2017 IEEE Intelligent Vehicles Symposium (IV)
June 11-14, 2017, Redondo Beach, CA, USA

Near-Future Traffic Evaluation based Navigation for Automated Driving Vehicles

Kuen-Wey Lin, Yih-Lang Li and Masanori Hashimoto

Abstract—**Once vehicles start to be driven automatically, people expect the driving routing is automatically and optimally selected. Supposing all the vehicles are navigated by a single system in the future, the navigation system will be able to provide instructions to each vehicle based on the evaluated near-future traffic information while the current navigation system frequently updates the routing based on current traffic information. This paper proposes a navigation method that guides vehicles based on the evaluated near-future traffic information. Experimental results with actual city maps show the evaluated near-future traffic information is helpful to mitigate traffic jam and reduce driving time.**

I. INTRODUCTION

With the global positioning system (GPS) system, an on-board navigation device for individual vehicles provides a shortest or fastest route based on the current traffic conditions for drivers. Moreover, as a leading navigation service provider, Google Maps [17] can offer history/real-time traffic information and route planning. These devices and services are designed to consider local optimality, *i.e.*, each user can query his optimal route according to the traffic information on the map at the time when his query is issued. Since each user does not know the routes of the other users, traffic jam that is going to happen cannot be known in advance neither. Thus, some research papers focusing traffic assignment problem or multi-agent routing problem have been proposed. A cooperative car navigation system [1][2] receives route information (current position, destination, and the planned path to the destination) from each vehicle and estimates the probability of traffic jam on each road. Then, each vehicle re-plans its route based on the information from the system by itself. Based on current traffic measurement and historical traffic data stored in a vehicle, an on-board route guiding system was proposed in [3]. Wang *et al.* [4] investigated route guidance in large-scale express ring-roads via simple reaction to real-time traffic measurements. With the designated service spots and tour plans from each user, Kuriyama *et al.* [5] presented a scheduling technique that iteratively removes the least important spots such that the modified plans can satisfy the time constraint from each user. A multi-agent scheme for controlling transportation networks is proposed by Negenborn *et al.* [6]. The system to be controlled is divided into some sub-systems, and each sub-system is assigned to an agent. Using model predictive control, Negenborn *et al.* [6] focus on how these agents perform communication between them and local computations to improve decision making. In the problem formulation of [7], a set of target points generated over time should be visited by a set of mobile agents. The authors of [7] proposed a multi-agent system to minimize the time between the appearance of a target and the time it is visited by one of the agents. Li *et al.* [8] presented a route guidance system to plan one route for a particular vehicle with the real-time traffic information. In [9][10], an energy-oriented navigation system is designed for hybrid electrical vehicle to find an energy optimal route that reduces gas emission. Wang *et al.* [11] developed a multi-agent based vehicle routing system to provide the optimal next turns to bypass the congested roads. After that, each vehicle uses its own on-board navigation device to find the new path to complete the rest of its journey. They focus on updating the routes only after the occurrence of a traffic event. Miao *et al.* [12] study the problem of taxi dispatch. With the occupancy status and location of each taxi, they predict the demand of passenger to regulate the mobility of idle taxis to balance the ratio of supply and demand. Dynamic traffic assignment (DTA) [15] model is used to represent the interaction between travel choices, traffic flows, and time and cost measures using a behaviorally sound approach.

In this paper, we propose an integrated navigation system for automated driving vehicles in the world as automated driving technologies become mature and popular. According to the game-theoretic point of view, each agent is interested in maximizing its own utility, and all agents will maximize a common global utility function. However, in this paper, each vehicle needs to *make a compromise with the near-future traffic conditions evaluated by the proposed algorithm*. The idea of compromise is the core of the algorithm. Unlike multi-agent system, in this paper, a query for route search issued by a vehicle is generated randomly and removed after reaching the destination. A vehicle does not receive information from other vehicles. To validate our idea, we compare two types of navigation systems to highlight the advantages of integrated navigation system with *near-future traffic evaluation* to diminish cruising time and mileage of automated driving vehicles. The concept is implemented with the proposed timetable as shown in Fig. 5.

The rest of this paper is organized as follows: Section II gives the preliminary and problem formulation. Section III presents our algorithm. Section IV presents experimental results. Finally, we conclude this paper in Section V.

