A Randomized Path Routing Algorithm for Decentralized Route Allocation in Transportation Networks

Uyen T. V. Nguyen
Dept. of Computing and Information Systems
University of Melbourne
utnguyen@unimelb.edu.au

Shanika Karunasekera
Dept. of Computing and Information Systems
University of Melbourne
karus@unimelb.edu.au

Lars Kulik
Dept. of Computing and Information Systems
University of Melbourne
lkulik@unimelb.edu.au

Egemen Tanin
Dept. of Computing and Information Systems
University of Melbourne
etanin@unimelb.edu.au

Rui Zhang
Dept. of Computing and Information Systems
University of Melbourne
rui.zhang@unimelb.edu.au

Haolan Zhang
Ningbo Institute of Technology
Zhejiang University
haolan.zhang@nit.zju.edu.cn

Hairuo Xie
Dept. of Computing and Information Systems
University of Melbourne
xieh@unimelb.edu.au

Kotagiri Ramamohanarao
Dept. of Computing and Information Systems
University of Melbourne
kotagiri@unimelb.edu.au

ABSTRACT

Route planning in current route guidance systems is primarily based on the shortest path strategy (e.g., suggesting a path that minimizes the travel distance between source and destination) for an individual driver. A drawback of such an approach is that there is a high probability that many drivers, who share the same source and destination, follow the same route. This can lead to over-saturated road links. We propose a simple yet effective multiple path routing algorithm that addresses this limitation. The proposed algorithm augments the A* search algorithm with a randomization component, which helps to perturb the order of the searched nodes and to produce diversified routes for different drivers with the same source and destination. Our algorithm enables decentralized route allocations, i.e., each vehicle computes its own route without a central server. We validate the performance of the proposed algorithm through microscopic traffic simulation. Experiments for various traffic scenarios show that our algorithm achieves a good path diversification while reducing the average travel time compared to the traditional shortest path algorithm.

1. INTRODUCTION

Modern route planning systems predominantly use a shortest path strategy to generate route plans. This strategy suggests a path with the lowest travel cost, which can be determined by travel distance and other factors, such as the time-dependent travel times [1, 2], the engine type of vehicles [3], the traffic emissions [4], the possibility of traffic hazards [5], the history of route choices [6] and the combination of multiple factors [7]. A potential problem with many of the existing systems is that different drivers, who share the same source and destination, are likely to be allocated with the same path. This can lead to traffic congestions if all the drivers travel on the suggested path at the same time. To demonstrate this, Figure 1 shows an example that a number of cars travel between two small areas (SRC and DST) in Melbourne, Australia. If all the routes are the shortest paths, there would be a large number of road links shared by all cars (enclosed by the dashed line). The over-saturation of traffic on the shared road links may cause traffic congestions in a large area, which should be avoided in a smart route guidance system [8]. One way to address the problem is providing multiple paths for different drivers with the same source and destination.

Recent research focused on multiple path routing in building evacuations [9] and road traffic [10, 11]. In [10], Lee et. al. propose an approach that uses a central server to compute multiple candidate paths for a specific pair of source and destination. Vehicles with the same source and destination are likely to be allocated with different paths. However, this approach can cause computation and communication bottlenecks if the number of queries to the server is high. In [11], Adacher et. al. propose a decentralized approach, in which each vehicle independently computes its own route with certain perturbation of the road network data, resulting in the diversification of paths. An advantage of this technique is that shortest paths are computed locally, which prevents a high level of computation workload at a central server. However, individual vehicles need to retrieve the traffic condition of each candidate road link from nearby servers for comput-

IWCTS'15, November 03-06, 2015, Bellevue, WA, USA
 ISBN 978-1-4503-3979-7/15/11...$15.00
DOI: http://dx.doi.org/10.1145/2834882.2834886

