSP assigns each selected vehicle to one of the k paths randomly. The goal is to avoid switching congestion from one spot to another by balancing the re-routed traffic among several paths. Compared to DSP, the price to pay is a higher computational complexity, $O(kV(E + VlogV))$ [23], which increases linearly with k. Although a larger k will allow better traffic balancing, it also increases the difference in the travel time among the k paths. Therefore, to prevent an excessive increase of the travel time for some drivers, RkSP limits the maximum allowed relative difference between the fastest and the slowest path to 20%.

2) Entropy Balanced k Shortest Paths (EBkSP): While RkSP addresses the main potential shortcoming of DSP (i.e., moving congestion to another spot), it has its own deficiencies. First, it increases the computational time, which matters because the alternative paths must be computed and pushed to vehicles before they pass the re-routing intersection. Second, it assigns paths randomly to vehicles, which is far from optimal both from a driver point of view and from the global traffic point of view. To address this second shortcoming of RkSP, we propose the EBkSP strategy. The idea is to perform a more intelligent path selection by considering the impact that each selection has on the future density of the affected road segments. The more intelligent path selection comes at the cost of a slightly increased complexity. However, we expect this optimization to improve the traffic from a global point of view. In addition, as in AR*, EBkSP ranks the cars to be re-routed based on an urgency function that quantifies the degree to which the congested road affects the driver travel time. Thus, the more affected vehicles will have priority and be re-routed first.

The entropy idea comes from Shannon information theory [37]. Several works [43], [10] have successfully applied it to compute the popularity of an area. To avoid creating new congestions through re-routing, we associate a "popularity" measure to road segments in EBkSP. We use entropy to define the popularity of a path as follows.

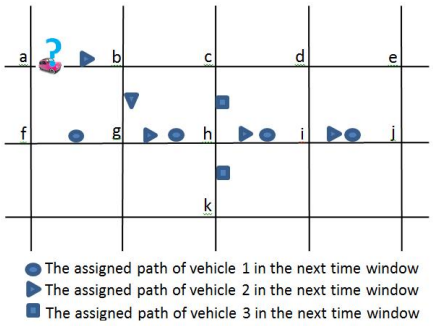


Fig. 3: A EBkSP re-routing example. All segments have same weight.

Definition 3. *Let* $(p_1, ..., p_k)$ *be the set of paths computed for the vehicle which will be assigned next. Let* $(r_1, ..., r_n)$ *be the union of all segments of* $(p_1, ..., p_k)$*, and let* $(fc_1, ..., fc_n)$ *be the set of weighted footprint counters associated with these segments. The popularity of* p_j *is defined as* $Pop(p_j) = e^{E(p_j)}$*.* $E(p_j)$ *is the weighted entropy of* p_j *and is computed as* $E(p_j) = -\sum_{i=1}^{n} \frac{fc_i}{N} \ln \frac{fc_i}{N}$*,* $N = \sum_{i=1}^{n} n_i$*.*

The value of $E(p_j)$ measures the probability that a number of vehicles will be on the path p_j in a time window. According to the above definition, we have $0 \leq Pop(p_j) \leq m$, where m is the number of vehicles. $Pop(p_j)$ has the maximum value m when every previously assigned vehicle traverses entirely p_j (i.e., they take the same path). $Pop(p_j)$ has the minimum value when no one takes the path p_j. Intuitively: the higher the popularity of a path, the higher the probability that more drivers will take this path.

After vehicle selection and ranking, we assign each vehicle to the least popular path among its k-shortest paths in order to avoid potential future congestions. Specifically, the first vehicle is assigned the current best path without considering others. Then, the road network footprints are updated based on the new path. When assigning the second vehicle, the popularity score of its k-shortest paths are calculated and the least popular path will be chosen. The process is then repeated for the rest of the re-routed vehicles.

Figure 3 illustrates an example of EBkSP re-routing. We assume that vehicles (v_1, v_2, v_3) have been initially assigned to their shortest time paths, and each road has the same weight ($\omega_i = 1$). Vehicles v_4 (identified by the question mark) arrives, and then EBkSP rerouting takes place. The footprints of (v_1, v_2, v_3) in the next time window are (fg, gh, hi, ij), (ab, bg, gh, hi, ij), and (ch, hk), respectively. For v_4, which travels from ab to ij, there are three alternative paths with similar travel times: $p_1(ab, bg, gh, hi, ij)$, $p_2(ab, bc, ch, hi, ij)$, and $p_3(ab, bc, cd, di, ij)$. The union of their segments is the set $(ab, bg, gh, hi, ij, bc, ch, cd, di)$, and their weighted footprint counters are (1,1,2,2,2,0,1,0,0). Consequently, N=11, $E_V(p_1)$=2.29, $E_V(p_2)$=1.67 and $E_V(p_3)$=0.53. Hence, v_4 will be assigned to p_3 because it is the least popular.

3) Flow Balanced k Shortest Paths (FBkSP): RkSP and EBkSP distribute the traffic load of the re-routed vehicles by randomly choosing between alternative paths or by balancing the system entropy among multiple paths. Since the key idea is load balancing, an alternative approach that we propose is

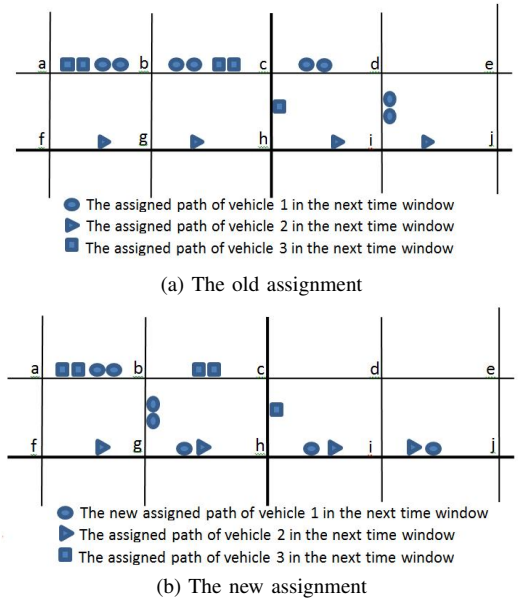


Fig. 4: A FBkSP example. $\omega_{fg} = \omega_{gh} = \omega_{hi} = \omega_{ij} = \omega_{ch} = 1$, $\omega_{ab} = \omega_{bc} = \omega_{cd} = \omega_{de} = \omega_{af} = \omega_{bg} = \omega_{di} = \omega_{ej} = 2$.

to directly balance the traffic load, i.e., the weighted footprint counters, through local search optimization [18]. The goal of the local "search" is to find the path assignment in which the sum of the weighted footprint counters is minimal, i.e., to minimize $\sum_{s_i \epsilon S} fc_{s_i}$ in a network region, where S is the set of all region segments. As we recall from Definition 2, weighted footprint counter fc_i indicates the impact of the traffic flow on road segment r_i (i.e., the possibility of generating future congestion on r_i). Therefore, the summation of the weighted footprints counters of all the road segments measures the risk of congestion of the whole network. In another words, as a weighted footprint counter indicates the future flow magnitude, minimizing the sum of the weighted footprint counters means having balanced flows on all paths, and thus, reducing the risk of producing congestion.

Figure 4 illustrates how the path assignment affects the total number of weighted footprint counters. Assume that initially the vehicles (v_1, v_2, v_3) are assigned to the paths (ab, bc, cd, di, ij), (fg, gh, hi, ij) and (ab, bc, ch), respectively, and that the road segments have different weights (cf. Figure 4). Then, the sum of the weighted footprint counters in this network region is 18 (cf. Figure 4a). However, if v_1 switches to the path (ab, bg, gh, hi, ij), the sum of the weighted footprint counters is reduced to 16 as shown in Figure 4b. Therefore, the system will select the latter assignment.