II. PRELIMINARY

A. Background

The first-generation navigation system for vehicles used well-developed routing algorithm to plan a shortest-distance route between two locations on a map without any traffic information. As more and more drivers adopt navigation

Kuen-Wey Lin and Yih-Lang Li are with Institute of Computer Science and Engineering, National Chiao Tung University, Hsin-Chu 300, Taiwan (kuenweylin.cs99g@nctu.edu.tw; ylli@cs.nctu.edu.tw). Masanori Hashimoto is with Department of Information Systems Engineering, Osaka University, Osaka 565-0871, Japan (hasimoto@ist.osaka.ac.jp).

devices to plan their routes, navigation vendors can collect real-time traffic information from the moving speed of navigation devices and further consider newly updated traffic information to find a locally optimal route for currently issued query, where a locally optimal route means the optimality is identified at a certain point in time and might change as time goes forward. To overcome this limitation, navigation vendors start to dynamically offer drivers another better route especially as vehicles slow down or even stop due to serious traffic jams. This dynamic update scheme gives a quick response to traffic jams and try to avoid traffic jams as soon as possible. However the detour to avoid traffic jams is usually suggested as any one road included in the pre-planned route is going to be congested, and at that moment drivers might have passed some redundant paths that lead to congested roads or escape away from congested roads. In other words, if we can evaluate the near-future traffic instead of predicting it, we can detect the traffic jams that are going to happen in the near-future and plan another route from the beginning to reach the destination location earlier than the locally optimal route. Such a routing also helps mitigate the traffic jams. The difference between *evaluation* and *prediction* is detailed as follows: Dynamically re-computing an optimal route for each user offers highly flexible navigation solution and has been adopted by current automatic navigation system such as Google Maps. This kind of navigation system is practical and reasonable due to quick response to real-time traffic information. However, the optimality of that route is identified at a certain time point and might change as time goes forward, i.e., optimality may change as the scheduled route of each driver is not known by navigation system and a road that is scheduled in the optimal route and not congested now might become congested. Regarding a future world where automated driving technologies have become mature and people also get used to taking automated driving vehicles as transportation, the scheduled route of each vehicle is known to navigation system. Thus the integrated navigation system owns a unique advantage of accurate traffic-jam evaluation over traditional navigation system; as a result, the routes of all vehicles can be evaluated by the proposed algorithm, and route distribution and preventing near-future traffic jams can also be realized too.

Let us give an example. Fig. 1 shows a map with eight vertices and ten edges. A vertex represents a junction and an edge represents a road. The number next to an edge is the elapsed time to pass through the edge. Notably, when the number of vehicles increases, the elapsed time in seconds for an edge also increases. Suppose all vehicles go into the map from vertices *A*, *B* and *E* and all towards vertex *H*. A return for the query request of a vehicle from vertex *A* to vertex *H* is a route *A*-*C*-*F*-*H* if no road is congested at the moment this query is issued. If some vehicles enter the map through vertices *B* and *E* earlier than the previous vehicle going into vertex *A*, the locally optimal routes of all vehicles must contain edge *FH*. Since the vehicles from vertices *B* and *E* enter the map earlier, they will enter the edge *FH* earlier than the vehicle from vertex *A*. Therefore edge *FH* becomes congested as the vehicle from vertex *A* approaches junction *F*. Navigation system then suggests another route *F*-*G*-*H* to avoid the traffic jam. It is obvious that the road connecting vertices *F* and *G* is a long way and redundant. We can avoid the redundant paths *AC*, *CF* and *FG* if we can detect a near-future

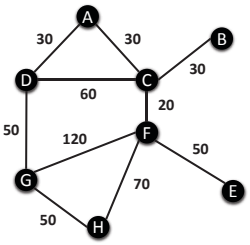

Fig. 1. A map used to illustrate our idea. There are eight junctions and ten roads in the map. The number next to an edge is the elapsed time (without congestion) to pass through the edge.

traffic jam in the beginning by evaluating the near-future traffic and plan another route *A*-*D*-*G*-*H* for the vehicle from vertex *A*. Notably *A*-*D*-*G*-*H* is a longer way than *A*-*C*-*F*-*H* if there is no traffic jam before reaching vertex *H*.