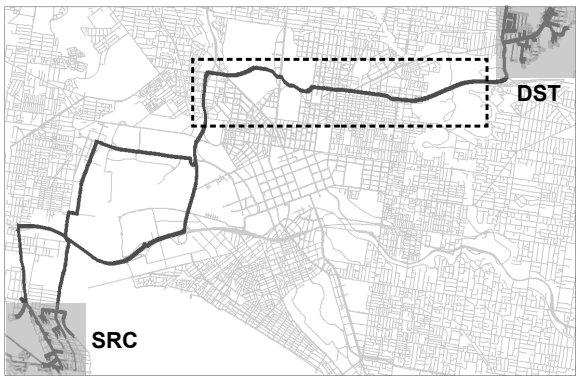

Figure 1: An example showing the limitation of the shortest path strategy: 1000 cars travel from an area (SRC) to another area (DST) in Melbourne, Australia. The bold lines indicate the routes generated by Dijkstra's algorithm. The region of potential traffic congestions is enclosed by a dashed line.

ing routes. This can be highly costly in terms of communication if there are a large number of candidate road links.

We propose a simple yet effective multiple path routing algorithm that can overcome the limitations of existing techniques. Our proposed algorithm is an extended and randomized version of the A* search algorithm. It has the following advantages:

1. It is decentralized, i.e., each vehicle computes its own route without the need for a central server;
2. It produces high quality routes, i.e., the travel cost of the suggested and the corresponding routes are close;
3. It provides excellent diversification of routes.

2. BACKGROUND

Shortest path problem. A *road network* can be modelled as a *directed graph* $G = (V, E, W)$, where a vertex (i.e., a node) $v \in V$ represents a road intersection, an edge $e \in E$ represents a road segment connecting two nodes, and a weight $w \in W$ represents the travel cost of an edge, e.g., travel time or travel distance. Given two nodes, s and t, the shortest path problem finds the path with the lowest cost between s and t.

Dijkstra's algorithm. A classical algorithm to solve the shortest path problem is Dijkstra's algorithm [12]. Dijkstra's algorithm works by scanning the nodes in an ascending order of their distance to the source node s. All the nodes are maintained in a min-priority queue Q, in which the distance to the source is the sorting key. Each node is associated with a score, $d_s(u)$, which is the distance of the shortest path from s to u. Each node also keeps the predecessor node in optimal path. At the start of the algorithm, $d_s(u)$ of each node, except the source node, is set to an infinite value and the previous node is set to an empty value. At each iteration, the algorithm first extracts the node $u \in Q$ with the minimum $d_s(u)$. Then for any edge $(u, v) \in E$, the algorithm checks whether $d_s(v)$ can be reduced if the path is through u. If this is true, the score of v is updated and the predecessor of v is set as u. The algorithm terminates as soon as t is extracted from Q.

As Dijkstra's algorithm scans all the nodes that are closer to s than t, its search space can be approximated by a circle centred on s with the radius equal to the distance from s to t. The size of the search space can have a significant impact on the efficiency of computation [13]. Certain variations of the algorithm help to reduce the search space by goal-directed techniques that guide the search toward the destination and skip the nodes that are not on the direction from s to t.

A* search algorithm. A classic goal-directed shortest path algorithm is A* [14]. Given a source s and a target t, A* search is similar to Dijkstra's algorithm except that the score of each node $v \in Q$ is defined as $f(v)$, which is the length of the best path from s to t through node v. $f(v)$ is computed as the sum of two values (Figure 2). One is the length of the shortest path from s to v, $d_s(v)$. Another is the estimated length of path from v to t, which is computed from a heuristic function, $h_t(v)$. At each step, the A* algorithm chooses the most promising node (i.e., node with the smallest $f(v)$) to extract next. This process is repeated until t is extracted from the priority queue.

