2013

Distributed optimisation for traffic management

Tran Viet Nhan Nghi
University of Wollongong

Recommended Citation

Nghi, Tran Viet Nhan, Distributed optimisation for traffic management, Master of Computer Science - Research thesis, School of Computer Science and Software Engineering, University of Wollongong, 2013. https://ro.uow.edu.au/theses/4827

University of Wollongong

DISTRIBUTED OPTIMISATION FOR TRAFFIC MANAGEMENT

A Thesis Submitted in Partial Fulfilment of
the Requirements for the Award of the Degree of

Master of Computer Science - Research

from

UNIVERSITY OF WOLLONGONG

by

Tran Viet Nhan Nghi

School of Computer Science and Software Engineering
Faculty of Informatics

2013

CERTIFICATION

I, Tran Viet Nhan Nghi, declare that this thesis, submitted in partial fulfilment of the requirements for the award of Master of Computer Science - Research, in the School of Computer Science and Software Engineering, Faculty of Informatics, University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications at any other academic institution.

(Signature Required)

Tran Viet Nhan Nghi
2 April 2013

Dedicated to

My parents, Dien & Hiep
and
My brother, Hao

Table of Contents

List of Tables

List of Figures

Distributed Optimisation for Traffic Management

Tran Viet Nhan Nghi

A Thesis for Master of Computer Science - Research

School of Computer Science and Software Engineering
University of Wollongong

ABSTRACT

This thesis reports on the development of a multi-agent approach to distributed traffic optimisation. In particular, I propose a solution to the dynamic traffic assignment problem in a decentralised manner and then I introduce the new infrastructurelessly decentralised traffic information system. By using this system, each vehicle agent is able to update the current traffic condition through vehicle-to-vehicle communication. For solving dynamic traffic assignment problem, I propose a novel completely decentralised multi-agent coordination algorithm, which is a synergy between dynamic distributed constraint optimisation problem (DynDCOP) algorithm and auction. Using this algorithm, vehicle agent is able to reduce its individual travel time as well as total travel time of overall system. The simulation is carried out in order to evaluate different traffic planning algorithms that include decentralised uncoordination, centralised coordination and decentralised coordination algorithms. Finally, the experimental results show that the performance of proposed decentralised coordination algorithm is high in comparison to centralised coordination algorithm.

KEYWORDS: multi-agent system, dynamic traffic assignment, dynamic distributed constraint optimisation problem, distributed traffic management

Acknowledgements

I would like to thank my main supervisor, Prof. Aditya K. Ghose, for his careful guidance through valuable innovation ideas, suggestions and discussions.
I always gratefully acknowledge the financial support through the University Postgraduate Award provided by SMART Infrastructure Facility and University of Wollongong. Without this support this work would never have been possible.
Thanks are also due to Graham Billiau, Terrence Lee who gave me much valuable advice on SBDO algorithm and traffic simulation in the early stages of this work.

Chapter 1

Introduction

Firstly, this chapter provides my motivation in conducting research on ***real-time traffic management****. Secondly, it gives a brief overview of the research work related to my topic. Finally, it provides an outline of my thesis.*

1.1 Motivation

The last two decades have witnessed a huge growth in global urban population. According to the urbanisation study of World Health Organization (WHO) [30], the global population lived in an urban area increased from less than 40% in 1990 to more than 50% in 2010. Moreover, by middle of the 21st century, it is predicted that the urban population in 2050 will be 5.2 billion increasing by more then twice in 2009 (2.5 billion).

Rapid growth of urban population is the major cause for the dramatic increase in traffic volume on road segments. Furthermore, the *traffic demand* generated by commuters for everyday life activities typically greater than the available road capacity (*supply*). Thus, it results in *traffic congestion* [2] that most likely occurs in major cities with large population. According to the Asian Development Bank, the cost of traffic congestion goes up to 2-5% of gross domestic product (GDP) of countries every year due to lost time and higher transport costs. Hence, the traffic congestion problem has received much attention of different communities and organisations ranging from academic researchers to industry practitioners and government authorities.

Traffic congestion can be reduced by either increasing supply, or by improving traffic management. With first approach, creating new routes or adding more capacity to existing road segments are feasible in practice. However, according to Braess's paradox [9], adding more roads to an existing transportation network might, in turn, lead to longer travel times of individual travellers. Besides first approach, the second approach can be realised using the current high technology developments for traffic management.