B. Problem Formulation

A simple traffic flow model is constructed to demonstrate the interdependence between the evaluated traffic condition and the choice of route for individual vehicle. The evaluated traffic condition is considered as macro phenomena, and the choice of route for individual vehicle is considered as micro behavior. The details of input of the problem are defined as follows:

- A weighted and directed graph $G=(V, E)$ which consists of a set of vertices $V = \{v_1, v_2, \ldots, v_{|V|}\}$ and a set of edges $E = \{e_1, e_2, \ldots, e_{|E|}\}$. The number of vertices is $|V|$, and a vertex represents a junction or spot; the number of edges is $|E|$, and an edge represents a road. The length of an edge $e \in E$ is given in advance. Each edge e_i has a weight denoted as $w_i \in \{x \mid x \in \mathbf{Q}, x > 0\}$. The weight is calculated by a weight function described as follows:

- A weight function $w(e, d_q^e) \in \{x \mid x \in \mathbf{Q}, x > 0\}$ which is utilized to calculate the cruising time of vehicle q to pass through edge e with traffic condition d_q^e. A traffic condition d_q^e is collected for vehicle q when q enters edge e. In this paper, we collect the number of vehicles on an edge. $w(e, d_q^e)$ is calculated based on Greenshield's V-K relationship [2] as follows:

$$w\left(e, d_q^e\right) = \frac{l_e}{s_e \times \left(1 - \frac{\left(n_e + 1\right) \times \left(l_v + g\right)}{l_e}\right)} \times k, \quad (1)$$

  where s_e is the speed limit of e; n_e is number of vehicles on e; l_v is the length of a vehicle; l_e is the length of an edge; g is the gap between two vehicles; K is a user-defined constant. In this paper, s_e, l_v, and g are given in advance. Note that l_v and g are fixed for each vehicle. K is set to 1.

- A set of queries $U = \{u_1, u_2, \ldots, u_{|U|}\}$. A query is issued by a vehicle, and a vehicle can issue many queries; however, a vehicle has at most one query at the same time. The number of queries is $|U|$. The requirement of each query $u_i \in U$ consists of the following three attributes given by u_i. $V_s^{u_i}$ is the location where u_i is

issued; $v_d^{u_i}$ is the destination of u_i ($v_s^{u_i}$, $v_d^{u_i} \in V$); $T_s^{u_i}$ is the time when u_i is issued. Note that, the general model to allow $v_s^{u_i}$ and $v_d^{u_i}$ at any location on a map can be realized by storing $v_s^{u_i}$ and $v_d^{u_i}$ in a vertex or an edge.

The details of output of the problem are defined as follows:

- A set of schedules $S = (s_1, s_2, \ldots, s_{|U|})$ where s_i denotes a schedule of query $u_i \in U$. Schedule s_i is an ordered list of edges which is the subset of E, and denoted by $(e_1^{u_i}, e_2^{u_i}, \ldots, e_{k_{s_i}}^{u_i})$ where $e_j^{u_i} \in E$, $e_j^{u_i}$ represents an edge traversed by the vehicle issuing query u_i, $e_1^{u_i}$ is the edge connecting $v_s^{u_i}$, and $e_{k_{s_i}}^{u_i}$ is the edge connecting $v_d^{u_i}$. The number of edges of s_i (that is, the number of edges traversed by the vehicle issuing query u_i) is k_{s_i}.

To facilitate the routing for a query u_i, we maintain the following attributes for each vertex $v \in V$:

- $v.\pi$ denotes a pointer to a preceding vertex or NIL.
- $v.\rho$ denotes a pointer to an edge or NIL because in real world two junctions may be connected by more than one road.
- $v.d$ denotes the upper bound on the weight of a routing path with minimum cruising time from $v_s^{u_i}$ (the source of u_i) to v.

For each edge $e \in E$, we maintain a timetable b_e. With a timetable, we can collect the traffic condition of the corresponding edge at a particular time α. As shown in Fig. 2, for the vehicles passing through edge e, we record the time when a vehicle enters e and the time when a vehicle exit e in b_e. Note that, these traffic condition stored in a timetable is calculated and updated by the proposed algorithm. For example, initially, a vehicle q is scheduled to pass through edge e at α_1, and then a timeline for q is added to b_e. However, at α_2 ($\alpha_2 > \alpha_1$) we update the path for q for some reasons such as traffic congestion, and edge e is not on the new path; then, we update b_e to remove the timeline for q.

The cruising time for a schedule s_i is denoted as $CT(s_i)$ and calculated by $\sum_{j=1}^{k_{s_i}} ct\left(e_j^{u_i}\right)$ where $ct\left(e_j^{u_i}\right)$ refers to the cruising time spent by the vehicle which issues query u_i and passes through edge e. $ct\left(e_j^{u_i}\right)$ is calculated by equation (1) and the traffic condition collected from timetable b_e when the vehicle enters edge e. Our target is to reduce the total cruising time of all schedules: $\sum_{i=1}^{|U|} ct(s_i)$.