To implement the optimization of the total number of footprints in a road network region, we use a random search strategy (cf. Algorithm 2). The system generates first a good path assignment solution for all selected vehicles by assigning to each vehicle the path with the current least number of footprints (lines 2-8 in Algorithm 2). This initial assignment does not necessarily guarantee the minimum sum of footprint counters of the considered network region, i.e., the union of all segments of the k shortest paths of the re-routed vehicle. Therefore, the system randomly modifies the initial assignment in order to improve it (lines 14-16). If the new assignment

Algorithm 2 Flow Balanced k Shortest Path Re-routing

```
1: procedure LocalOptAssign(allkPaths, sortedVehicles)
   {generate initial solution}
2: for all vehicle in sortedVehicles do
3:     {origin, dest}=getVehicleOD(vehicle)
4:     newpath = pickPath_leastfootprints(allkPaths, origin, dest)
5:     reduction=getReduction()
6:     vehicle.selectedpath=newpath
7:     updateFootprint(vehicle)
8: end for
   {locally optimize the initial solution}
9: iter=0
10: repeat
11:     for all vehicle in sortedVehicles do
12:         {origin, dest}=getVehicleOD(vehicle)
13:         newpath=pickpath_random(allkpath,orgin,dest)
14:         newreduction=getReduction(newpath,vehicle.selectedpath)
15:         if newReduction<reduction then
16:             vehicle.selectedpath=newpath
17:             updateFootprint(vehicle)
18:             reduction=newReduction
19:         end if
20:     end for
21:     iter=iter+1
22: until iter>MaxIteration{MaxIteration is a constant, set as 10 here.}
23: end procedure
```

reduces the total number of weighted footprint counters in the network region, the new assignment is accepted (lines 18-19). Otherwise, the assignment is rejected. This process runs iteratively until the limit number of iterations is attained (line 26).

Disjointness of the k paths. The k-shortest paths (kSP) algorithm used in the paper computes a set of k shortest-time paths that are loopless but potentially overlapping. Using k disjoint shortest paths (or paths with a low degree of similarity) does not necessarily improve the re-routing performance of our algorithms. In order to compute k disjoint shortest paths, a typical algorithm computes first m shortest paths (m>k), and then selects the k disjoint paths from the set of m paths. Once the computation cost for determining the m paths is paid, EBkSP and FBkSP will perform better over m paths than over a subset of k paths because the total number of road segments that can be used for load balancing is larger. The experimental results presented in Section V-B confirm that increasing the k value improves significantly the effectiveness of the re-routing.

C. Re-routing Process

In this section, we present the global re-routing process on which our traffic guidance system is based on. The process is presented in Algorithm 3. The system periodically looks for signs of congestion in the road network (line 4). If signs of congestion are detected, then the system selects the vehicles situated near to the congested road segments (cf. Section III-C) and ranks them based on the urgency function (cf. Section III-D). Finally, alternative routes are computed for the selected vehicles by using one of the five proposed re-routing strategies. It is worth noticing that except AR^*, all the other re-routing strategies optimize the alternative path search by grouping the vehicles on their origin-destination (line 10). This can lead to a significant reduction of the computational cost as showed in Section V-B.

Algorithm 3 The main process

```
1: procedure main
2: while true do
3:     updateEdgeWeights()
4:     congestedRoads=detectCongestion(edgeWeights)
5:     if #congestedRoads>0 then
6:         for all road in congestionRoads do
7:             selectedVehicles=selectedVehicles ⋃ selectVehicles(road)
8:         end for
9:         sortedVehicles=sortByUrgency(selectedVehicles)
10:        allpaths=Emtpy
11:        if not AR* then
12:            odPairs=updateODPairs(selectedVehicles)
13:            if DSP then
14:                allPaths=Dijkstra(odPairs)
15:            else
16:                allPaths=compuate_all_kShortestPaths(odPairs)
17:            end if
18:            doReroute(allPaths, sortedVehicles)
19:        else
20:            for all vehicle in sortedVehicles do
21:                {origin, dest}=getVehicleOD(vehicle)
22:                newPath=AstarRepulsion(origin,dest)
23:                if newPath is not empty then
24:                    setRoute(vehicle, newPath)
25:                end if
26:            end for
27:        end if
28:    end if
29:    wait(period) {The process executes periodically.}
30: end while
31: end procedure

32: procedure doReroute(allPaths, sortedVehicles)
33: if FBkSP then
34:    LocalOptAssign(allPaths, sortedVehicles)
35: else
36:    for all vehicle in sortedVehicles do
37:        {origin, dest}=getVehicleOD(vehicle)
38:        if DSP then
39:            newPath = allPaths[origin][dest][0]
40:        end if
41:        if RkSP then
42:            newPath = pickPath_random(allPaths[origin][dest])
43:        end if
44:        if EBkSP then
45:            newPath = pickPath_leastPopular(allPaths[origin][dest])
46:            updateFootprint(vehicle, newPath)
47:        end if
48:        setRoute(vehicle, newPath)
49:    end for
50: end if
51: end procedure
```

D. Dynamic Traffic Assignment

The work on DTA algorithms is essential for the problem we consider in this paper, i.e., improving the driving travel time through traffic re-routing and guidance. Nevertheless, as explained in Section II, DTA is not yet the most viable solution for real-time traffic guidance, mainly because of the DTA's very high computational complexity coupled with the high dynamics of the traffic and the imperfections in traffic knowledge. In spite of this, DTA can offer valuable information as, for example, the level of improvement in the travel time that can be achieved in an ideal situation (i.e., where computational cost is not an issue and the traffic information is perfect). Therefore, we employ DTA to obtain a lower bound on the optimization of the travel time for comparison with the results produced by the proposed strategies.

The DTA model that we use in this paper tries to achieve

stochastic user equilibrium (SUE) through an iterative simulation process and mathematical modeling (see Section II). Given the traffic demand, it chooses some initial routes assuming zero traffic. Then, it calculates the network load and the travel times by simulation and updates the route choices of the drivers. This process is repeated until the travel times are stationary or a maximum number of iterations is reached. The simulation-based DTA tool we employ was proposed in [14], [5]. At least three parameters have to be given as input: a road network, a set of trips, and the maximum number of iterations. The higher the number of iterations is, the higher is the probability to achieve a SUE traffic state. In our experiments, we defined the maximum number of iterations to 50, since that was the value specified in [8]. The DTA algorithm, as defined in [14], is summarized next:

Step 1:Initialize the route of each driver by the optimal route in the empty network.

Step 2:Calculate the time dependent costs of the road segments by simulation.

Step 3:Recalculate the optimal routes of a certain portion p of the drivers using the time dependent costs from step 2.

Step 4:If routes have changed in step 3, go to step 2.

Note that the DTA algorithm involves not only shortest path graph computations but also simulations. The purpose of the simulation is to help DTA acquire a relative accurate estimation of the travel times given the assignment of the previous iteration. Then, the estimated travel times are used to adjust the assignment in the next iteration. However, this inevitably leads to increased computational burden. In comparison, our approach proposes alternative routes to drivers during their entire journey based on the dynamic conditions in the road network, and most of the computation is spent on shortest path graph algorithms. Therefore, we expect our approach to be more efficient than DTA.

