Vehicular Sensor Networks

Juan Pan and Cristian Borcea

Computer Science Department

New Jersey Institute of Technology

University Heights, Newark, NJ 07102, USA

Email: jp238@njit.edu, borcea@njit.edu

Table of Contents

Vehicular Sensor Networks

I. INTRODUCTION

The global population spends a tremendous amount of time in traffic every day. For example, it is estimated that average American spends at least 50 minutes a day commuting [1]. Vehicular sensor networks, which leverage vehicular sensors and wireless communication, could improve our in-car experience by making travel safer and faster. In the past decades, with the advance of the new technologies, various electronics components started to be included in the automotive system such as acoustic, infrared, still/motion video cameras, etc. In recent years, most new vehicles come already equipped with GPS receivers and navigation systems. Car manufacturers such as Ford, GM, BMW and Toyota have already announced efforts to include significant computing power within their automotive design [2-5]. After Google revealed its first self-driving car, a number of auto-makers began venturing into this research area. This trend is expected to continue and, in the near future, the number of vehicles equipped with computing technologies and wireless network interfaces will increase dramatically. These vehicles will be able to run network protocols that will exchange messages for safer and more fluid traffic on the roads. Several protocols have been proposed for implementing vehicular communication. The most promising standard is IEEE 802.11p. The physical layer of 802.11p is dedicated short-range 5.9 GHz for communication among vehicles and with road side infrastructure [6].

So far, a diversity of applications have been proposed and evaluated over vehicular sensor networks. Based on the functionality, those applications can be divided into three categories: road safety, local information sharing and congestion monitoring/avoidance.

A typical safety related application example is highway cooperative collision avoidance [7]. On highways, an initial collision is often followed by a series of collisions involving the following vehicles. This is due to the inability of drivers to react in time to emergency situations. By using wireless communication, the initial car involved in the accident can warn the nearby vehicles such that their drivers can hit the brake faster. Another example is abnormal driver behaviour (e.g., drunkenness and fatigue) detection in literature [8], where each vehicle's on-board unit calculates the score of abnormality for each driver through Dynamic Bayesian Networks based on the contextual information collected from various sources including traffic management centre (TMC), road site units and other vehicles' broadcasting messages. Once the abnormality score is above a certain threshold, an alert is send to the driver to prevent accidents from happening.

A large body of applications focus on creating information sharing services to facilitate people's daily lives such as parking availability, taxi dispatching, and context-aware services. EZCab [9] describes a faster and more scalable system that allows people to book nearby cabs using their smartphones. When needed, a client sends a booking request to nearby vehicles, and the free cabs reply to the client by means of vehicular ad hoc communication. Similarly, the work in [10] proposes real-time parking navigation service in large parking lots through vehicular ad hoc communication. Each vehicle acts as a parking information senor and communicates with the incoming vehicles to find the most convenient parking spot. The literature [11] presents a context-aware migratory service model for regional information sharing such as traffic jam, local advertisements, parking lot availability, etc. The information in a specific area is sensed by the passing vehicles and disseminate to other vehicles in proximity.

Recently, utilizing vehicular sensor networks for congestion monitoring/avoidance has drawn great attention. Vehicular sensor networks provide a cost-effective platform for traffic data collection and vehicle re-routing to avoid congestion and to improve the travel time. The rest of this chapter will focus on leveraging vehicular sensor networks for congestion avoidance.

II. VEHICULAR SENSOR NETWORKS FOR CONGESTION AVOIDANCE

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. Projects such as Mobile Millennium [12, 13], CarTel [14], JamBayes [15], Nericell [16], and surface street estimation [17] use vehicle probe data collected from on-board GPS devices to reconstruct the state of traffic and estimate shortest travel time. Similarly, services such as INRIX [18] 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. Differently, Waze [19] utilizes social networks to provide traffic information. In these applications, vehicles act as traffic sensors and relay traffic reports to TMC to obtain an accurate view of the current traffic. 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.

Trafficview [20] and StreetSmart [21] are examples of distributed systems that could help drivers avoid congestion: vehicles collect and disseminate traffic information to nearby vehicles, and thus the drivers may change routes if they observe traffic congestions. Peertis [22, 23] discusses an efficient and scalable architecture that incorporates a peer-to-peer overlay into a vehicular network based on cellular Internet communication to provide traffic information to drivers. Scientists are looking for inspiration in Biology and Internet protocols as well. Wedde et al. developed a road traffic routing protocol in work [24], Bee-JamA, based on honey bee behavior. Similarly, Tatomir et al. [25] proposed a route guidance system based on trial-laying ability of ants. Inspired by the well-known Internet routing protocols, Prothmann et al. [26] proposed decentralized Organic Traffic Control. Some other distributed applications include intelligent traffic light adjustment [27]. Nevertheless, these applications simply use the collected data for naive sub-optimal shortest path computation which potentially switch congestion from one spot to another. Besides, for larger scale networks, due to wireless contention, latency and packet loss, vehicles are unable to acquire complete global traffic view, thus make sub-optimal re-routing decisions.

Many of the above solutions expect the drivers to react to congestion by changing their routes. According to Wardrop's first traffic equilibrium principle [28], if accurate traffic time on each road segment can be obtained, then drivers could be able to reach user-optimum traffic equilibrium. It is known, however, that no true equilibrium can be found under congestion [12]. Even more important, the usefulness of such services is limited by their reactive nature: they cannot prevent congestions.