For the past five years, there has been a rapid rise in the use of *Intelligent Transportation System* (ITS) [3, 1, 17]. ITS is a general term for integrated applications of communication and information technologies for alleviating traffic congestion. By using a wide variety of *traffic management strategies*, ITS assists individual travellers in making better, more coordinated and intelligent decisions.

Since 1970s, *Dynamic Traffic Assignment* (DTA) [32, 22, 12] has been used intensively by transportation research community for studying the dynamic of transportation system for *transportation planning*. The goal of DTA is assigning routes to individual travellers at different time points of simulation in order to transform traffic system state to approximate *dynamic user equilibrium* (DUE) state. At DUE state, no individual travellers have any incentive to change their current routes and the traffic system achieves *social optimum* (SO). SO means that the total travel time of all individual travellers is minimised and their current route choices are optimal.

Simulation-based DTA model [10, 18] has become an efficient approach for solving DTA problem by combining simulation and iteration algorithms for finding the optimal routes converging approximately traffic system state to DUE state. In simulation-based DTA model, the process of computing the optimal routes for all individual travellers takes place in a centralised manner. Because of centralisation characteristic and the lack of optimisation techniques, simulation-based DTA approach is inappropriate for real-time application and large-scale transportation network. Obviously, the *equilibrium-searching algorithm* may iterate indefinitely for finding the optimal routes because of the lack of exploiting any optimisation technology. Moreover, when using centralised processing system, the algorithm's speed of converging to DUE state is relatively slow especially for the transportation network with a extremely large number of vehicles.

Inspired by the important applications of DTA and the aforementioned disadvantages of simulation-based DTA models, I propose the solution to DTA problem in a completely decentralised manner. The traffic system described in solution to DTA problem is a *multi-agent system*, where vehicles are modelled as *autonomous vehicle agents*. These agents are capable of making their own decisions on route selection in order to cooperatively reduce total travel time by *vehicle-to-vehicle* (V2V) communication in a completely decentralised manner.

The research topic of this thesis is closely related to the work described in honours thesis of Lee [25]. However, the work of this thesis significantly extends Lee's work on *peer to peer coordinated traffic planning* by making the followings contributions:

- Build the model of DTA problem as DynDCOP model,
- Designing the infrastructurelessly decentralised traffic information system,
- Proposing the completely decentralised multi-agent coordination algorithm for solving DTA problem using SBDO algorithm and auction,
- Conducting experiments for evaluating proposed coordination algorithm with different planning algorithms.

1.2 Related Work

In [39], Yang and Recker modelled complete distributed traffic information system that operates without any centralised control and allows dynamic vehicle online routing. Vehicles in this system contribute to produce real-time traffic information by generating and exchanging local traffic information sensed by themselves through vehicle-to-vehicle communication. Based on this real-time traffic information, vehicles make their own in-trip rerouting decisions to alternative routes on the basis of rational-boundary and binary-logit models. The result of simulation shows that vehicles with the dynamic rerouting capability are able to reduce not only their own individual travel time, but also total travel time of all vehicles within the system. However, the limitation of this model is that the proposed in-trip rerouting strategy might cause the traffic jams switch from one road to another. Every vehicle, which is in congested situation, will behave in the same way based on its local view of overall system and therefore the total travel time might not be improved.

Bazzan et al. [4] proposed centralised and decentralised approaches for computing routes for vehicles. The decentralised approach allows vehicles to reroute when they perceive that actual travel time is greater then expected time. Based on its own traffic information, vehicle calculates new route and communicate it to another vehicles who are on the links of new route in order to receive the cost for travelling these links. If the cost of new route is appropriate, vehicle will change their current route to new route, otherwise it will replan again and repeat this process. It is obvious from this approach that the costs, which are requested by vehicles from another ones for evaluating their new routes, become obsolete and inaccurate. Moreover, this approach has the same above-mentioned limitation in [39] as the traffic jams will occur in another road that many vehicles travel through after performing re-routing process.