V. Evaluation

The main objective of our simulation-based evaluation is to study the performance of the five re-routing strategies under various scenarios. Specifically, we address the following questions:

- Which strategy leads to the most benefits for drivers in terms of travel time and number of re-routings?
- What is the tradeoff between strategy effectiveness and their efficiency in terms of computation time? How do the proposed strategies compare to a DTA-based approach in terms of effectiveness and efficiency?
- Which strategies scale better with the number of cars?
- How do parameters (number of alternative paths, car selection level, etc.) influence the performance?
- How robust is the system under various compliance rates (i.e., percentage of drivers who follow the guidance) and penetration rates (i.e., percentage of vehicles which have our software)?

A. Simulation setup

We employed SUMO 15.0 [7] and TraCI [41] for our simulations. SUMO is an open source, highly portable, microscopic road traffic simulation package designed to handle large road networks. TraCI is a library providing extensive commands to control the behavior of the simulation including vehicle state, road configuration, and traffic lights. We implemented the re-routing strategies algorithms using TraCI. Essentially, when SUMO is called with the option to use TraCI, SUMO starts up, loads the scenario, and then waits for a command. Thus, variables in the simulation can be changed (e.g., new paths assigned to certain vehicles). Then, a new command can be sent with how many seconds to run the simulation before stopping and waiting for another command.

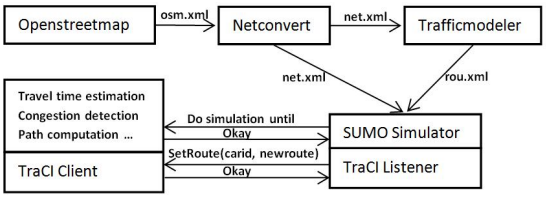


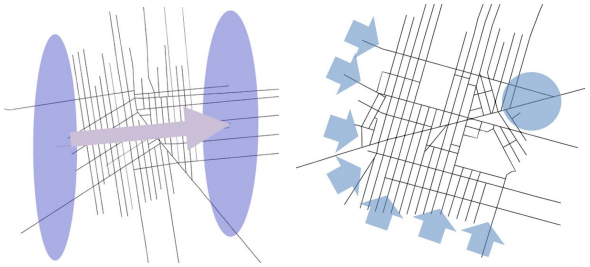
Fig. 5: The simulation process

TABLE I: Statistics of the two road networks

	Brooklyn	**Newark**
Network area	$75.85km^2$	$24.82km^2$
Total number of road segments	551	578
Total length of road segments	$155.55km$	$111.41km$
Total number of intersections	192	195

We downloaded two urban road maps from OpenStreetMap [16] in osm format. One is a section of Brooklyn, NY and the other is in Newark, NJ. We use the Netconvert tool in SUMO to convert the maps into a SUMO usable format, and the Trafficmodeler tool [31] to generate vehicle trips. Netconvert removes the pedestrian, railroad, and bus routes, and sets up a static traffic light at each intersection to make the simulations more realistic (as the maps do not have STOP signs). All roads have the same speed limit (13.9 m/s); some roads have one lane in each direction, while others have just one lane based on the specification in the OpenStreetMap osm file. The statistics of the two networks are shown in Table I. By default, the shortest travel time paths are automatically calculated and assigned to each vehicle at the beginning of simulation based on the speed limit. Figure 5 illustrates the simulation process. Figures 6 (a) (b) show the traffic flow in both networks. We used Trafficmodeler to generate a total of 1000 cars in the Brooklyn network from the left area to the right area in an interval of 1000 seconds. The origins and the destinations are randomly picked from the left area and the right area, respectively. In the Newark network, 906 cars were generated having the origins picked randomly from the peripheral road segments and the destinations on the road segments inside the hot spot circle.

In the simulations, we use the default settings in SUMO 15.0 for vehicle length=5m, the minimal gap=2.5m, the car following model (Krauss [22]), and the driver's imperfection=0.5. For each scenario, we average the results over 20 runs. Initially, we assume an ideal scenario in which all drivers have the system and accept the route guidance. We relax these assumptions in the last part of the evaluation. Table II defines



(a) Brooklyn (b) Newark

Fig. 6: Traffic flow in the road networks

the parameters used in our evaluation. We performed extensive experiments to determine the best values for these parameters. Section V-B will show results for urgency function, level L, and number of paths k. For the sake of brevity, we do not show results for period, δ, and β. We choose 450s as re-routing period and 0.7 as the congestion threshold because they produce good results with moderate computation. We observe that β between [0.05, 0.1] produces good results on both networks for AR^*. Thus, we select 0.05 for all the following experiments.

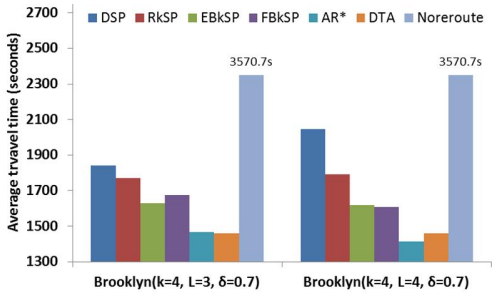
We also implemented a DTA-based re-routing strategy (cf. Section IV-D) by using a DTA tool provided with the SUMO generator.

B. Results and Analysis

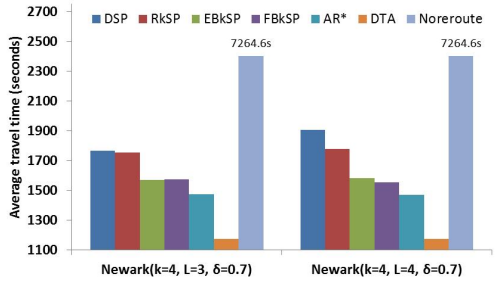
Average travel time. Figure 7 presents the average travel time obtained with the five strategies and with DTA on both networks. The "no-reroute" bars indicate the travel time in the absence of any re-routing. The results show that all the proposed strategies improve the travel time significantly. In most cases, the proposed strategies obtain travel times at least two times lower than no-rerouting. For instance, with a selection level of 3, compared to "no-reroute", EBkSP reduces the travel time by 2.2 times and 4.5 times on Brooklyn and Newark, respectively. As expected, DTA has the best average travel time since it can achieve user equilibrium. Based solely on the obtained average travel time, we rank the five strategies as following: DTA>AR* >(EBkSP, FBkSP)>RkSP>DSP>no-rerouting. The results confirm the hypotheses laid out in Section IV with the statistical significance of 95% confidence

TABLE II: Parameters used in the evaluation

period	The frequency of triggering the re-routing; by default period=450s
threshold δ	Congestion threshold; if $K_i/K_{jam} > \delta$, the road segment is considered congested; by default $\delta = 0.7$
urgency	Urgency policy: RCI or ACI
level L	Network depth to select vehicles for re-routing starting from the congested segment and using BFS on the inverted network graph
# paths k	The max number of alternative paths for each vehicle; by default $k = 4$
repulsion weight β	The weight of repulsion in AR^*; by default $\beta = 0.05$