This situation could be avoided by new solutions based on dynamic user-optimal traffic assignment (DTA) which lead 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 [29], [30, 31], 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 [33], [34], [35], [36] have gained greater acceptance in recent years: simulations are used to model the theoretical insights that cannot be derived quickly from analytical approaches. Unfortunately, there is still a significant gap between the theoretical or simulation results and potentially deployable solutions: 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.

In summary, despite good progress, existing solutions have not been able to provide an intelligent, cost-effective, easy-deployable real time traffic guidance system. Such a system should possess the following properties: 1) fully utilize vehicles as mobile sensors to provide accurate traffic view; 2) be able to compute near-optimal routing paths for vehicles based on the collaborative knowledge collected from vehicles; 3) provide effective vehicle re-routing guidance to individual drivers in a timely manner; 4) be robust and work well even when drivers do not follow the guidance. In the following section, we present a system that achieves these goals.

III. A DRIVER RE-ROUTING SYSTEM FOR CONGESTION AVOIDANCE

The objective is to implement and evaluate a real-time, cost-effective, and easily deployable vehicular traffic guidance system that reduces the effect of traffic congestions and lowers the trip times for all drivers. Implicitly, fuel consumption and pollution will be reduced as well. To achieve this goal, we developed a system consisting of smart phone-based vehicular networks and a back end server infrastructure for traffic monitoring and coordination. Smart phones were chosen as the vehicular platform because they are already carried by drivers in many vehicles, are powerful (have several communication interfaces, GPS, accelerometer, powerful CPU, plenty of storage, etc.), and are easily programmable. Once they become widespread, vehicular computing systems could be considered instead of smart phones. Figure 3.1 presents a comparison between these existing solutions and the system. We propose to leverage the smart phone ubiquity to design and quickly deploy a cheap and effective traffic re-routing system. Such a system will benefit all of us through faster routes, less money spent on gas, and lower pollution.

Figure 3.1: The system overview

Specifically, 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, this issue is not addressed and it is assume that the centralized service receives traffic data from vehicles and road-side sensors where available.

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 [37], [13]. We use the Greenshield's model [38] 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:

$$V_i = V_f\left(1 - \frac{K_i}{K_{jam}}\right), \quad T_i = \frac{L_i}{V_i} \quad (1)$$

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, $\frac{K_i}{K_{jam}}$ is the ratio between the current number of vehicles N_i and the max number of vehicles N_{max}. The current number of vehicles N_i is obtained from the traffic data collected by the service, whereas the N_{max} = length of road/ (average 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 $\frac{K_i}{K_{jam}} > \delta$, where $\delta \in [19]$ is a predefined threshold value. Choosing the right value for delta 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.

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 [39]. 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.

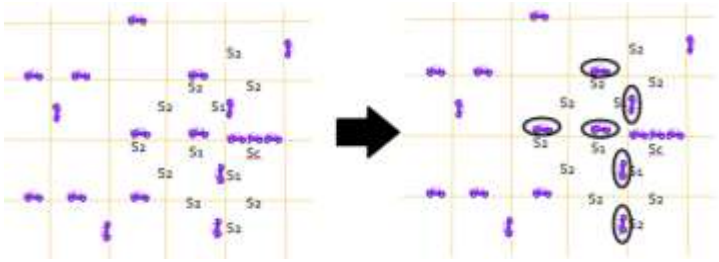


Figure 3.2: The vehicle selection process

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 rerouting. The process is illustrated in Figure 3.2. 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, path computations, and path assignments detailed in the following sections.

D. *Rank of the selected vehicles*

The selected vehicles need to be ranked and assigned to alternative paths according to their rank. In this way, the performance of the strategies improves. 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 it's 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.

E. Alternative Route Computation and Assignment

There are two main requirements for the re-routing algorithm: (1) compute an alternative routes for each driver that improve both single driver's trip time and global network efficacy (2) push the guidance to drivers fast to allow them enough time to switch on the new route. Essentially, a best effort algorithm is required, which finds good enough alternatives with real-time constraint.

Accordingly, two types of solutions were developed. The first solution is "Single Shortest Path Strategies", the second one is "Multiple Shortest Paths Strategies". The former computes one single path for each vehicle collaboratively according to the other vehicles' paths. The later computes k loopless shortest paths [40], [41], [42] according to current travel time and pick the optimal one for each vehicle. This is because one of the main goals of the system is to prevent moving congestion from one path to another. If all vehicles are re-routed on their current shortest paths, the algorithm might end-up doing exactly that. For this reason, vehicles are provided with alternative paths that lower their currently predicted travel time, but these paths do not have to be the shortest. The specification of the re-routing algorithms is presented in Section IV.

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 [15], [13], [16]. 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 rerouted 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 rerouting process executed by the traffic guidance system in Section IV-C. Finally, Section IV-D presents a Dynamic Traffic Assignment strategy [36] 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, the system described in this 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 + V log(V)) [43], where E is the number of road segments and V is the number of intersections of the road network. This strategy is expected 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, AR* is proposed, which modifies the A* search algorithm to include the prior re-routing decisions into the computation of the current shortest path. A* [44] uses best-first search and heuristic function to determine the order in which the network nodes (road intersections in this 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 this 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. As the drivers share their route information with the central 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 w_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, $V f_{avg}$ is the average free flow speed of the network, len_i is the length of i, $V f_i$ 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. The weights were incorporated into the formula in order to count for different road characteristics. For example, suppose there are two road segment 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*, the heuristic function F(x) is modified to include the other vehicles sharing the same path as a repulsive force. Specifically, the repulsive score R(x) of a node x is defined 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 same as in the original algorithm. 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, the values are normalized and F(x) is a linear combination of the two factors. Parameter β presents the weight of the repulsion. If β is too high, the resulting path will be diverted too far away from the optimal. Similarly, if β is too low, it will be the same as the naive shortest path (DSP) strategy. Therefore, the beta value was varied from 0.01 towards 0.5 and it was observed that the interval [0.05, 0.2] results in better performance in Section V-B.