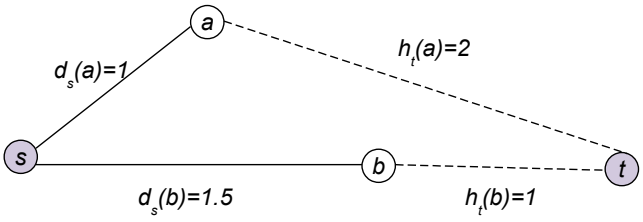

Figure 2: Demonstration of A* search. There are two possible paths from node s to node t, one is through node a, another is through node b. In this example, $f(a) = d_s(a) + h_t(a) = 1 + 2 = 3$ and $f(b) = d_s(b) + h_t(b) = 1.5 + 1 = 2.5$. Therefore, b is extracted before a during the search.

By introducing the heuristic function, the search is goal-directed and pulls faster to the destination. Even though being a heuristic search, A* algorithm is guaranteed to find an optimal solution as long as h_t is an *admissible* heuristic function, or never overestimates the actual distance between two nodes. The tighter the estimate is, the faster the search is toward the destination. However, A* algorithm is not efficient for road networks (especially if the travel cost is measured based on travel time) as the estimated bounds are poor and the performance gain can be marginal [15].

Overdo A*. Overdo A* algorithm is a modified version of A* in which the *admissibility* of the heuristic function is no longer guaranteed [16]. Overdo A* works by increasing the contribution of the second component, h_t, in calculating the score of a vertex v. This multiplicative factor is called *overdo* parameter [16]. Assuming the *overdo* parameter is k, the score of a node v is computed as $f(v) = d_s(v) + k \times h_t(v)$. Intuitively, this corresponds to giving the destination a high potential and biasing the search towards the destination faster. When k equals to one, the search becomes A* search. With higher k values, the search space is reduced substan-

tially but the optimal solution is not guaranteed to be found. However, empirical studies show that the Overdo A* search runs much faster while the solution quality is acceptable from a practical point of view [16].

3. RANDOM A* SCALING ALGORITHM

We propose Random A* Scaling algorithm (RandomAS), an extension and a randomized version of A* and Overdo A*. The details of RandomAS are described in Algorithm 1. RandomAS uses a priority queue Q to manage a list of nodes to be processed next. At first, only the source node s is in the queue. At each iteration, the node with the minimum score is extracted from Q. The score of a node $v \in Q$ is a weighted combination between $d_s(v)$, which is the shortest distance from s to v, and $h_t(v)$, which is the estimated distance from v to target t. The combination is specified as $f(v) = d_s(v) + k \times h_t(v)$, where k is the overdo parameter randomly selected from a range, $[1, k_{max}]$ (k_{max} is the only parameter of RandomAS). Whenever a node u is extracted from Q, the overdo parameter k is randomly selected from the specified range. The scores of the nodes in Q are then updated with the new k value. Each edge $(u, v) \in E$ is then checked and v is put into Q, if v is not already in Q and has not been extracted from Q. This process is repeated until the destination node is extracted from Q.

By assigning the overdo parameter k to a random value in a specified range and changing it over time, the order of the nodes to be extracted from the queue is perturbed and this helps to randomize the route generated by the algorithm in each run. To demonstrate how RandomAS works, Figure 3 shows an example graph. In this example, the measurement of distances is based on Euclidean distance. There are three different routes from A to F: $R_1 = (A, B, F)$, $R_2 = (A, C, F)$ and $R_3 = (A, D, E, F)$ with the route length $l(R_1) = 14$, $l(R_2) = 15$, and $l(R_3) = 17$. We assume that $k_{max} = 2$. At first, the source node A is put into the queue Q. When A is extracted from Q, the overdo parameter, k, is set to a random value between 1 and 2. Let us assume that k is set to 1.6. The three successor nodes, B, C and D, are then put into Q with the corresponding scores, $f(B) = 5 + 1.6 * 9 = 19.4$, $f(C) = 8 + 1.6 * 7 = 19.2$ and $f(D) = 5 + 1.6 * 10 = 21$. As C is the node with the lowest score, C is extracted in the next iteration. Assuming k is set to 1.2 when C is extracted, the scores of the remaining nodes in Q are updated as $f(B) = 5 + 1.2 * 9 = 15.8$ and $f(D) = 5 + 1.2 * 10 = 17$. The successor node of C, F, is then put into Q with the score $f(F) = 15$. As F has the lowest score in Q, it is extracted in the next iteration and the algorithm terminates. Therefore, the final route is R_2. We should note that other routes may be returned at different running times with different k values.