(a) Brooklyn



(b) Newark

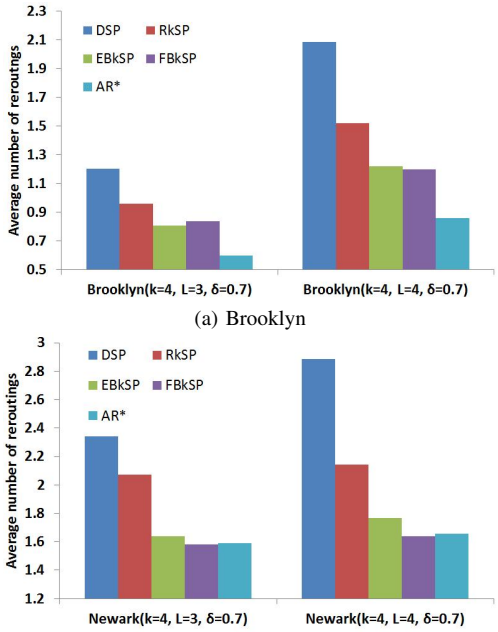
Fig. 7: Average travel time (L=(3,4), k=4, urgency=ACI, period=450s, δ=0.7, β=0.05)

interval. DSP can improve the travel time because it re-routes dynamically the vehicles by considering the traffic conditions. However, in some cases, e.g., if many vehicles have similar current positions and destinations, respectively, new congestions can be created by the re-routing process. RkSP avoids this shortcoming since it balances the traffic flow over several paths. Nevertheless, a randomly picked path is not necessarily the best one. EBkSP and FBkSP offer even better performance by carefully selecting the path for each re-routed vehicle. Finally, AR* has the best performance among the proposed strategies as it considers all the other vehicles in the road network in the computation of a new route.

Our experiments also demonstrated that setting the depth level to 3 or 4 is best for selecting a relatively optimal number of vehicles for re-routing (the two values lead to similar performance for Brooklyn, while level 3 is better for Newark). Lower level values do not select enough cars, whereas higher values increase the number of re-routings (see Figure 8). Therefore, we set the level parameter to 3 in the remaining experiments.

Average number of re-routings. It is important that the re-routing frequency for a given vehicle during a trip stay low. From the driver point of view, changing the path to the destination too often can be distracting and annoying. From the system point of view, having a low number of re-routings means decreasing the computational burden because the re-routing process is costly. Figure 8 compares the number of re-routings across the five proposed strategies. In terms of average number of re-routings AR* <(EBkSP, FBkSP)<RkSP<DSP with 95% confidence interval [4]. Compared to DSP, AR* reduces the average number of re-routings by up to 2.0 and 1.5 times, while compared to RkSP, AR* is better by 1.6 and 1.3 times on Brooklyn and Newark, respectively. The reason

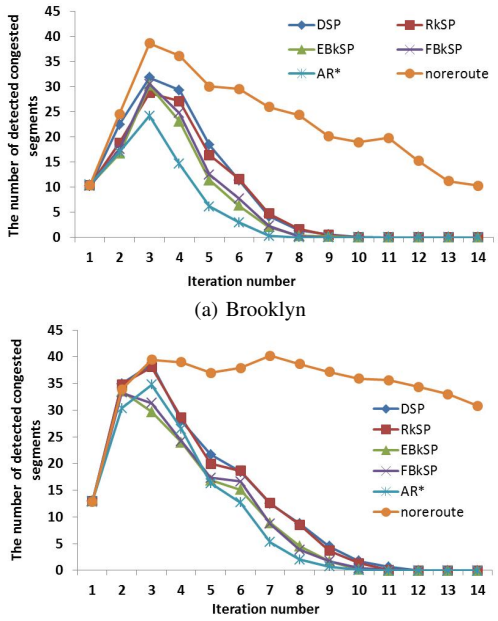
[4]We performed a t-test for each pair of strategies.



(a) Brooklyn

(b) Newark

Fig. 8: Average number of re-routings (L=(3,4), k=4, urgency=ACI, period=450s, δ=0.7, β=0.05)



(a) Brooklyn

(b) Newark

Fig. 9: Number of congested road segments as function of the number of iterations.(L=3, k=4, urgency=ACI, period=450s, δ=0.7, β=0.05)

is that by considering future path information in the re-routing decision, EBkSP, FBkSP and AR* can not only mitigate the current congestion, but also avoid creating new congestions; hence, the lower necessity for recurrent re-routing.

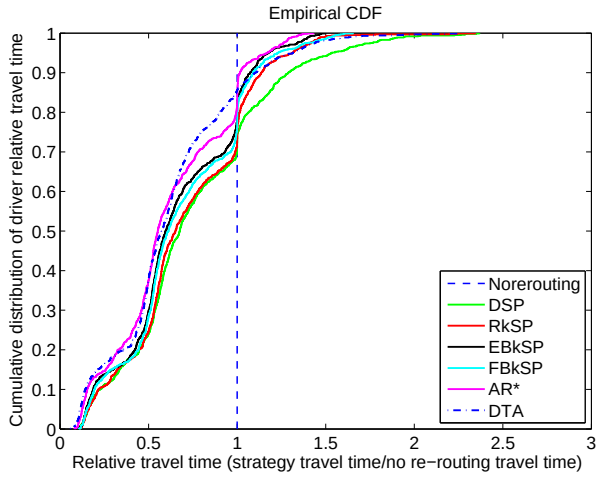
To confirm this analysis, we also measured the number of congested segments in each iteration. Figure 9 shows the results. As traffic is generated during the first 1000 seconds (i.e., iterations 1-3), the number of congested roads increases for all strategies. Then, the number of congested roads decreases for two reasons. First, no more traffic is generated, and this effect is observed in the "no-rerouting" curve. Second, and more importantly for our strategies, re-routing helps to dramatically reduce this number. As expected, EBkSP and FBkSP have comparable results and reduce the number of congested roads faster than DSP and RkSP. Also, we noticed that although AR* did not have the best performance at the beginning of the simulation, it was capable to alleviate congestion much faster than the other methods afterwards.

Distribution of travel time and re-routing frequency. The average travel time and the average number of re-routings measure the performance of the system from a global point of view. Here, we investigate the performance from a driver point of view. How many drivers end up with a shorter travel time? We introduce two new metrics. The first metric is the relative travel time (RelT), which is defined as the ratio of the travel time with re-routing and the travel time with no re-routing; thus, RelT measures the travel time gains or loses for individual drivers. The second metric is RRF, which in this experiment is defined as the number of re-routings per hour experienced by a driver; thus, RRF measures the driver distraction due to re-routing.

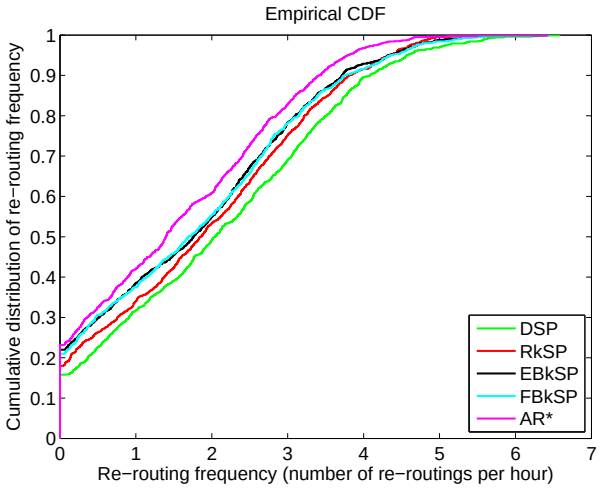
Figure 10 (b) presents the cumulative distribution of RelT and RRF for each re-routing strategy for the Brooklyn network. The values are averages (per driver) computed across 20 runs of simulations. We obtained similar results for the Newark network, which we omit. AR* has the best results for both RelT and RRF, followed closely by EBkSP and FBkSP. The system manages to improve the travel time for a large majority of drivers. Similarly, a large majority of drivers experience no more than 3 re-routings per hour, which we believe is acceptable in city scenarios with heavy traffic.