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 F score value is removed from the queue (lines 16-19), the values of its neighbours are updated accordingly (lines 27-28), and these neighbours 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.

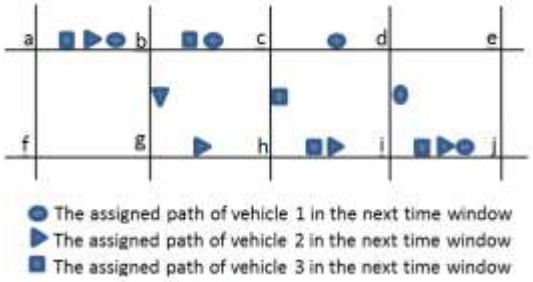


Figure 4.1: AR* re-routing example. All road segments have same weight and β= 0.5

Figure 4.1 illustrates a simple example of how AR* is used in re-routing. Given the assumption 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 v3 and the path ab, bc, cd, di, ij is obtained for the same reasons.

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
```

In

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 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 behaviours. Firstly, 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). Secondly, 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, a new class of re-routing strategies designed to obtain the best trade-off between efficiency and effectiveness are introduced. For these strategies, the re-routing process is divided into two steps. First, k-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, the computation time is expected 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 selects the path the least employed by other vehicle traces. Hence, this strategy is expected to have effectiveness similar to AR*. Three heuristics for the selection of the best path among the k-shortest paths are proposed.

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 + V log(V))) [40], 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, the EBkSP strategy is proposed. 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, this optimization is expected 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 [45]. Several works [46], [47] have successfully applied it to compute the popularity of a visited area among all the users. To avoid creating new congestions through re-routing, a "popularity" measure is associated with road segments in EBkSP. Entropy is used to define the popularity of a path as follows.

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* $Pop(p_j) = e^{E(pj)}$. *E(pj) is the weighted entropy of p_j and is computed as*

$$E(pj) = -\sum_{i=1}^{n} \frac{fc_i}{N} ln \frac{fc_i}{N}, N = \sum_{i=1}^{n} n_i.$$

The value of $E(p_j)$ measures the probability that a number of vehicles will be on the path pj in a time window. According to the above definition, the value of $Pop(p_j)$ is $0 < Pop(p_j) < m$, 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, the central server assigns 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.

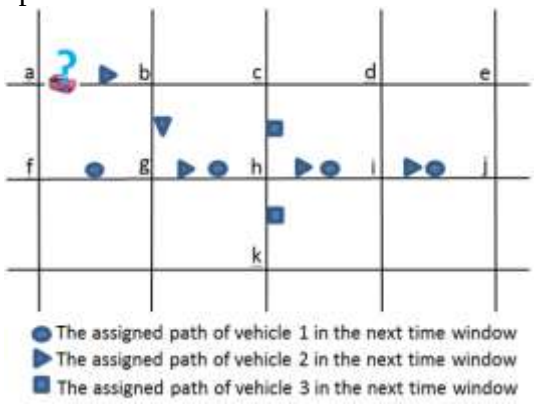


Figure 4.2: An EBkSP re-routing example. All segments have same weight

Figure 4.2 illustrates an example of EBkSP re-routing. It is assumed that vehicles (v_1, v_2, v_3) have been assigned to their paths before v_4, and each road has the same weight (i.e., $\omega_i = 1$). 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 (p_1)=2.29, E (p_2)=1.67 and E(p_3)=0.53. Hence, v_4 will be assigned to p_3 because it is the least popular.

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
```

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 designed to directly balance the traffic load, i.e., the weighted footprint counters, through local search optimization [48]. 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 in a network region, where S is the set of all region segments. As described in 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 4.3 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 4.3). Then, the sum of the weighted footprint counters in this network region is 18 (cf. Figure 4.3 (a)). 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 4.3 (b). Therefore, the system will select the latter assignment.

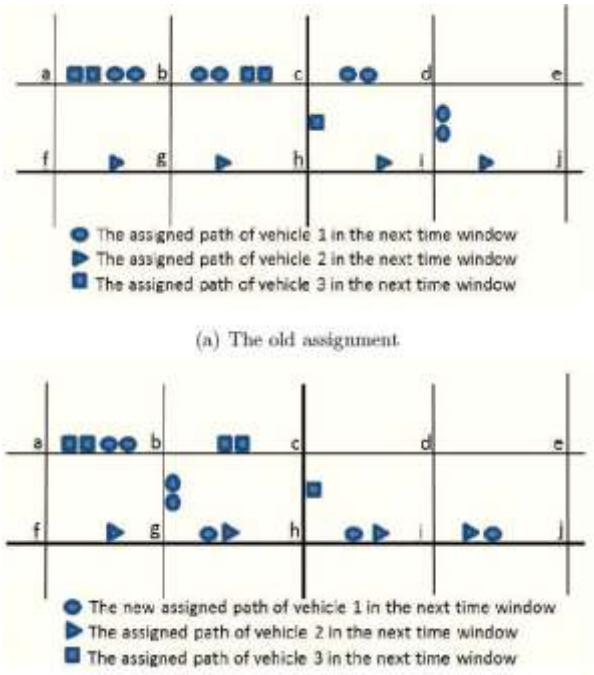


Figure 4.3: A FBkSP example. $\omega_{fg} = \omega_{gh} = \omega_{hi} = \omega_{ij} = \omega_{ch} = 1$, $\omega_{ab} = \omega_{bc} = \omega_{cd} = \omega_{de} = \omega_{af} = \omega_{bg} = \omega_{di} = \omega_{ej} = 2$

To implement the optimization of the total number of footprints in a road network region, a random search strategy (cf. Algorithm 2) is implemented. 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 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).