The traffic route sharing in this paper is based on the assumption that all autonomous vehicles follow their pre-planned routes and no accidental event happens such as traffic collisions and road maintenance.

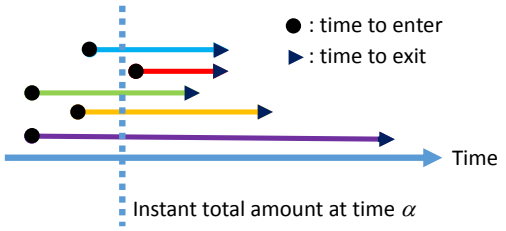

Fig. 2. A timetable consist of a set of timelines. Each timeline is for a vehicle and shows a period when the vehicle is on the road. We can collect the number of vehicles on a road at a particular time α using the corresponding timetable. In the example, the number is four.

III. ALGORITHM

The proposed integrated navigation system maintains the set U using a queue Y. We repeatedly select a query u_i from Y (the one with earliest $T_s^{u_i}$), find a path with minimum cruising time for u_i on the graph $G=(V, E)$ (the weight of each $e \in E$ is the cruising time for u_i to pass e), updates $v_s^{u_i}$ and $T_s^{u_i}$ to direct u_i to the next junction (vertex) along the newly found path, and updates and evaluates new traffic information for $G=(V, E)$ according to the newly found path.

The pseudo code is shown in Fig 3. Line 3 extracts a query u_i from Y. Line 4 initializes the values of d, π, and ρ on $G=(V, E)$. Line 5 calls an enhanced Dijkstra's algorithm [16] to find a path with minimum cruising time and update the values of d, π, and ρ. With these attributes, we can know the roads and junctions included in the newly found route. The enhanced Dijkstra's algorithm utilizes the timetable of each edge to compute the number of vehicles on an edge at a specific time to collect the traffic information. After that, the algorithm calculates the cruising time needed to pass through that edge with the collected traffic information and the weight function $w(e, d_q^e)$. We direct u_i to the junction next to the source and update the corresponding timeline in the timetable of the roads where u_i is planned to pass through, as shown in Line 6. A timeline of a timetable records the time of the vehicle of u_i to enter this road and the time to exit this road. Finally, dynamically re-computing a fastest route for the vehicle of u_i before entering next junction is realized in Lines 7 and 8 by updating u_i's $v_s^{u_i}$ and $T_s^{u_i}$, and then re-inserting it to Y.

Now, an illustrative and detailed map example in Fig. 6 with twenty directed edges and eight vertices is used to demonstrate how our navigation system avoids near-future traffic jams that are evaluated in advance. Since a road in real life is directed, we duplicate each edge in the map with the same length. The black number next to a road is its length between two junctions in meter. We assume the maximum driving speed on each road is 13.8 meters per second (13.8 m/s) - about 50 kilometers per hour (50 km/h). In this case, we assume that there are twenty-one queries. One query from A to H is called Q_{AH}. Ten queries from B to H are named as Q_{BH1} to Q_{BH10}. Another ten queries from E to H are named as Q_{EH1} to Q_{EH10}. Q_{AH} starts at 00:05 (mm:ss) while the other queries start at 00:00. We assume that a traffic congestion occurs on edge e_{FH} when the number of vehicles exceeds 20. We also assume that if there is no traffic congestion on an edge, a vehicle's cruising speed on that edge can be at maximum speed.

Algorithm: The Proposed Algorithm
Input: The map $G=(V, E)$, the weight function $w(e, d_q^e)$, and a set U of queries
Output: A set S of schedules
Begin
1. Initialize queue Y with U;
2. **while** $Y \neq \varnothing$
3. Extract a query u_i with minimum $T_s^{u_i}$ from Y;
4. Initialize map G;
5. Use the enhanced Dijkstra's algorithm to find a path with minimum cruising time for u_i;
6. Evaluated and update the traffic information on G;
7. Assign u_i's $V_s^{u_i}$ to the junction which is next to the original source on the found path;
8. Update $T_s^{u_i}$ of u_i, and then re-insert u_i to Y;

end

Fig. 3. The Proposed near-future traffic information evaluation Algorithm.