However, there is a relatively small percentage of drivers (i.e., ranging from 10% for AR* to 25% for DSP), that end up with increased travel time after re-routing. The observed increase is limited to less than 50% for most of these drivers. Note that this phenomenon is equally present in DTA, where around 15% of the drivers have increased travel time. The main reason for these results is that the proposed re-routing strategies have not been designed to achieve user-optimal equilibrium, and thus cannot guarantee the best travel time for each user. More surprisingly, even DTA which was designed to achieve user equilibrium cannot do it; our conjecture is that this is due to the difficulty to find an equilibrium under congestion [9]. We understand that a few bad experiences with the system could impact its adoption rate. Therefore, as future work, we plan to investigate strategies to lower the number of drivers with increased travel time and to bound this increase to low values.

CPU time. So far, the results indicate that AR* produces the best travel times (near to the DTA times), followed closely by EBkSP, FBkSP, and in some cases, by RkSP. An important question is what is the computational performance among all the proposed five strategies. To answer it, we need to first look at the algorithm complexity for the Dijkstra shortest path (used by DSP), k-shortest paths (used by RkSP, EBkSP and FBkSP) and A* (used by AR*). Dijkstra shortest path and k-shortest paths require $O(E+VlogV)$ and $O(kV(E+VlogV))$,



(a) Relative travel time CDF



(b) Re-routing frequency CDF

Fig. 10: CDF of relative travel time and re-routing frequency per hour on Brooklyn network.(L=3, k=4, urgency=ACI, period=450s, δ=0.7, β=0.05)

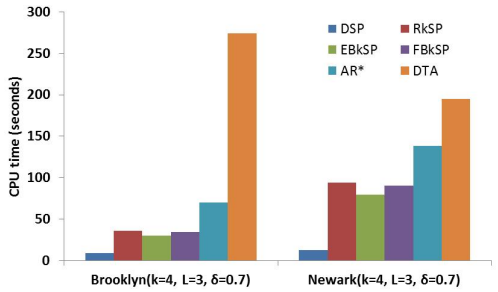
respectively, while A* was proven to be faster than Dijkstra [35]. However, this complexity analysis is pertinent only when we consider the selection of an alternative path for one single vehicle. From the system point of view, the global computational complexity also depends on the number of re-routings processed in a time window; this number is a function of the number of congested road segments and the congestion severity (i.e., how many vehicles are selected for re-routing). Additionally, DSP, RkSP, FBkSP and EBkSP compute shortest paths after grouping the vehicles on their origin-destination, whereas AR* calculates a new path for each vehicle. Therefore, AR* could require a larger computation time than the other methods.

Figure 11 (a) shows the global CPU time consumed for re-routing by the five methods and by DTA. Note that the experiments were conducted on a 64 bit Ubuntu machine with Intel Core i5-2467M CPU(1.6GHz) and 4GB of memory. We observe DSP requires the least CPU time for re-routing, mainly due to the low complexity of the shortest path algorithm and to grouping the re-routed vehicles. AR* consumes significantly more CPU time. For example, it requires 2 and 2.3 times more CPU time than RkSP and EBkSP on Brooklyn. The main reason is that AR* cannot group the re-routed vehicles like the other methods as stated in Section IV-A2.

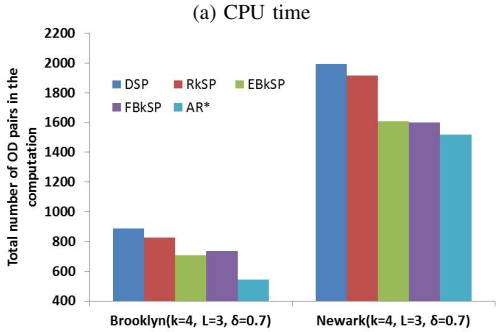
EBkSP, FBkSP and RkSP are situated in between the above mentioned methods from the CPU time point of view. Interestingly, EBkSP and FBkSP require less computation time than RkSP even though they execute more complex path selection algorithms in addition to the k-shortest path computation. The explanation is that EBkSP and FBkSP decrease the total number of re-routings processed in a period. This decrease becomes apparent when we look at the number of origin-destination (OD) pairs involved in the computation as indicated in Figure 11 (b). The total number of OD pairs is lower for EBkSP and FBkSP than for RkSP. While DSP has the largest number of OD pairs, it still has the lowest CPU time because of its much lower computational complexity for path calculation.

DTA has the largest CPU time and scales poorly with an increasing number of vehicles (in terms of CPU time) when compared to AR* or the other proposed methods (as shown in Figure 12 (b)). Also, it is worth noticing that DTA assumes all vehicles in the system known at the beginning (i.e., when it computes its routes). However, in real life, vehicles may appear at any time, and DTA would be required to perform its expensive computation over and over again. Therefore, due to its very high computational cost in real life, DTA may be impractical (i.e., it may not be able to compute alternative routes fast enough in order to mitigate congestions).

In conclusion, if we consider both the travel time and the CPU time, EBkSP and FBkSP appear to be the best strategies since they offer the best trade-off between re-routing effectiveness and computational efficiency. If computational cost is not an issue, one can use the AR* strategy, while in

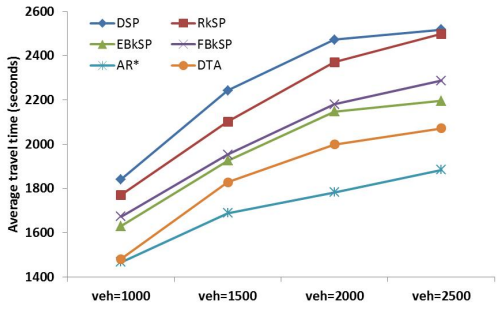


(a) CPU time

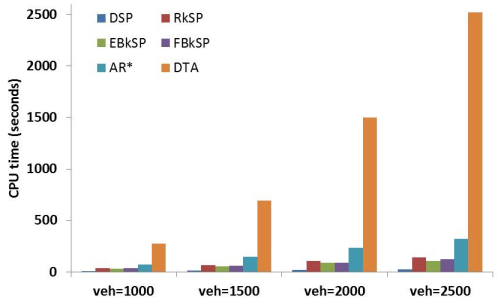


(b) Number of OD pairs

Fig. 11: CPU time and number of origin-destination pairs for both networks (L=3, k=4, urgency=ACI, period=450s, δ=0.7, β=0.05)

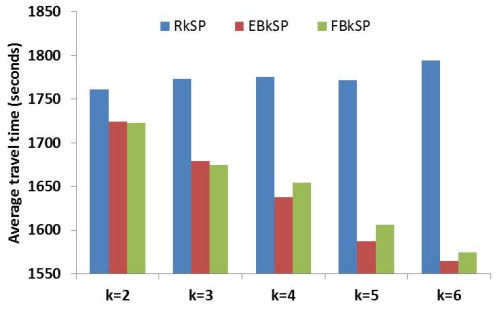


(a) Average travel time

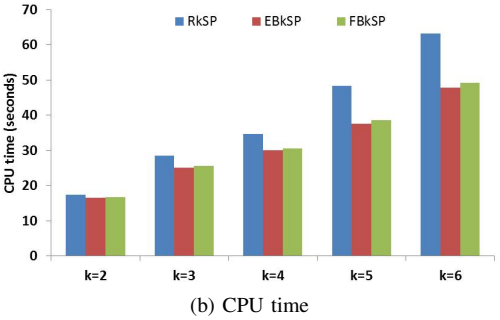


(b) CPU time

Fig. 12: The average travel time and CPU time for Brooklyn network for different traffic densities (L=3, k=4, urgency=ACI, period=450s, δ=0.7, β=0.05)



(a) Average travel time



(b) CPU time

Fig. 13: Average travel time, CPU time for RkSP, EBkSP and FBkSP as function of k for the Brooklyn network (L=3, k=(2, 3, 4, 5, 6), urgency=ACI, period=450s, δ=0.7)

the opposite case, DSP is the most appropriate choice.