To demonstrate how RandomAS works on real road networks, we use RandomAS to generate routes for the same scenario as shown in Figure 1. The results are shown in Figure 4. We can see that the routes suggested by RandomAS are significantly more diversified, compared with the routes suggested by Dijkstra's algorithm. This can help to prevent the heavy use of certain shared road links as observed in Figure 1. In addition, the routes returned by RandomAS are still close to the optimal routes.

Algorithm 1 Random A* Scaling algorithm

Input: Road network $G_{V,E,W}$, source s, destination t, k_{max}
Output: A route from s to t

```
1: Q ← ∅, Q' ← ∅
2: for vertex v ∈ V do
3:    d_s(v) ← ∞; h_t(v) ← ∞; f(v) ← ∞
4: end for
5: d_s(s) ← 0; h_t(s) ← 0; f(s) ← 0;
6: Q.Insert(s);
7: while ¬ Q.Empty() do
8:    u ← Q.ExtractMin();
9:    Q'.Insert(u);
10:   if u ⇔ t then
11:      return;
12:   end if
13:   k ← random(1, k_max);
14:   for vertex v ∈ Q do
15:      f(v) ← d_s(v) + k × h_t(v);
16:   end for
17:   for edge e_{u,v} ∈ E do
18:      if v ∈ Q' then
19:         continue;
20:      end if
21:      if v ∉ Q then
22:         Q.Insert(v);
23:      end if
24:      d_s(v)' ← d_s(u) + weight_e
25:      h_t(v)' ← EstimateCost(v, t)
26:      f(v)' ← d_s(v)' + k × h_t(v)'
27:      if f(v)' < f(v) then
28:         d_s(v) ← d_s(v)'
29:         h_t(v) ← h_t(v)'
30:         f(v) ← f(v)'
31:         prev(v) ← u;
32:      end if
33:   end for
34: end while
```

4. EXPERIMENTS

In order to evaluate the performance of RandomAS, we conduct experiments using the real road network data of Melbourne, Australia. The road network is extracted from OpenStreetMap (`https://www.openstreetmap.org/`), covering an $15km \times 10km$ area of Melbourne with 38278 nodes and 63920 edges. The performance of our method is compared against Dijkstra's algorithm using various evaluation measures. All algorithms are implemented in Java.

4.1 Evaluation measures

Three measures, the route ACCuracy (ACC), the Road Usage Index (RUI) and the Simulated Travel Time (STT) are used in the experiments. The details of each measure are described in the following subsections.

4.1.1 Route accuracy

The accuracy of a route R is defined as:

$$ACC(R) = \frac{l(R_{Opt})}{l(R)} \quad (1)$$

where $l(R_{Opt})$ is the length of the optimal route and $l(R)$ is the length of R. The route accuracy achieves the maximum

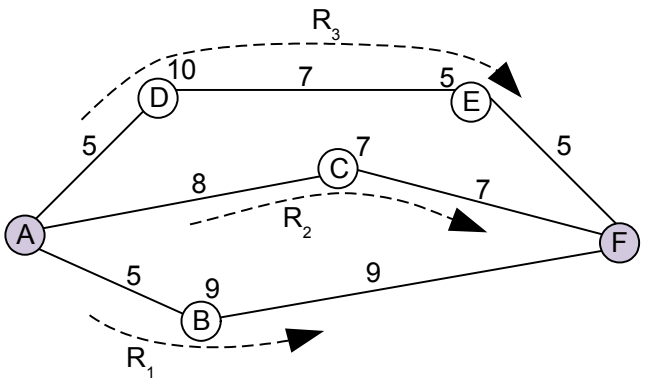

Figure 3: A demonstration of RandomAS. A is the source and F is the destination. The number attached to a node shows the distance between the node and the destination. The numbers attached to the edges show the length of the edges. R_1, R_2 and R_3 are the three possible routes.