For managing traffic in decentralised manner, DCOP techniques have been applied extensively in [31] [34] [14] [23]. Ottens and Faltings [31] have used Asynchronous Open DPOP, a complete asynchronous DCOP algorithm developed on the basis of DPOP [33], for solving truck task coordination problem (TTC). TTC is the multi-agent planning problem that consists of set of trucks and a set of packages that need to be picked up and delivered to customers. Truck agents need to coordinate their plan in order to decide which truck agent will be responsible for packages that locate in the overlapping areas between two or more truck agents.

In [34], hybrid method of coalition formation and DCOP algorithm OptAPO [27] has been presented for resolving conflict between convoys travelling road network with limited resources such as road capacity. The solution of convoy movement problem is the set of routes that must be satisfied the condition that the number of convoys on a link does not exceed its capacity. OptAPO algorithm also has been used in Bazzan's work for coordinating traffic lights in [14] and different DCOP algorithms [28] [27] [33] have been evaluated in order to measure their performances for solving traffic light coordination in [23].

Despite of the fact that there have been increasing concerns about the developments of decentralised traffic management systems associated with technologies from control engineering and computer science, all the existing approaches face the requirements for efficiency, scalability (large-scale network of agents) and adaptivity to dynamically changing environment.

1.3 Thesis Structure

This thesis is organised as follows:

- Chapter 2 provides the background of research topic,
- Chapter 3 describes the distributed traffic management problem and the decen-

tralised multi-agent coordination algorithm for solving it,

- Chapter 4 presents the experimental results of comparing different planners using traffic simulation,
- Chapter 5 summarises the work of this thesis and discusses about future work.

Chapter 2

Background

This chapter provides a background on applications of agent technology in traffic management, dynamic traffic assignment and dynamic distributed constraint optimisation problem.

2.1 Applications of Agent Technology in Traffic Management

Traffic congestion is not trivial problem for solving in modern society because of the dynamics and uncertainty to predict in order to alleviate it. For reducing the traffic congestion, road authority could increase the capacity of existing transportation infrastructure by adding more roads, lanes. Thus, this requires a lot of money, time for designing and evaluating the efficiency of the new designed transportation infrastructure. However, another potential method to avoid traffic congestion is increasing the efficiency of existing transportation infrastructure by applying techniques from computer science field to traffic management.

For the past five years there has been a rapid rise in the use of agent-based technology in traffic management [5] [11] [15]. Autonomic, collaborative, mobile and reactive features make intelligent agents prominent from the point of view of traffic and transportation. The automated traffic control and management system can be implemented because the autonomy of intelligent agents in operating without the direct involvement of humans.

Recently, the vehicular ad-hoc network (VANET) has been developed and standardised in order to support the vehicle-to-vehicle and vehicle-to-infrastructure communication. Therefore, intelligent agents in transportation have the ability to collaborate and coordinate in order to optimise global utility, e.g. total travel time.

Moreover, intelligent agents are capable of adapting to dynamically changing environment by responding to these changes in a timely fashion. Therefore, intelligent agents can be used in developing an agent-based transportation system based on real-time traffic conditions. Multi-agent system provides techniques and methods that have been utilised in many sides of traffic and transportation including the followings:

- Modeling and simulation,
- Intelligent traffic control and management,
- Dynamic routing and congestion management,
- Driver-infrastructure collaboration,
- Decision support.

Real-time traffic services including real-time traffic information and dynamic route guidance have been used widely and become a fast-growing business in the last few years. According to iSuppli Corp [26], the overall profit produced by real-time traffic services will increase rapidly from $268 million in 2008 to $4.7 billion in 2014. Moreover, the number of worldwide customers using these services will rise to 184.9 million in 2014 from 18.5 million in 2008. Companies providing such services include TomTom with commercial TomTom HD Traffic service [35] and free Google Maps [19]. With the support of real-time traffic information, users utilising these services are advised on selecting the best route through traffic jams among the set of alternative optimal routes returned by central server on users's queries.

For the accuracy of traffic information supplied to the users, probe data collected from cell phones and navigation devices is used to calculate traffic density and to predict traffic jams. According to TomTom HD Traffic's description, probe data is accumulated from 80 million anonymous travelling mobile phone users and 1 million users of TomTom services and consequently more than 1 billion probe data is collected every day. The more customers employ TomTom HD Traffic service, the higher the quality of services they get. In other words, commuters on the road could be benefit of reduction of 15% of total travel time by taking advantage of sending their routes to a central server and receiving optimal route from it.