C. Re-routing Process

In this section, the global re-routing process is presented, which was the basis for the traffic guidance system described in this thesis. 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 and ranks them based on the urgency function. 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 considered here, i.e., improving the individual 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, DTA is employed 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 used here tries to achieve stochastic user equilibrium (SUE) through an iterative simulation process and mathematical modelling (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 employed here was proposed in [49], [36]. 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 these experiments, the maximum number of iterations is defined to 50, since that was the value specified in [50]. The DTA algorithm, as defined in [36], 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, the approach used here prompts 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, this approach is expected to be more efficient than DTA.

V. EVALUATION OF THE RE-ROUTING SYSTEM

The main objective of this simulation-based evaluation is to study the performance of the five re-routing strategies under various scenarios. Specifically, to 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 trade-off 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 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 this software)?

The experimental settings are firstly introduced in Section V-A. Afterwards, the results are presented and analysed in Section V-B.

Table 5.1: Statistics of the Two Road Networks

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

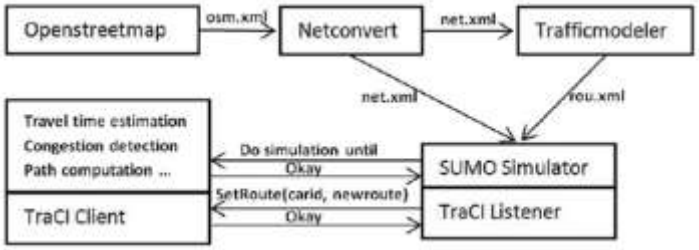


Figure 5.1: The simulation process

A. Experimental Settings

Both SUMO [51] and TraCI [52] were employed for these 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 behaviour of the simulation including vehicle state, road configuration, and traffic lights. The re-routing strategies algorithms were implemented 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.

Two urban road maps were downloaded from OpenStreetMap [53] in osm format. One is a section of Brooklyn, NY and the other is in Newark, NJ. The Netconvert tool in SUMO was used to convert the maps into a SUMO usable format, and the Trafficmodeler tool [54] 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 5.1. 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 5.1 illustrates the simulation process. Figures 5.2 (a) (b) show the traffic flow in both networks. Trafficmodeler was used 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, 908 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.

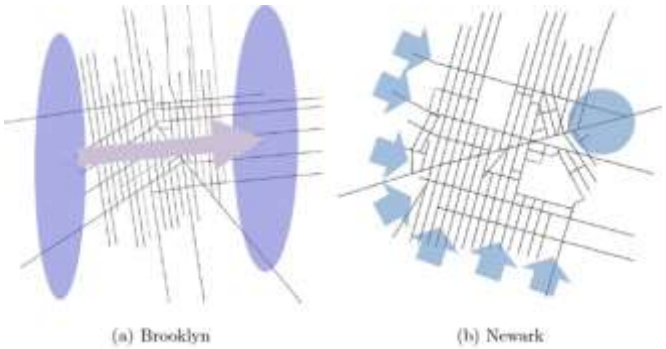


Figure 5.2: Traffic flow in the road networks

Table 5.2: Parameters in Centralized Re-routing Algorithms

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 $\beta = 0.05$

In the simulations, the default settings in SUMO 15.0 were used for vehicle length=5m, the minimal gap=2.5m, the car following model (Krauss [55]), and the driver's imperfection=0.5. For each scenario, the results are averaged over 20 runs. Initially, an ideal scenario is assumed, in which all drivers have the system and accept the route guidance. These assumptions are relaxed in the last part of the evaluation. Table 5.2 defines the parameters used in this evaluation.

A DTA-based re-routing strategy (cf. Section IV-D) was also implemented by using a DTA tool provided with the SUMO generator. Contrary to the proposed approach, the DTA strategy computes the routes leading to user equilibrium for all the vehicles in one shot, before any vehicle starts its journey. Thus, the DTA tool produces a file containing the routes of all the simulated vehicles, which is supplied to the SUMO simulator. Based on this route file, SUMO generates a single, continuous simulation, i.e., without any other route changes as in the case of the proposed strategies. Hence, the CPU time measured in the next section indicates, in the case of DTA, the time required to produce the route file, whereas in the case of the proposed strategies, the cumulated time (i.e., over the whole simulation) required to compute alternative paths for the re-routed vehicles.