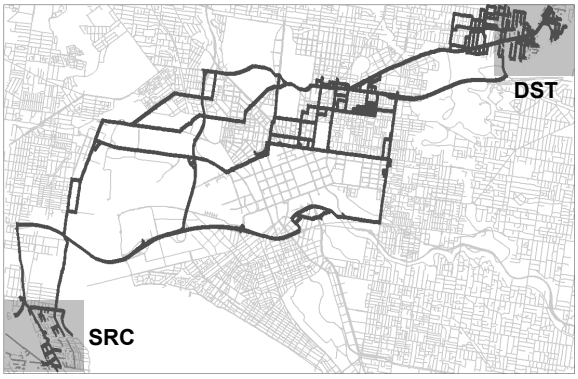

Figure 4: RandomAS routes for the same set of vehicles in the scenario presented in Figure 1.

value, 1, when the length of R is equal to the length of the optimal route.

4.1.2 Road usage index

Given a set of routes, $S_R = \{R_1, R_2, ..., R_m\}$, generated by RandomAS and a set of routes, $S_D = \{D_1, D_2, ..., D_n\}$, generated by Dijkstra's algorithm, the road usage index measures the ratio of the aggregated route length between the two sets of routes:

$$RUI(S_R) = 1 - \frac{Road\ usage\ length(S_D)}{Road\ usage\ length(S_R)} \quad (2)$$

where the *road usage length* is computed as the sum of the lengths of the distinct road links used by all the routes in the specified set. If different routes use different roads, the traffic can be well distributed, i.e., there is no common road shared by a large number of vehicles. Therefore, the road usage index provides a measure of path diversification. A higher value means a higher diversification of routes.

Table 1: Average route accuracy (ACC) and road usage index (RUI) of RandomAS on 1000 source-destination pairs.

k_{max}	1.5	2	3	4	5
ACC	0.99	0.97	0.94	0.93	0.91
RUI	0.53	0.66	0.75	0.79	0.81

4.1.3 Simulated travel time

In addition to the static measures such as route accuracy and road usage index, we employ a microscopic traffic simulator to simulate and measure the travel time of vehicles in various realistic scenarios. The traffic simulator implements a number of traffic models to simulate realistic driver behaviour, such as car-following and lane-changing. The simulated vehicles obey various road rules, such as give-way rules at roundabouts and intersections. Traffic lights are also simulated.

4.2 Parameter setting

We examine the impact of the input parameter, k_{max}, on the quality of the routes generated by RandomAS. We randomly select 1,000 source-destination pairs and run RandomAS 100 times for each pair. Table 1 shows the average route accuracy (ACC) and the road usage index (RUI) of RandomAS routes when k_{max} increases from 1.5 to 5. We can see that there is a slight decrease in ACC when k_{max} increases. ACC in all the cases is above 0.9. RUI increases with the increase of k_{max}, which means that we can get a better route distribution with higher k_{max} values. To achieve a good balance between ACC and RUI, k_{max} is set as 2 in all of the following experiments.

4.3 Experimental results

We compare RandomAS with the shortest path strategy in various realistic scenarios using the microscopic traffic simulator (Section 4.1.3). The routes of vehicles are generated with the shortest path strategy (i.e., Dijkstra's algorithm) and the RandomAS algorithm. The average travel time of vehicles is collected from the simulations.

4.3.1 Single pair test case

We simulate the scenario, where a number of cars travel from source A to destination B. Figure 5 (a) and 5 (b) show the routes generated by Dijkstra and RandomAS, respectively when the number of cars is 1000.