First, each query u_i is processed in increasing order of its issued time $T_s^{u_i}$. Because the issued times of Q_{BH} series and Q_{EH} series are the same, they are sorted in increasing order of index number, i.e., from Q_{BH1} to Q_{EH10}. The final element in the queue is Q_{AH} because of its latest issued time. Regarding the feature of near-future traffic jam evaluation, for a road and all vehicles that are planned to pass it, a timetable for this road is maintained to store the times of every vehicle to enter and exit this road. A timeline in a timetable of an edge corresponds to a route containing this edge and represents the time period of a vehicle to pass through this edge.

Three attributes for the first query Q_{BH1} are the source B, the destination H, and departure time at 00:00. Then, we extract Q_{BH1} from Y and initialize G for the enhanced Dijkstra's algorithm. The algorithm identifies the path B-C-F-H for Q_{BH1} for the initial condition without any vehicle on the map G. The path is composed of three edges: e_{BC}, e_{CF}, and e_{FH}. We create or update the timetables for three edges. According to the given parameters, we can know that Q_{BH1} will arrive C at 00:30, arrive F at 00:50, and arrive H at 02:00. The evaluated traffic information is shown in Fig. 4. For query Q_{BH1}, finally, we change its source from B to C, change its issued time to 00:30, and then re-insert it to Y.

Next, queries from Q_{BH2} to Q_{BH10} are sequentially fetched from the queue Y and are executed one by one with the same operations as previous ones followed by updating related timetables on G. After that, the vehicles issuing Q_{BH} series arrive C at 00:30; that is, their new source is C now and new issued time is 00:30. Then, next queries under process are queries from Q_{EH1} to Q_{EH10} because their issued times are all 00:00. Path E-F-H is identified for them and related timetables are also updated. Then the vehicles issuing these queries arrive F at 00:50; that is, their new source is F and new issued time is 00:50.

Finally, Q_{AH} is fetched from the queue due to its issued time is 00:05. At this moment, by evaluating the timetable of e_{FH}, the enhanced Dijkstra's algorithm identifies a near-future traffic jam occurring on edge e_{FH}, caused by more than 20 vehicles on edge e_{FH}, as the vehicle issuing query Q_{AH} arrives

Edge	Timetable
e_{BC}	Q_{BH1}: 00:00->00:30
e_{CF}	Q_{BH1}: 00:30->00:50
e_{FH}	Q_{BH1}: 00:50->02:00

Fig. 4. Timetables for edges e_{BC}, e_{CF}, and e_{FH} after handling query Q_{BH1}.

Edge	Timetable
e_{BC}	Q_{BH1}(Q_{BH2}, Q_{BH3}, ..., Q_{BH10}): 00:00->00:30
e_{CF}	Q_{BH1}(Q_{BH2}, Q_{BH3}, ..., Q_{BH10}): 00:30->00:50
e_{FH}	Q_{BH1}(Q_{BH2}, Q_{BH3}, ..., Q_{BH10}, Q_{EH1}, Q_{EH2}, ..., Q_{EH10}): 00:50->02:00
e_{EF}	Q_{EH1}(Q_{EH2}, Q_{EH3}, ..., Q_{EH10}): 00:00->00:50
e_{AD}	Q_{AH}: 00:05->00:35
e_{DG}	Q_{AH}: 00:35->01:35
e_{GH}	Q_{AH}: 01:25->02:15

Fig. 5. Timetables for the best solution of the twenty-one queries. Note that, we select a longer path A-D-G-H for Q_{AH} to detour the near-future traffic jam at edge e_{FH}. The proposed algorithm can detect the traffic jam on e_{FH} when the vehicles issuing Q_{AH} arrive vertex A, instead of F.

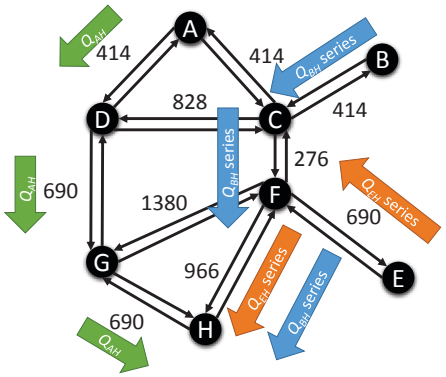

Fig. 6. The final result of our algorithm. The black number next to a road is its length between two junctions in meter. There are eight vertices (junctions), twenty edges (roads), and three series of queries in the example.

F at 00:55 if a shorter path A-C-F-H is selected. Finally another path A-D-G-H with longer distance but less cruising time to detour edge e_{FH} is suggested, as shown in Fig. 5. The final result of our algorithm are shown in Fig. 6. Notably, traditional navigating algorithm without the capability of near-future traffic-jam evaluation cannot know this traffic jam in advance until it arrives F because this traffic jams occur at 00:50. To detour edge e_{FH}, traditional navigation system will suggest a new route F-G-H as the vehicle issuing Q_{AH} arrives F, which is longer and slower than our route A-D-G-H.