B. Results and Analysis

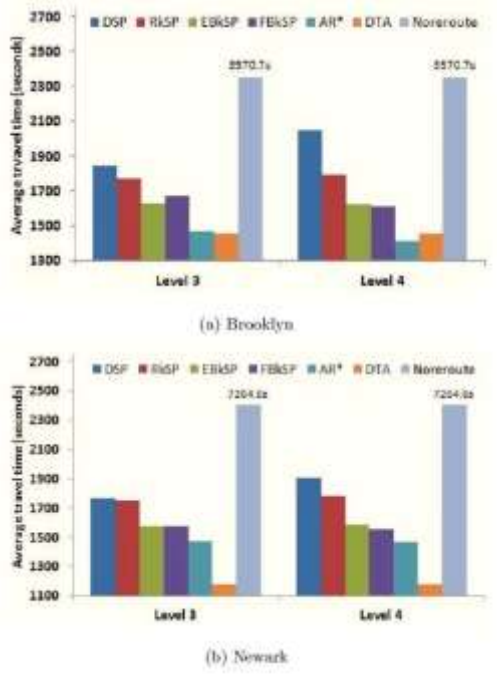


Figure 5.3: Average travel time (L= (3, 4), k=4, urgency=ACI, period=450s, δ =0.7, β =0.05)

1. Average Travel Time

Figure 5.3 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, the five strategies are ranked as follows: DTA>AR*>(EBkSP, FBkSP)>RkSP>DSP>no-rerouting. The results confirm the hypotheses laid out in Section IV with statistical significance of 95% confidence interval. DSP can improve the travel time, since 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.

The 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 5.4). Therefore, the level parameter is set to 3 in the remaining experiments.

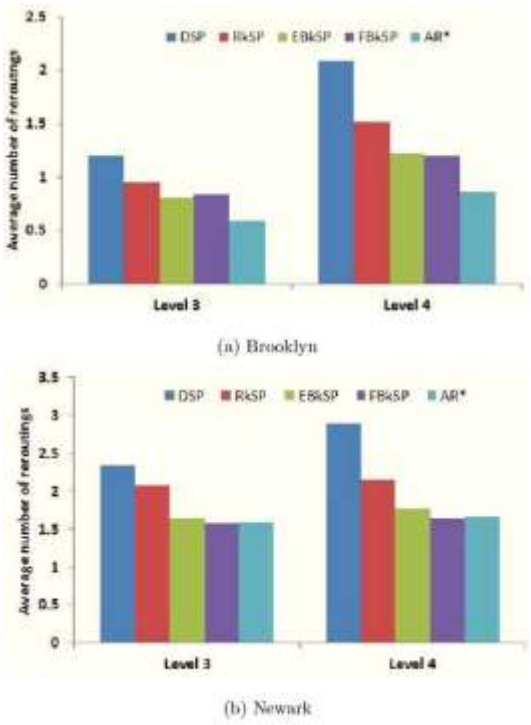


Figure 5.4: Average number of re-routings (L= (3, 4), k=4, urgency=ACI, period=450s, δ =0.7, β =0.05)

2. Average Number of Re-routings

It is important that the re-routings 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 5.4 compares the number of re-routings across the five proposed strategies. The statistical analysis shows that AR^* < (EBkSP, FBkSP) <RkSP<DSP in terms of average rerouting number with 95% confidence interval. For example, 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 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.

3. CPU Time

At this point, the results indicate that AR* produces the best average 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. If the computational complexity of the algorithms that the strategies are based on is considered, the complexities of the Dijkstra shortest path (used by DSP), k loopless shortest paths (used by RkSP, EBkSP and FBkSP) and A^* (used by AR^*) algorithms must be evaluated. Dijkstra shortest

path and k loopless shortest paths require O(E + V log(V)) and O(kV (E + V log(V))), respectively, while A* was proven to be faster than Dijkstra [56]. However, this complexity analysis is pertinent only when the selection of an alternative path for one single vehicle is considered. 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). Moreover, 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.

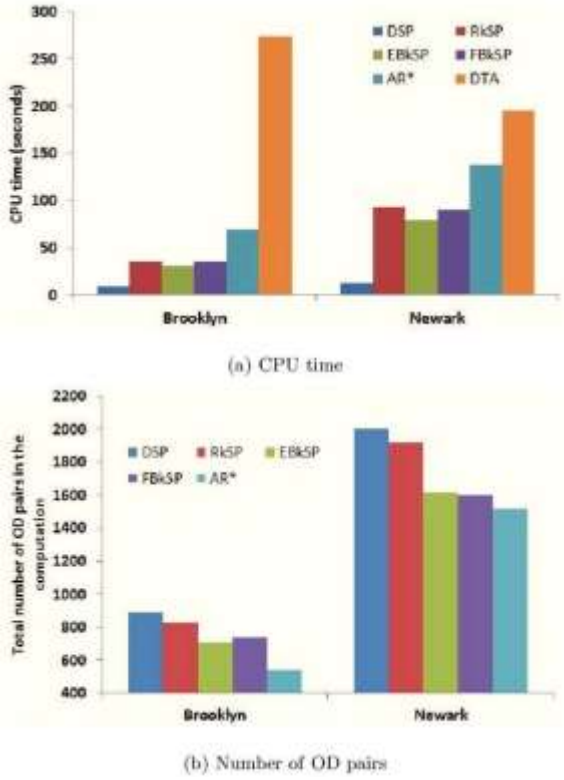


Figure 5.5: CPU time for both the networks (L=3, k=4, urgency=ACI, period=450s, δ =0.7, β =0.05)

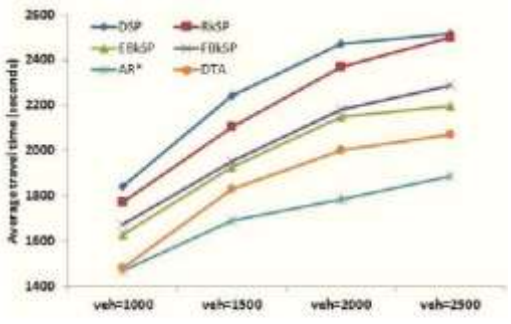
Figure 5.5 (a) shows the global CPU time consumed for re-routing by the five methods and by DTA. Note that the experiments were conducted on a 64 bit Ubuntu machine with Intel Core i5-2467M CPU (1.6GHz) and 4GB of memory. The following four observations were noted regarding the CPU time results:

- DSP requires the least CPU time for re-routing, mainly due to the low complexity of the shortest path algorithm (compared to the k-shortest paths algorithm) and to grouping the re-routed vehicles.
- AR* consumes significantly more CPU time. Specifically, it requires 2.0 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-A.
- 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

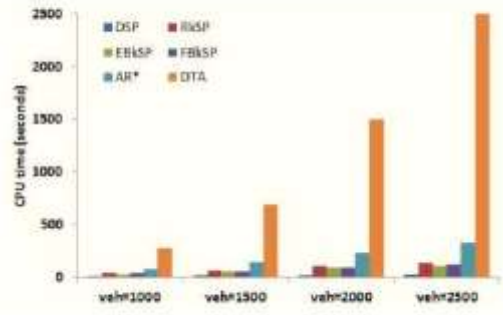
than RkSP even though they execute more complex path selection algorithms in addition to the k-shortest path computation. The explanation is that EBkSP and FBkSP decrease the total number of re-routings processed in a period. This decrease becomes apparent when the number of origin-destination (OD) pairs involved in the computation was examined, as indicated in Figure 5.5 (b). The total number of OD pairs is lower for EBkSP and FBkSP than for RkSP. Figure 5.5 (b) also shows that the although DSP leads to the largest number of OD pairs, it still has the lowest CPU time because of the much lower computational complexity of Dijkstra algorithm compared to the k-shortest path algorithm.

- 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 5.6 (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 both the average travel time and the CPU time are considered, 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 the opposite case, DSP is the most appropriate choice.



(a) Average travel time

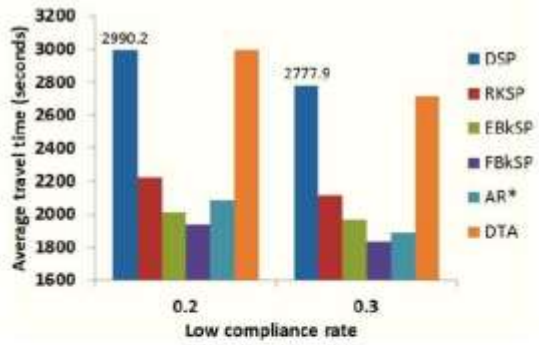


(b) CPU time

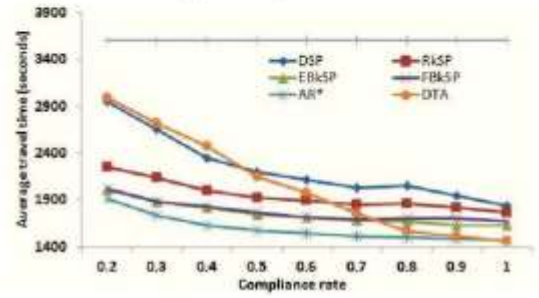
Figure 5.6: The average travel time and CPU time for Brooklyn network for different traffic densities (L=3, k=4, urgency=ACI, period=450s, δ =0.7, β =0.05)

4. 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, another set of experiments were conducted on the Brooklyn network, where the number of vehicles were increased from 1000 to 2500. Figure 5.6 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. It was apparent that DTA exhibits particularly poor scalability compared to the proposed strategies, confirming the 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 5.6 (a)) when the number of vehicles was above 1500. This is certainly due to the fact that the 50 iterations limit, 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.



(a) Low compliance rate



(b) All compliance rates

Figure 5.7: 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)

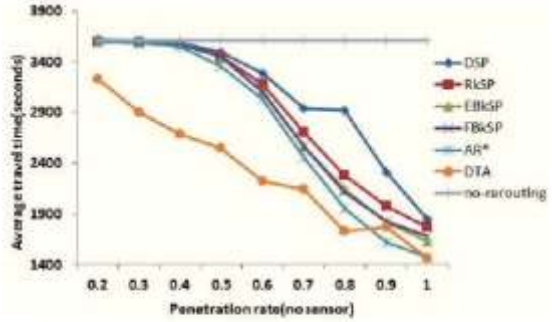
5. Compliance Rate

It is unrealistic to assume that every driver follows the re-routing guidance. The drivers' compliance rate (i.e., the possibility for the driver to accept the guidance) is an important factor for the re-routing strategy design. Therefore, the average travel time was measured, while varying the compliance rate for the five proposed strategies and for DTA. Specifically, for the strategies, given a compliance rate of x%, at each re-routing period, each of the selected vehicles

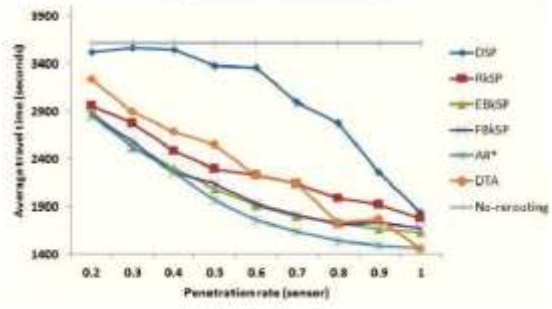
change their routes with x% possibility. As for DTA, x% of the vehicles are randomly selected to follow the DTA assigned route, while the rest of the vehicles follow the shortest time route.