If all the cars use the shortest path strategy, they follow the same route, which is highly likely to cause traffic congestions. In contrast, if RandomAS is used, the cars are assigned to diversified paths, which can help to prevent the formation of traffic congestion. Figure 6(a) shows the average travel time for each strategy when the number of cars increases from 50 to 2,000. When the number of cars is small (i.e., 50 or 100 cars), the average travel times of the two strategies are similar. However, when the number of cars increases, the cars following RandomAS routes arrive their destinations in a significantly shorter time, compared with the cars following Dijkstra routes. When there are 2000 cars, RandomAS

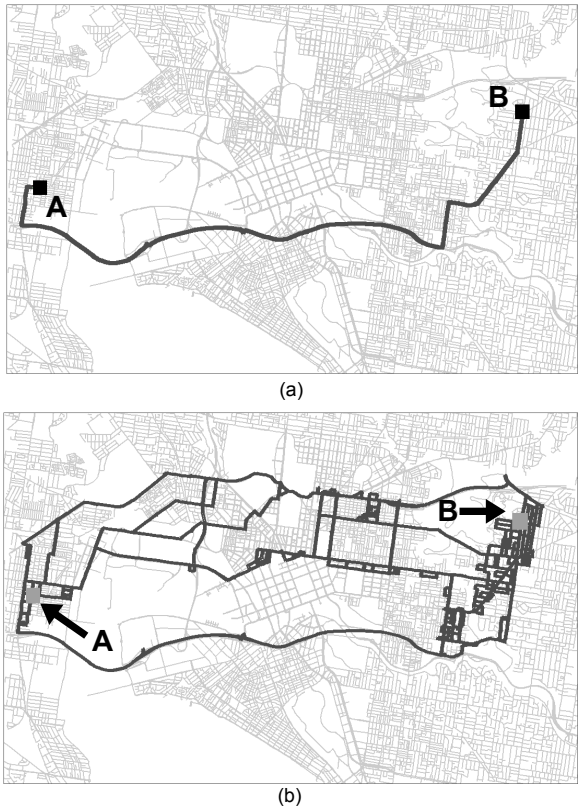

Figure 5: (a) Dijkstra and (b) RandomAS routes for 1000 cars from A to B. The average ACC and RUI of RandomAS routes are 0.94 and 0.87, respectively.

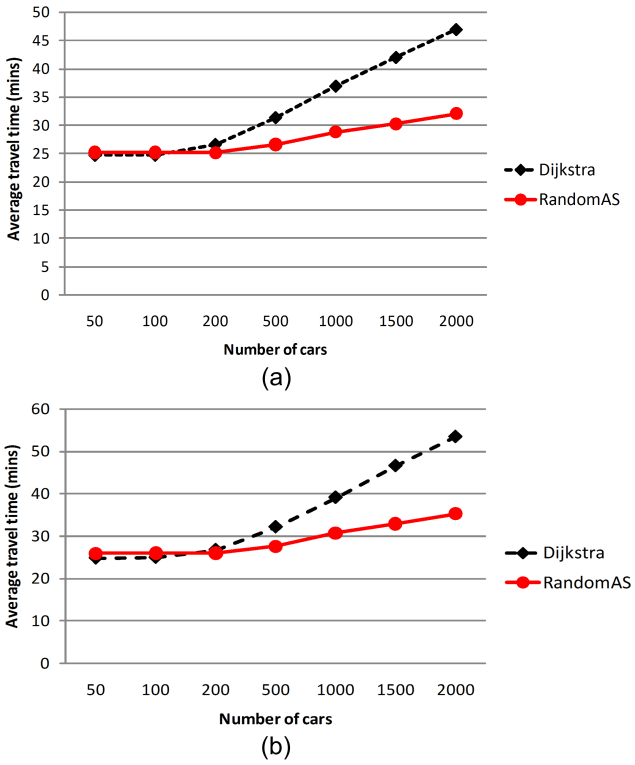

Figure 6: Average travel time vs. number of cars following Dijkstra routes and RandomAS routes on (a) an empty road network; (b) a road network with 5000 random background cars. The gap between the two types of routes is larger in (b) than (a).

can reduce the travel time from the shortest path strategy by more than 30%.