IV. Experimental Results

We implemented our algorithm in the C/C++ language on a 3.4-GHz Intel machine with 16-GB memory under the CentOS 7.1.1503 operating system. Three maps for Taipei, Kyoto, and Osaka metropolitan maps are downloaded from OpenStreetMap [13]. The data of Taipei, Kyoto, and Osaka metropolitan area are shown in Table I. To generate a set of

effective queries, we use the data of traffic flow from Department of Transportation, Taipei City Government [14].

Based on the trend of traffic flow during a day, we randomly generate many sets queries with different quantity for comparison. We compared our algorithm with dynamic-update based conventional navigation algorithm ***without*** near-future evaluation capability (DNXNFE).

Table II shows the results of nine sets of queries for the three maps. We list the number of queries, the total reduced cruising time (in hour), and the total reduced mileage (in kilometer) in Table II. We can observe that the reduced time and total reduced mileage can be up to 600 thousands hours (772680 hours versus 166849 hours) and five millions kilometers (12,176,417,246 km versus 7,082,353,617 km), respectively, for the case of Taipei_700k. In this paper, we only show the cases with a lot of queries due to the limited pages. Because the target of our algorithm is to reduce the cruising time, in some cases with few queries, the total mileages of all cars increase. However, for the cases with a lot of queries, the saving of mileages is apparent. Note that, for all cases, the total cruising time is improved. To investigate the effectiveness of the proposed algorithm, we show the comparison between the proposed algorithm and DNXNFE with column charts and line charts, as shown in Fig. 7 and Fig. 8. For cruising time (mileage), we classify all vehicles into two parts depending on whether their cruising time (mileage) is improved or deteriorated. For each part, the average improvement and deterioration rates and total improvement and deterioration numbers are computed accordingly. The following data is average statistics for every hour, and their abbreviations are explained in Table III.

Fig. 7 shows the case of Taipei_700k. We can observe that the numbers of queries with cruising-time and mileage improvement are much larger than those with deterioration in both. The average cruising-time (mileage) improvement rates per hour can be up to about 50% (35%) from 20:00 to 24:00. Avg_Time_Pos_Imp (Avg_Time_Neg_Imp) is the average of all positive (negative) improvement in cruising time at a time period. Thus, the proposed navigation improves the cruising-time in the situation then the number of cars is large and traffic jams easily occurs, as we expected. On the other hand, the worst average deterioration rates are about -10% during some periods of time. The reason why few vehicles have deteriorated cruising time and mileage every hour is that, to escape from congested regions, some vehicles that lead to or pass by congested region are scheduled to detour congested region in advance. The newly scheduled route should be away from congested region and then the traffic along the newly scheduled route increase, causing the vehicles whose shortest routes contain the newly scheduled route to spend more cruising time due to increased traffic.

Fig. 8 shows the difference in speed for the case of Taipei_700k between ours algorithm and DNXNFE. For DNXNFE, at each time period, we calculate the number of vehicles at a speed range. We also calculate the number for our algorithm. Then, we subtract the number of DNXNFE from the one of our algorithm. The difference is shown in *Z*-axis, and the speed range is shown in *X*-axis. The data per hour is shown in *Y*-axis. We can observe that there is a peak at [40-50], implying that most of our queries are with the average speed between 40km/h and 50km/h. However, there is a hollow at (0-10) and [10-20], which means that most of DNXNFE's queries are with the average speed below 20km/h; that is, our algorithm can effectively avoid the traffic jam in these cases.

Table I. The data of the maps of the three cities.

Name	Taipei	Kyoto	Osaka
Num. of roads	212,289	164,349	347,449
Num. of junctions	78,656	58,252	116,986
Population	2,703,829	1,468,879	2,694,731
Area	272km^2	828km^2	223km^2

Table II. Comparison between the proposed algorithm and DNXNFE.