Figure 5.7 (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 5.7 (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 time than DTA. The reason is that when compliance is low, the drivers who comply benefit much more from this 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 user equilibrium, inclusively for the compliant drivers. Differently, these 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.



(a) Sensors: Vehicles Only



(b) Sensors: Vehicles & Road-Side

Figure 5.8: 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)

6. Penetration Rate

To understand how easy is to deploy this solution in real life, the effect of the penetration rate on the average travel time was studied. Specifically, before the system starts, each vehicle was predefined with x% possibility of owning the system(e.g. providing position information and receiving guidance). Once the system starts, only those the x% vehicle will have the chance to be rerouted. The penetration rate is a parameter of major importance for two reasons. First, since only the vehicles which have the system provide position and route information to the server, the

accuracy of congestion detection and road travel time estimation depend directly on this parameter (i.e., the lower the penetration rate is, the lower the accuracy is and vice-versa). Second, similar to the compliance rate, the effectiveness of the load balancing mechanism implemented by the re-routing strategies increases with the percentage of the vehicles that use the system.

Figure 5.8 (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 these methods, DTA performs better under low penetration rates since the DTA re-routing is not triggered by the congestion detection as in this approach.

To boost the adoption of the system, it is possible 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 5.8 (b) demonstrates that these methods can significantly improve the travel time even under low penetration rates if road-side sensor information is available. Moreover, 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. By comparison, these strategies can take advantage of the sensor information to divert the x% of the drivers and to reduce congestion.

VI. CONCLUSION

The ubiquity of mobile devices such as smart phones or on-bord vehicle units is leading to real-life vehicular sensor networks. This chapter presented 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-tailored 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 chapter on a key element of our re-routing system, i.e., the re-routing strategies. We proposed five rerouting 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.

As future work, we intend to investigate a hybrid architecture that off-loads parts of the computation and decision process in the network and uses vehicle-to-vehicle communication to better balance the need for privacy, scalability, and low overhead with the main goal of low average travel time.

1. *How long are American commutes?* 2011; Available from: http://www.economist.com/blogs/freeexchange/2011/10/surveys.
2. *Car-to-car communication*. 2012; Available from: http://www.bmw.com/com/en/insights/technology/technology_guide/articles/cartocar_communication.html.
3. Santo, M. *Toyota, Audi promise driverless car demos at CES 2013*. 2013; Available from: http://www.examiner.com/article/toyota-audi-promise-driverless-car-demos-at-ces-2013.
4. Vijayenthiran, V. *GM Participating In U.S. Car to car Communications Trial*. 2012; Available from: http://www.motorauthority.com/news.
5. Vijayenthiran, V. *Ford Powers Ahead With Development Of Car-To-Car Communication Technology*. 2012; Available from: http://www.motorauthority.com/news.
6. Jiang, D. and L. Delgrossi, *IEEE 802.11 p: Towards an international standard for wireless access in vehicular environments*, in *IEEE Vehicular Technology Conference*. 2008. p. 2036-2040.
7. Biswas, S., R. Tatchikou, and F. Dion, *Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety.* Communications Magazine, IEEE, 2006. **44**(1): p. 74-82.
8. Al-Sultan, S., A. Al-Bayatti, and H. Zedan, *Context Aware Driver Behaviour Detection System in Intelligent Transportation Systems (ITS).* 2013.
9. Zhou, P., et al. *EZCab: A cab booking application using short-range wireless communication*. in *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*. 2005. IEEE.
10. Lu, R., et al. *SPARK: a new VANET-based smart parking scheme for large parking lots*. in *INFOCOM 2009, IEEE*. 2009. IEEE.
11. Riva, O., et al., *Context-aware migratory services in ad hoc networks.* Mobile Computing, IEEE Transactions on, 2007. **6**(12): p. 1313-1328.
12. Hunter, T., et al., *Path and travel time inference from gps probe vehicle data.* NIPS Workshop on Analyzing Networks and Learning with Graphs, 2009.
13. Work, D.B., et al., *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. p. 5062-5068.
14. Eriksson, J., et al., *The pothole patrol: using a mobile sensor network for road surface monitoring*, in *Proceedings of the 6th international conference on Mobile systems, applications, and services (MobiSys 2008)*. 2008, ACM. p. 29-39.
15. Horvitz, E., et al., *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*. 2005. p. 244-257.
16. Mohan, P., 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*. 2008. p. 323-336.
17. Yoon, J., B. Noble, and M. Liu, *Surface street traffic estimation*, in *Proceedings of the 5th international conference on Mobile systems, applications and services*. 2007. p. 220-232.
18. *Inrix*. Available from: http://www.inrix.com/.
19. *Waze*. Available from: http://www.waze.com/.