To simulate more realistic situations, we randomly generate 5000 background cars, in addition to the cars that travel between the specified source and destination. The background cars are generated with random sources and destinations. When a background car reaches its destination, a new background car is generated. In other words, the number of background cars remains constant throughout a simulation. The results in Figure 6(b) shows the advantage of RandomAS in this scenario. RandomAS can reduce the travel time from the shortest path strategy by more than 30% when the number of cars is at the highest level.

4.3.2 Full scenario test case

In this experiment, we simulate the scenario, where a number of cars travel toward Melbourne CBD from eight locations around the area during peak hours. Figure 7(a) and 7(b) show the routes generated by Dijkstra and the routes generated by RandomAS, when there are 1000 cars starting from each of the locations.

Figure 8 shows the average travel time of the cars that follow Dijkstra routes and RandomAS routes when the number of cars that start from each location varies between 50 and 1500. There are 10000 additional random background cars during the simulations. The results show that there is a negligible difference in the travel time between the two algorithms when the number of cars starting from the locations is small (i.e., 50 or 100 cars per location). However, as the number of cars increases, RandomAS shows its advantage over the shortest path strategy with a significant reduction of the average travel time. This is due to the fact that RandomAS distributes the cars over different paths in the road network and prevents the formation of traffic congestions on the road segments that are shared by a large number of cars. When there are 1500 cars starting from each of the locations, RandomAS can reduce the travel time from the shortest path strategy by 30%.

5. CONCLUSION

RandomAS is a novel multiple path routing algorithm that diversifies routes in transportation networks. RandomAS is more advanced than other multiple path algorithms as it is easy to implement and is fully decentralized. The algorithm enables a natural mechanism to reduce traffic congestions. Our experiments show that RandomAS can reduce the average travel time of vehicles by more than 30% from the shortest path algorithm while the randomized routes are close to the optimal shortest paths.

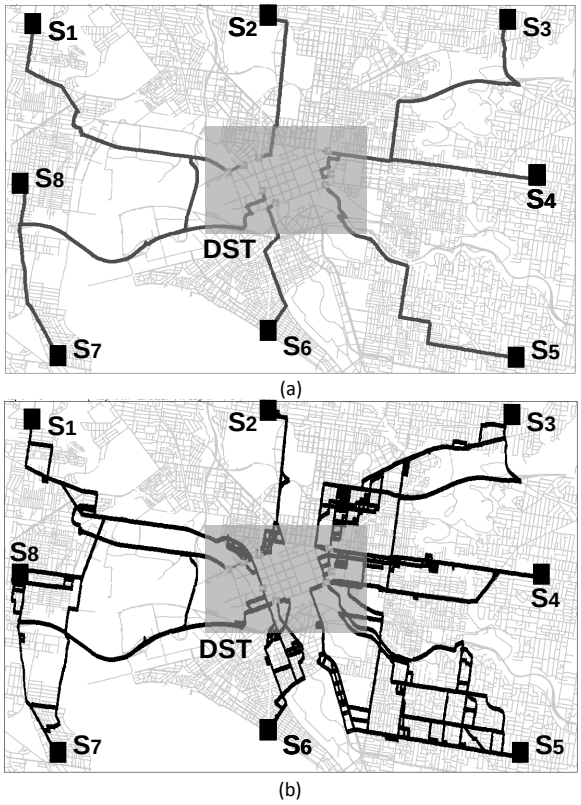

Figure 7: (a) Dijkstra routes and (b) RandomAS routes for the full scenario test case. There are 1000 **cars starting from each of the eight locations,** S_1 **to** S_8**. The destination area is DST. The average ACC and RUI of RandomAS routes are** 0.98 **and** 0.72**, respectively.**