Name	Num. of Query	Total Reduced Cruising Time (Improvement)	Total Reduced Mileage (Improvement)
Taipei_700k	706,984	605,831 (78.41%)	5,094,064 (41.84%)
Taipei_530k	530,240	20,465 (15.01%)	480,493 (8.61%)
Taipei_353k	353,492	6,002 (7.41%)	170,249 (4.81%)
Taipei_295k	294,575	4,139 (6.25%)	117,983 (4.03%)
Kyoto_353k	353,492	1,559 (1.61%)	59,139 (1.37%)
Kyoto_295k	294,575	1,041 (1.3%)	41,712 (1.16%)
Kyoto_177k	176,747	443 (0.93%)	14,087 (0.66%)
Osaka_353k	353,492	2,115 (2.25%)	10,039 (0.15%)
Osaka_295k	294,575	1,502 (1.94%)	147 (0.003%)

Table III. Abbreviations used in Fig. 7.

Abbreviation	Meaning
Avg_Time_Pos_Imp	Average cruising-time improvement rate
Avg_Time_Neg_Imp	Average cruising-time deterioration rate
Avg_Mile_Pos_Imp	Average mileage improvement rate
Avg_Mile_Neg_Imp	Average mileage deterioration rate
Time_Pos_Num	Number of queries with improved cruising time
Time_Neg_Num	Number of queries with deteriorated cruising time
Mile_Pos_Num	Number of queries with improved mileage
Mile_Neg_Num	Number of queries with deteriorated mileage
Total	Total number of queries

V. CONCLUSION

In this paper we proposed an integrated navigation system with near-future traffic jam evaluation to guide a vehicle to detour the roads where traffic jams are going to happen in the near future. We also proposed to realize the integrated navigation system a graph model using the concept of timetable to store the present and near-future traffic conditions. With near-future traffic jam evaluation, experimental results showed that total cruising time and mileage during a period of time for several different traffic conditions can be largely reduced as compared to current navigation systems that dynamically identify a local optimal route based on current traffic conditions for each query. Future works involve allowing more complex accidental events to happen before reaching destination location and designing more practical cruising time computation model considering traffic light systems.

REFERENCES

[1] T. Yamashita, K. Izumi, and K. Kurumatani, "Car Navigation with Route Information Sharing for Improvement of Traffic Efficiency," *IEEE ITSC'04*, pp. 465-470.

[2] T. Yamashita, K. Izumi, K. Kurumatani, and H. Nakashima, "Smooth Traffic Flow with a Cooperative Car Navigation System," *ACM AAMAS'05*, pp. 478-485.

[3] B. S. Kerner, H. Rehborn, M. Aleksic, and A. Haug, "Traffic Prediction Systems in Vehicles," *IEEE ITSC'05*, pp. 251-256.

[4] Y. Wang, M. Papageorgiou, G. Sarros, and W. J. Knibbe, "Real-Time Route Guidance for Large-Scale Express Ring-Roads," *IEEE ITSC'06*, pp. 224-229.

[5] H. Kuriyama,, Y. Murata,, N. Shibata, K. Yasumoto, and M. Ito, "Congestion Alleviation Scheduling Technique for Car Drivers Based on Prediction of Future Congestion on Roads and Spots," *IEEE ITSC'07*, pp.910-915.

[6] R.R. Negenborn, B. De Schutter, and J. Hellendoorn, "Multi-Agent Model Predictive Control for Transportation Networks: Serial versus Parallel Schemes," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 3, pp. 353–366, Apr. 2008.

[7] A. Arsie, K. Savla, and E. Frazzoli, "Efficient Routing Algorithms for Multiple Vehicles with No Explicit Communications," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, pp. 2302-2317, Oct. 2009.

[8] C. Li, S. G. Anavatti, and T. Ray, "Adaptive Route Guidance System with Real-Time Traffic Information," *IEEE ITSC'12*, pp. 367-372.

[9] A. Hrazdira, A. Cela, R. Hamouche, A. Reama, S.I Niculescu, B. Rezende, and Ch. Villedieu, "Optimal Real-Time Navigation System Application to a Hybrid Electrical Vehicle," *IEEE ITSC'12*, pp.409-414.

[10] T. Jurik, A. Cela, R. Hamouche, A. Reama, R. Natowicz, S.I Niculescu, Ch. Villedieu, and D. Pachetau, "Energy Optimal Real-Time Navigation System Application to a Hybrid Electrical Vehicle," *IEEE ITSC'13*, pp. 1947-1952.

[11] S. Wang, S. Djahel, and J. McManis, "A Multi-Agent Based Vehicles Re-routing System for Unexpected Traffic Congestion Avoidance," *IEEE ITSC'14*, pp. 2541-2548.