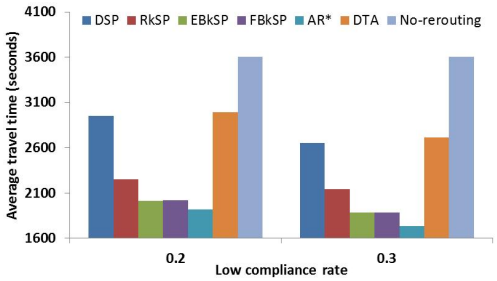
Traffic density. The results presented up to here already offer a good idea about the capabilities of the proposed re-routing strategies to alleviate traffic congestions. Yet there is an important aspect that still needs to be explored, i.e., how the proposed methods scale with the increase of the traffic volume. To respond to this question, we conducted another set of experiments on the Brooklyn network, where we increased the number of vehicles from 1000 to 2500. Figure 12 shows the obtained results both for the average travel time and the CPU time for different traffic densities. AR* and DTA present the best scalability from the average travel time point of view. However, these methods are also the least scalable from the CPU time point of view. As we can see, DTA exhibits particularly poor scalability compared to the proposed strategies, confirming our hypothesis that DTA is not yet a suitable approach for real-time traffic management. Also, somewhat interestingly, AR* obtained better average travel times than DTA (see Figure 12 (a)) when the number of vehicles was above 1500. This is certainly due to the fact that the 50 iterations limit we set in the DTA tool is not sufficient to achieve user equilibrium for higher traffic densities. Therefore, a higher number of iteration is needed in this case, which will lead evidently to even higher CPU times.

Number of alternative paths. k is a determinant parameter for the performance of RkSP, FBkSP and EBkSP, which require k-shortest paths computation. A larger k value allows for better traffic balancing but introduces higher computational complexity. Furthermore, the maximum allowed difference between the slowest path and the fastest path is 20% in our setting. Therefore, large k values may not be necessary because they would lead to computing many useless paths. Figure 13 compares the performance of RkSP, EBkSP and FBkSP with different k values on the Brooklyn network. The k value is irrelevant for DSP and AR*.

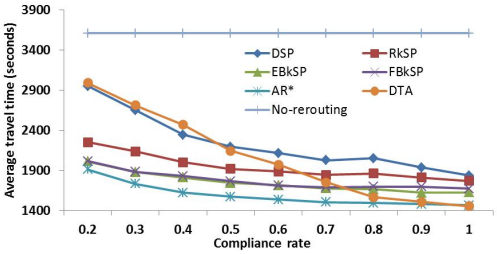
We observe that RkSP does not exhibit any performance improvement for $k > 2$, while both EBkSP and FBkSP consistently produce lower travel times with higher k values. Figure 13 (b) shows the computational cost increases linearly with k for all the kSP methods. However, EBkSP and FBkSP are more scalable than RkSP especially for larger k values (e.g., EBkSP requires 32% less CPU time than RkSP when k equals 6). The efficiency of EBkSP and FBkSP is due to the reduction of the number of OD pairs.

Urgency function. Among the five proposed algorithms, EBkSP, FBkSP and AR* use an urgency function to sort the list of vehicles selected for re-routing (cf. Section III-D). To measure the performance difference between the two proposed ranking policies – RCI and ACI (cf. Definition 1), we conducted the ANOVA statistics test over the average travel time from 30 simulations with EBkSP and FBkSP. The results show that ACI produces lower average travel times than RCI ($p < 0.01$) with a 95% confidence interval. The result confirmed our previous analysis in Section III-D. Thus, we used ACI as our default urgency function in all the experiments.

Compliance rate. It is unrealistic to assume that every driver follows the re-routing guidance. The drivers' compliance rate is an important factor for the re-routing strategy design. Therefore, we measured the average travel time while varying the compliance rate for the five proposed strategies and for DTA. For our strategies, a compliance rate of x% means that each of the selected vehicles switches to the new route with x% probability during each re-routing period. For DTA, x% of the vehicles are randomly selected to follow the DTA assigned route, while the rest follow their initial shortest time route.

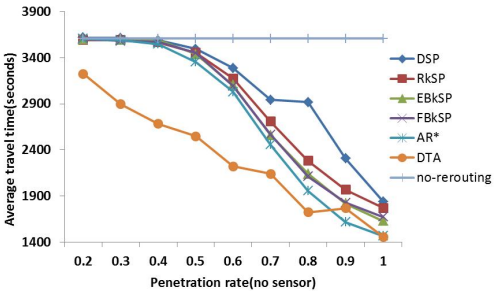


(a) Low compliance rate

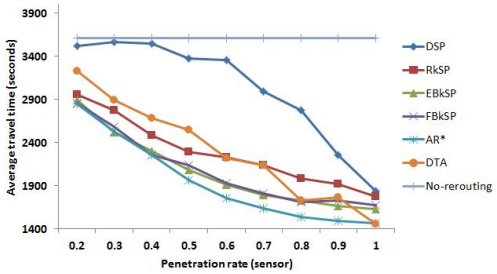


(b) All compliance rates

Fig. 14: Average travel time as a function of the compliance rate on Brooklyn network. (L=3, k=4, urgency=ACI, period=450s, δ=0.7, β=0.05)



(a) Sensors: Vehicles Only



(b) Sensors: Vehicles & Road-Side

Fig. 15: Average travel time as a function of the penetration rate on Brooklyn network (L=3, k=4, urgency=ACI, period=450s, δ=0.7, β=0.05)

Figure 14 (a) indicates that the average travel time can be significantly improved by all five strategies even under low compliance rates. This is due to the fact that even under low compliance rates, the drivers who comply with the guidance can still receive more rapid routes, which in turn can improve the travel time for the rest of the drivers. Figure 14 (b) shows the average travel time for a wide range of compliance rates.

In particular, when the compliance rate is low, RkSP, EBkSP, FBkSP and AR* show significantly better travel times than DTA. The reason is that when compliance is low, the drivers who comply benefit much more from our guidance than from the DTA guidance. In the DTA approach, the route computation is done once before any vehicle enters the network. If the compliance rate is low, the DTA-computed routes are far from a user equilibrium, inclusively for the compliant drivers. Differently, our strategies can adjust the vehicles' routes periodically based on the current traffic information. Therefore, although the non-compliant drivers create congestion in the network, the compliant ones can still receive fairly good paths, which implicitly reduces the congestion level in the network.

Penetration rate. To understand how easy is to deploy our solution in real life, we study the effect of the penetration rate on the average travel time. Specifically, each vehicle has $x\%$ probability of participating in the system, i.e., providing positioning information and receiving guidance. Hence, the system sees and guides only these $x\%$ of the vehicles. The accuracy of congestion detection and road travel time estimation depend directly on this parameter. Additionally, similar to the compliance rate, the effectiveness of the load balancing mechanism implemented by the re-routing strategies increases with the percentage of vehicles that use the system.