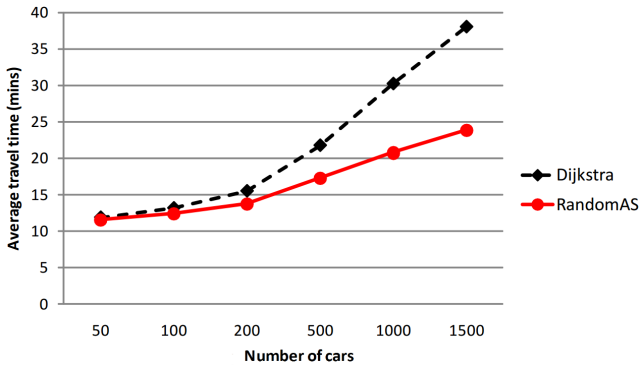

Figure 8: Average travel time of the cars starting from eight locations (Figure 7) vs. number of cars starting from each of the locations. In addition to the cars starting from the specified locations, the traffic network also contains 10000 **background cars with random sources and destinations.**

6. REFERENCES

[1] Jan Fabian Ehmke, André Steinert, and Dirk Christian Mattfeld. Advanced routing for city logistics service providers based on time-dependent travel times. *Journal of computational science*, 3(4):193–205, 2012.

[2] Kuilin Zhang, Hani Mahmassani, and Chung-Cheng Lu. Probit-based time-dependent stochastic user equilibrium traffic assignment model. *Journal of the Transportation Research Board*, (2085):86–94, 2008.

[3] Sabine Storandt. Quick and energy-efficient routes: computing constrained shortest paths for electric vehicles. In *IWCTS*, pages 20–25. ACM, 2012.

[4] Andreas Gazis, Tânia Fontes, Jorge Bandeira, Sérgio Pereira, and Margarida C Coelho. Integrated computational methods for traffic emissions route assessment. In *IWCTS*, pages 8–13. ACM, 2012.

[5] Thomas Liebig, Nico Piatkowski, Christian Bockermann, and Katharina Morik. Route planning with real-time traffic predictions. In *LWA Workshops: KDML, IR and FGWM*, pages 83–94, 2014.

[6] Ling-Yin Wei, Yu Zheng, and Wen-Chih Peng. Constructing popular routes from uncertain trajectories. In *SIGKDD*, pages 195–203. ACM, 2012.

[7] Yi-Hwa Wu, Harvey J Miller, and Ming-Chih Hung. A GIS-based decision support system for analysis of route choice in congested urban road networks. *Journal of Geographical Systems*, 3(1):3–24, 2001.

[8] Jeffrey L Adler and Victor J Blue. Toward the design of intelligent traveler information systems. *Transportation Research Part C: Emerging Technologies*, 6(3):157–172, 1998.

[9] Qingsong Lu, Yan Huang, and Shashi Shekhar. Evacuation planning: a capacity constrained routing approach. In *Intelligence and Security Informatics*, pages 111–125. Springer, 2003.

[10] Chi-Kang Lee. A multiple-path routing strategy for vehicle route guidance systems. *Transportation Research Part C: Emerging Technologies*, 2(3):185–195, 1994.

[11] Ludovica Adacher, Marta Flamini, and Gaia Nicosia. Decentralized algorithms for multiple path routing in urban transportation networks. In *Triennial Symposium on Transportation Analysis*, 2007.

[12] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[13] Alexandros Efentakis, Dieter Pfoser, and Agnès Voisard. Efficient data management in support of shortest-path computation. In *IWCTS*, pages 28–33. ACM, 2011.

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

[15] Andrew V Goldberg and Chris Harrelson. Computing the shortest path: a search meets graph theory. In *ACM-SIAM symposium on Discrete algorithms*, pages 156–165. SIAM, 2005.

[16] Riko Jacob, Madhav Marathe, and Kai Nagel. A computational study of routing algorithms for realistic transportation networks. *Journal of Experimental Algorithmics*, 4:6, 1999.