20. Nadeem, T., et al., *TrafficView: traffic data dissemination using car-to-car communication.* ACM SIGMOBILE Mobile Computing and Communications Review, 2004. **8**(3): p. 6-19.
21. Dornbush, S. and A. Joshi, *StreetSmart traffic: Discovering and disseminating automobile congestion using VANETs*, in *Vehicular Technology Conference, IEEE 65th*. 2007. p. 11-15.
22. Rybicki, J., et al., *Challenge: peers on wheels-a road to new traffic information systems*, in *Proceedings of the 13th annual ACM international conference on Mobile computing and networking*. 2007. p. 215-221.
23. Rybicki, J., et al., *PeerTIS: a peer-to-peer traffic information system*, in *Proceedings of the sixth ACM international workshop on VehiculAr InterNETworking*. 2009. p. 23-32.
24. Senge, S. 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)*. 2010. p. 78-83.
25. Tatomir, B., et al., *Dynamic routing in traffic networks and MANETs using ant based algorithms*, in *Proceedings of the 7th International Conference on Artificial Evolution, Lille, France*. 2005.
26. Prothmann, H., et al., *Decentralized Route Guidance in Organic Traffic Control*, in *Proceedings of the 5th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2011)*. 2011. p. 219-220.
27. Gradinescu, V., et al., *Adaptive traffic lights using car-to-car communication*, in *Vehicular Technology Conference, IEEE 65th*. 2007. p. 21-25.
28. Wardrop, J.G., *Some theoretical aspects of road traffic research.* Proceedings of the Institution of Civil Engineers, Part II, 1952. **1**(36): p. 252-378.
29. Friesz, T.L., et al., *A variational inequality formulation of the dynamic network user equilibrium problem.* Operations Research, 1993: p. 179-191.
30. Merchant, D.K. and G.L. Nemhauser, *Optimality conditions for a dynamic traffic assignment model.* Transportation Science, 1978. **12**(3): p. 200-207.
31. Merchant, D.K. and G.L. Nemhauser, *A model and an algorithm for the dynamic traffic assignment problems.* Transportation Science, 1978. **12**(3): p. 183-199.
32. Peeta, S. and A.K. Ziliaskopoulos, *Foundations of dynamic traffic assignment: The past, the present and the future.* Networks and Spatial Economics, 2001. **1**(3): p. 233-265.
33. Chiu, Y.C., et al., *Dynamic Traffic Assignment: A Primer.* Transportation Research E-Circular, 2011(E-C153).
34. Taylor, N.B. *CONTRAM 5, an enhanced traffic assignment model*. 1990. Transport and Road Research Laboratory, Crowthorne, United Kingdom.
35. Mahmassani, H.S., 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*. 1992: Capri, Italy.
36. Gawron, C., *Simulation-based traffic assignment-computing user equilibria in large street networks*. 1999, University of Cologne, Germany.
37. Maerivoet, S., *Modelling Traffic on Motorways: State-of-the-Art, Numerical Data Analysis, and Dynamic Traffic Assignment*. 2006, Katholieke Universiteit Leuven.
38. Banks, J.H., *Introduction to transportation engineering*. 2002: McGraw-Hill: New York.
39. Manual, H.C. *Highway capacity manual*. 2000. Washington, DC.

40. Lawler, E.L., *A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem.* Management Science, 1972: p. 401-405.
41. Martins, E.Q.V. and M.M.B. Pascoal, *A new implementation of Yen's ranking loopless paths algorithm.* 4OR: A Quarterly Journal of Operations Research, 2003. **1**(2): p. 121-133.
42. Perko, A., *Implementation of algorithms for K shortest loopless paths.* Networks, 1986. **16**(2): p. 149-160.
43. Fredman, M.L. and R.E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms.* Journal of the ACM (JACM), 1987. **34**(3): p. 596-615.
44. Hart, P.E., N.J. Nilsson, and B. Raphael, *A formal basis for the heuristic determination of minimum cost paths.* IEEE Transactions on Systems Science and Cybernetics, 1968. **4**(2): p. 100-107.
45. Shannon, C.E., *A mathematical theory of communication.* ACM SIGMOBILE Mobile Computing and Communications Review, 2001. **5**(1): p. 3-55.
46. Cranshaw, J., et al., *Bridging the gap between physical location and online social networks*, in *Proceedings of the 12th ACM international conference on Ubiquitous computing*. 2010. p. 119-128.
47. Xu, T. and Y. Cai, *Feeling-based location privacy protection for location-based services*, in *Proceedings of the 16th ACM conference on Computer and communications security*. 2009. p. 348-357.
48. Hoos, H.H. and T. Stützle, *Stochastic local search: Foundations and applications*. 2005: Access Online via Elsevier.
49. *DuaIterate*. Available from: http://sumo.sourceforge.net/doc/current/docs/userdoc/Tools/Assign.html.
50. Behrisch, M., D. Krajzewicz, and Y.P. Wang, *Comparing performance and quality of traffic assignment techniques for microscopic road traffic simulations*, in *Proceedings of international symposium on dynamic traffic assignment (DTA 2008)*. 2008: Leuven, Belgium.
51. Behrisch, M., et al., *SUMO - Simulation of Urban Mobility: An Overview*, in *Proceedings of the 3rd International Conference on Advances in System Simulation (SIMUL 2011)*. 2011: Barcelona, Spain.
52. Wegener, A., et al., *TraCI: an interface for coupling road traffic and network simulators*, in *Proceedings of the 11th communications and networking simulation symposium*. 2008. p. 155-163.
53. Haklay, M. and P. Weber, *OpenStreetMap: User-generated street maps.* IEEE Pervasive Computing, 2008. **7**(4): p. 12-18.
54. Papaleondiou, L.G. and M.D. Dikaiakos, *Trafficmodeler: A graphical tool for programming microscopic traffic simulators through high-level abstractions*, in *Vehicular Technology Conference, IEEE 69th*. 2009. p. 1-5.
55. Krauss, S., P. Wagner, and C. Gawron, *Metastable states in a microscopic model of traffic flow*. Physical Review E, 1997. **55**(5): p. 5597.
56. Schultes, D., *Route planning in road networks.* Karlsruhe: Universität Karlsruhe (TH). Fakultät für Informatik. Institut für Theoretische Informatik, Algorithmik II, 2008.