In this set of experiments, we consider two cases: only the vehicles equipped with the system provide traffic information (which we call "penetration rate without sensors"), and both vehicles equipped with the system and road-side sensors provide traffic information (which we call "penetration rate with sensors"). For clarity, we list in Table III the main differences between these two cases and the experiments for compliance rate.

Figure 15 (a) shows the average travel time for various penetration rates, when traffic data are collected only from vehicles (i.e., no support from road-side sensors). When the penetration rate is low, the performance of the proposed methods is the same as "no-reroute". In this case, the service does not have enough data to accurately detect signs of congestion. Once the penetration rate is greater than 0.4, the system is able to improve the travel time. For penetration rates above 0.6, EBkSP, FBkSP and AR* start to perform better than DSP and RkSP, since a larger number of vehicles are re-routed, which requires a more advanced load balancing mechanism. Compared to our methods, DTA performs better under low penetration rates since the DTA re-routing is not triggered by the congestion detection as in our approach.

To boost the adoption of the system, we believe that data from road-side sensors (in conjunction with data from vehicles) can be leveraged to detect congestion more accurately in case of low penetration rates. When road sensors are present, the road traffic density can be measured, the congestion can be detected, and the travel time can be estimated more accurately. Figure 15 (b) demonstrates that most of our methods can significantly improve the travel time even under low penetration rates if road-side sensor information is available. Furthermore, EBkSP, FBkSP, and AR* perform better than DTA in this case. When penetration rate is low ($x\%$), DTA distributes evenly the $x\%$ vehicles without considering the rest, which can still create congestion. Therefore, the alternative routes proposed by DTA are not as effective in alleviating congestion. Our strategies can take advantage of the sensor information to divert the $x\%$ of

TABLE III: Comparison between compliance and penetration rate

	Road traffic density	**Congestion detection**	**Re-routing**	**Load balancing**
Compliance rate	Accurate	Accurate	All vehicles provide positioning information. At each re-routing, only $x\%$ of vehicles follow the guidance	Considers all vehicles (except DSP)
Penetration rate without sensors	Inaccurate	Inaccurate	Predefined $x\%$ of vehicles that provide positioning information receive guidance and comply with the guidance	Considers only the $x\%$ of vehicles (except DSP)
Penetration rate with sensors	Accurate	Accurate	Predefined $x\%$ of vehicles that receive guidance and comply with the guidance. The system obtains accurate traffic data from sensors and vehicles	Considers only the $x\%$ of vehicles (except DSP)

the drivers and to reduce congestion.

Summary of the experimental results. The objectives of the experiments were threefold. First, we measured the effectiveness of the proposed methods to alleviate congestion and to improve the drivers' travel time, their computational efficiency in achieving this goal, and their scalability with an increasing number of vehicles. To have a more precise idea of the performance level offered by the proposed methods, we used a state-of-the-art DTA tool as a baseline. The experimental results show our methods are very effective in fighting congestion, being capable of improving the travel time as much as a DTA approach. In addition, our methods are (much) more computationally efficient and scalable than DTA, which makes them more appropriate for use in real-life applications. Furthermore, the palette of proposed methods offers a wide range of choices having different trade-offs between effectiveness and efficiency, which responds to a large variety of real-life scenarios.

Second, we conducted a more in-depth set of experiments to understand how the parameters used by our methods influence their performance. The experimental results give a good indication as to which are the most suitable values for some of these parameters (i.e., urgency function, re-routing period, congestion threshold) in conjunction with the employed method. For other parameters (i.e., number of alternative paths k) and algorithms (i.e., EBkSP and FBkSP), it is still a matter of choice between effectiveness and efficiency, opening the possibility to even finer system tuning.

Third, we considered a more realistic scenario and assessed the robustness of our system under compliance and penetration rates below 100%. The results indicate that our system can still improve significantly the travel time even under low compliance rates and that our approach is more robust than DTA. Besides, if accurate traffic data (e.g., collected by road side sensors) can be provided to the system, then our strategies exhibit good robustness with various penetration rates.

TABLE IV: Comparison of all the five strategies

Effectiveness	AR* >(EBkSP, FBkSP)>RkSP>DSP
Efficiency	DSP>(EBkSP, FBkSP)>RkSP>AR*
Re-routing frequency	AR* >(EBkSP, FBkSP)>RkSP>DSP
Robustness	(AR*, EBkSP, FBkSP)>RkSP>DSP

To summarize the performance of the proposed strategies, we rank them according to four criteria in Table IV, where $A > B$ means that strategy A is better than strategy B. While each method has its own advantages and shortcomings, the experimental results made us conclude that the entropy method (EBkSP) and the local optimization method (FBkSP), which led to very similar results, should be the preferred strategies, as they offer the best trade-off between performance and computation cost.

VI. Conclusions and Future Work

Heartened by the ubiquitousness of mobile devices such as smart phones or on-bord vehicle units, this paper envisions a novel approach to tackle the ever more stringent problem of traffic congestion. This approach is based on a traffic guidance system that monitors traffic and proactively pushes individually-tailord re-routing guidance to vehicles when there are signs of congestion. The system is responsible for several functions such as traffic data representation, congestion prediction, and selection of the vehicles to be re-routed. We chose to focus in this paper on a key element of our re-routing system, i.e., the re-routing strategies. We proposed five re-routing strategies to compute alternative routes for vehicles. Then, we conducted an extensive set of simulation-based experiments to validate our approach. The results showed the proposed re-routing algorithms are very effective in mitigating congestion and adapt well to the dynamic nature of the traffic, being also more efficient and scalable than existing approaches. In addition, our traffic guidance system remains

useful even with low compliance rate and moderate penetration rate. The experiments also demonstrated how the performance can be tuned by varying parameters such as re-routing method, number of alternative paths, and density threshold.

As future work, we plan to explore three directions. First, we will design an adaptive approach for vehicle selection that considers additional parameters such as road segment length, measured compliance rate, and estimated penetration rate. The objective is to obtain a (fully) self-tunable system capable of automatically adjusting the system parameters depending on the traffic conditions. Second, we will study methods to reduce the number of drivers who do not benefit from re-routing and to limit the increase in their travel time. Third, we intend to investigate a hybrid architecture that off-loads parts of the computation and decision process in the network and uses V2V communication to better balance the need for privacy, scalability, and low overhead with the main goal of low average travel time.

REFERENCES

[1] http://www.google.com/mobile/.

[2] http://www.tomtom.com/en_gb/products/mobile-navigation/.

[3] http://www.inrix.com.

[4] http://www.autobahn.nrw.de.

[5] http://sumo.sourceforge.net/doc/current/docs/userdoc/Tools/Assign.html.

[6] J.H. Banks. *Introduction to transportation engineering*. McGraw-Hill, 2002.

[7] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. Sumo - simulation of urban mobility: An overview. In *Proceedings of the 3rd International Conference on Advances in System Simulation(SIMUL 2011)*, pages 63–68, Barcelona, Spain, 2011.

[8] M. Behrisch, D. Krajzewicz, and Y.P. Wang. Comparing performance and quality of traffic assignment techniques for microscopic road traffic simulations. *DTA2008*, 2008.

[9] Y.C. Chiu, J. Bottom, M. Mahut, A. Paz, R. Balakrishna, T. Waller, and J. Hicks. Dynamic traffic assignment: A primer. *Transportation Research E-Circular*, (E-C153), 2011.

[10] J. Cranshaw, E. Toch, J. Hong, A. Kittur, and N. Sadeh. Bridging the gap between physical location and online social networks. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*, pages 119–128. ACM, 2010.