[12] F. Miao, S. Lin, S. Munir, J. A. Stankovic, H. Huang, D. Zhang, T. He, and G. J. Pappas, "Taxi Dispatch with Real-Time Sensing Data in Metropolitan Areas a Receding Horizon Control Approach," *ACM ICCPS '15*, pp. 100-109.

[13] OpenStreetMap [Online]. Available: https://www.openstreetmap.org/

[14] Department of Transportation, Taipei City Government [Online]. Available: http://english.dot.gov.taipei/

[15] Y.-C. Chiu, J. Bottom, M. Mahut, A. Paz, R. Balakrishna, T. Waller, J. Hicks, "Dynamic Traffic Assignment: A Primer," *Transportation Research Circular E-C153*, June 2011.

[16] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. 2009. *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.

[17] Google Maps [Online]. Available: https://www.google.com/maps

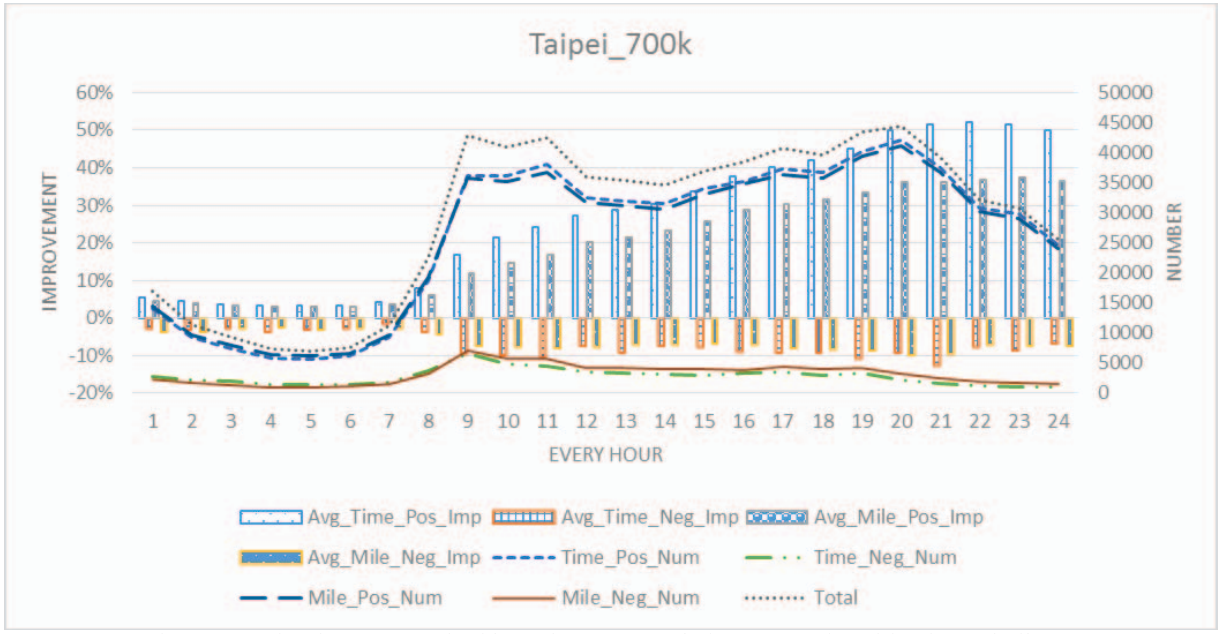

Fig. 7. Comparison between ours algorithm and DNXNFE – The improvement in cruising time and mileage.

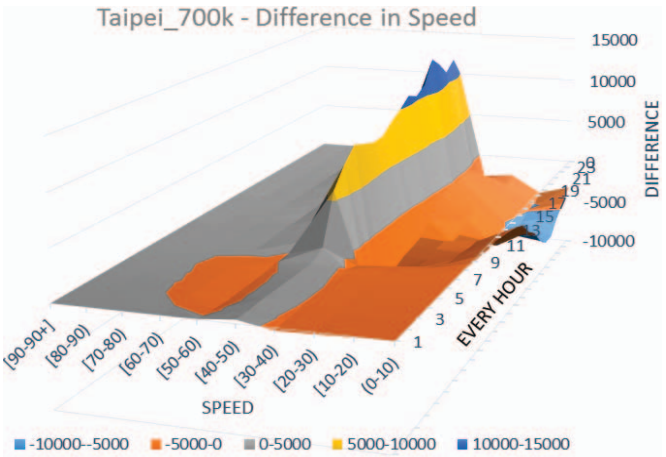

Fig. 8. Comparison between ours algorithm and DNXNFE - The difference in speed.