[11] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan. The pothole patrol: using a mobile sensor network for road surface monitoring. In *Proceeding of the 6th international conference on Mobile systems, applications, and services (MobiSys 2008)*, pages 29–39. ACM, 2008.

[12] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, July 1987.

[13] T.L. Friesz, D. Bernstein, T.E. Smith, R.L. Tobin, and BW Wie. A variational inequality formulation of the dynamic network user equilibrium problem. *Operations Research*, pages 179–191, 1993.

[14] C. Gawron. *Simulation-based traffic assignment–computing user equilibria in large street networks*. PhD thesis, University of Cologne, 1999.

[15] B. George, S. Kim, and S. Shekhar. Spatio-temporal network databases and routing algorithms: A summary of results. *Advances in Spatial and Temporal Databases*, pages 460–477, 2007.

[16] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, 2008.

[17] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[18] H.H. Hoos and T. Stützle. *Stochastic local search: Foundations and applications*. Morgan Kaufmann, 2005.

[19] E. Horvitz, J. Apacible, R. Sarin, and L. Liao. Prediction, expectation, and surprise: Methods, designs, and study of a deployed traffic forecasting service. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence(UAI-2005)*, pages 244–257, 2005.

[20] T. Hunter, R. Herring, P. Abbeel, and A. Bayen. Path and travel time inference from gps probe vehicle data. *NIPS Analyzing Networks and Learning with Graphs*, 2009.

[21] B.S. Kerner. Optimum principle for a vehicular traffic network: minimum probability of congestion. *Journal of Physics A: Mathematical and Theoretical*, 44:092001, 2011.

[22] S. Krauss, P. Wagner, and C. Gawron. Metastable states in a microscopic model of traffic flow. *Physical Review E*, 55(5):5597, 1997.

[23] E.L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, pages 401–405, 1972.

[24] S. Maerivoet. *Modelling Traffic on Motorways: State-of-the-Art, Numerical Data Analysis, and Dynamic Traffic Assignment*. PhD thesis, Katholieke Universiteit Leuven, 2006.

[25] H. S. Mahmassani, T-Y. Hu, and R. Jayakrishnan. Dynamic traffic assignment and simulation for advanced network informatics (dynasmart). In *Proceedings of the 2nd International CAPRI Seminar on Urban Traffic Networks*, Capri, Italy, 1992.

[26] N. Malviya, S. Madden, and A. Bhattacharya. A continuous query system for dynamic route planning. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 792–803. IEEE, 2011.

[27] H.C. Manual. Highway capacity manual, 2000.

[28] D.K. Merchant and G.L. Nemhauser. A model and an algorithm for the dynamic traffic assignment problems. *Transportation Science*, 12(3):183–199, 1978.

[29] D.K. Merchant and G.L. Nemhauser. Optimality conditions for a dynamic traffic assignment model. *Transportation Science*, 12(3):200–207, 1978.

[30] P. Mohan, V.N. Padmanabhan, and R. Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 323–336. ACM, 2008.

[31] LG Papaleondiou and M.D. Dikaiakos. Trafficmodeler: A graphical tool for programming microscopic traffic simulators through high-level abstractions. In *Proceedings of the 69th IEEE Vehicular Technology Conference(VTC Spring 2009)*, pages 1–5. IEEE, 2009.

[32] S. Peeta and A.K. Ziliaskopoulos. Foundations of dynamic traffic assignment: The past, the present and the future. *Networks and Spatial Economics*, 1(3):233–265, 2001.

[33] H. Prothmann, H. Schmeck, S. Tomforde, J. Lyda, J. Hahner, C. Muller-Schloer, and J. Branke. Decentralized route guidance in organic traffic control. In *Proceedings of the 5th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2011)*, pages 219–220. IEEE, 2011.

[34] D. Schrank, T. Lomax, and S. Turner. Tti urban mobility report, 2011.

[35] D. Schultes. Route planning in road networks. *Karlsruhe: Universität Karlsruhe (TH). Fakultät für Informatik. Institut für Theoretische Informatik, Algorithmik II*, 2008.

[36] S. Senge and H. Wedde. Bee inspired online vehicle routing in large traffic systems. In *Proceedings of the 2nd International Conference on Adaptive and Self-Adaptive Systems and Applications(ADAPTIVE 2010)*, pages 78–83, 2010.

[37] C.E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[38] B. Tatomir, S. Fitrianie, M. Paltanea, and L. Rothkrantz. Dynamic routing in traffic networks and manets using ant based algorithms. In *Proceedings of the 7th International Conference on Artificial Evolution, Lille, France*, October 2005.

[39] N.B Taylor. *CONTRAM 5, an enhanced traffic assignment model*. TRRL research report. Transport and Road Research Laboratory, 1990.

[40] J.G. Wardrop. Some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers, Part II*, 1(36):252–378, 1952.

[41] A. Wegener, M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, and J.P. Hubaux. Traci: an interface for coupling road traffic and network simulators. In *Proceedings of the 11th communications and networking simulation symposium*, pages 155–163. ACM, 2008.

[42] D.B. Work, O.P. Tossavainen, S. Blandin, A.M. Bayen, T. Iwuchukwu, and K. Tracton. An ensemble kalman filtering approach to highway traffic estimation using gps enabled mobile devices. In *Proceedings of the 47th IEEE Conference on Decision and Control, 2008.*, pages 5062–5068. IEEE, 2008.

[43] T. Xu and Y. Cai. Feeling-based location privacy protection for location-based services. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 348–357. ACM, 2009.

Juan (Susan) Pan received the M.S. degree in computer science from Tianjin University, Tianjin, China, in 2006. She is currently working toward the Ph.D. degree with the Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, USA. Her research interests include vehicular and ubiquitous computing, distributed systems, and social network analysis.

Iulian Sando Popa received the B.S. degree from the University Politehnica of Bucharest, Bucharest, Romania, in 2005 and the Ph.D. degree from the University of Versailles Saint-Quentin-en-Yvelines (UVSQ), Versailles, France, in 2009, both in computer science. Since 2012, he has been an Assistant Professor of computer science with the UVSQ and a member of the Secured and Mobile Information Systems Team, Inria, Le Chesnay, France. Since 2012, he has been an Assistant Professor of computer science with the UVSQ and a member of the Secured and Mobile Information Systems Team, Inria Paris-Rocquencourt, Le Chesnay 78145, France. His main research interests include embedded database management systems, spatiotemporal databases, and mobile data management and systems, with a particular focus on topics revolving around privacy and personal data management.

Karine Zeitouni received the Ph.D. degree in computer science from the Pierre and Marie Curie University, Paris, France, in 1991. She is a Professor of computer science with the University of Versailles-Saint-Quentin-en-Yvelines, Versailles, France. She is the author of around 70 referred articles in journals and conference proceedings. Her main research interests include spatiotemporal databases and knowledge extraction, with a focus on applications in the fields of transport, environment, and health. Dr. Zeitouni regularly serves at conferences of the (spatial) database community as a member of program committees (e.g., Association for Computing Machinery (ACM) Special Interest Group on SPATIAL INFORMATION, the International Symposium on Spatial and Temporal Databases, and the ACM Special Interest Group on Management of Data).

Cristian Borcea received the Ph.D. degree from Rutgers University, New Brunswick, NJ, USA, in 2004. He is currently an Associate Professor with the Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, USA. He is also a Visiting Associate Professor with the National Institute of Informatics, Tokyo, Japan. His research interests include mobile computing and sensing, ad hoc and vehicular networks, and distributed systems. Dr. Borcea is a member of the Association for Computing Machinery and USENIX.