Despite the fact that current real-time traffic services provide realistic support for drivers to make route decisions based on their local view of the overall system, the efficiency and performance of route guidance systems that use these services have to be thoroughly analysed and evaluated. The result of these analysis and evaluation might be a valuable source of inspiration for us to propose a novel way of information sharing, traffic congestion alleviating and travel time reduction.

First, the limitations and issues of current real-time traffic services using by real-time traffic guidance systems are followings:

- The real-time traffic information received by worldwide users come from *heterogeneous* information sources that produced by variety of methods for collecting probe data and calculating travel time. These sources with diverse qualities could not ensure the standardised level of accuracy of real-time traffic information. Moreover, it's nearly impossible for real-time traffic services providers to collaborate in order to improve the quality of traffic information and then services for their customers.

- By making their own decisions based on the number of alternative routes received from the central server of provider, travellers actually cause the traffic jams switching from a set of roads to another. Because having the local views of the overall system, self-interested commuters usually chooses the quickest routes instead of collaborating their choices in order to avoid traffic congestion and reduce the total travel time of all individuals.

- With likely millions of queries on optimal route plans from travellers, central server might pay an expensive cost of processing these queries in right time for travellers. Moreover, if some unexpected events happen on the roads e.g incidents, road works, etc, the the set of alternative routes for each traveller must be calculated from scratch and the time of completing this task might delay the result that must be sent to travellers straight away. Therefore we need a kind of proactive system that can handle every change in the traffic network for providing the high quality solution to customer in permitted restrict amount of time.

- Traffic network is essentially a geographically distributed multi-agent system. In fact, the two-way communication between travellers and central server is not always effective. Because of the low bandwidth of telecommunication network, either the queries of travellers on optimal routes or data that is sent from central server to traveller could be delayed. Additionally, the failure of central processing system causes all subscribers to real-time traffic services their losses of navigating through the road network. For that reason, we must design an effective communication mechanism between travellers in a decentralised manner.

- Privacy issue has been increasingly become an important aspect for evaluating the security of a system. Users of TomTom HD Traffic, Google Maps are advised to share their start, destination locations and maybe their route plans with service providers. This data collected from these users could be analysed by the same providers or sold to another companies for the purpose of doing research on advertising strategy, recommendation system, etc. Therefore, real-time traffic services provider is not be able to guarantee the personal identity of customers.

Yamashita and Kurumatani [38] have proposed the centralised approach using route information sharing between drivers in order to avoid the traffic congestion. Each driver searches the route with minimum travel time and broadcasts route information to the route information server. The route information server then uses driver's route information to predict the possible traffic congestion and sends it back to driver. Driver uses traffic congestion information to revise its route plan in order to find the best one.

Gratie and Florea [20] addressed the benefit of possible alternative routes when the traffic became congested. Actually,in their approach , multi-agent system has been used in centralised way by considering agents as driver, intersection and city. Routing algorithm uses probability formula for selecting the alternative route, but this algorithm can not work with the real-world traffic.

Moreover, challenging issue has been marked with their approach is that the routing algorithm needs to be changed in order to provide the best alternative routes in a decentralised manner. However, the experimental results show that centralised intelligent routing has proved itself to be an effective approach to avoid the traffic congestion.

In [39], Yang and Recker modelled complete distributed traffic information system that operates without any centralised control and allows dynamic vehicle online routing. Vehicles in this system contribute to produce real-time traffic information by generating and exchanging local traffic information sensed by themselves through vehicle-to-vehicle communication. Based on this real-time traffic information, vehicles make their own in-trip rerouting decisions to alternative routes on the basis of rational-boundary and binary-logit models.

The result of simulation shows that vehicles with the dynamic rerouting capability are able to reduce not only their own individual travel time, but also total travel time of all vehicles within the system. However, the limitation of this model is that the proposed in-trip rerouting strategy might cause the traffic jams switch from one road to another. Every vehicle, which is in congested situation, will behave in the same way based on its local view of overall system and therefore the total travel time might not be improved.

2.2 Traffic Assignment

2.2.1 Overview of Traffic Assignment

The aim of traffic assignment is trying to establish the network traffic flow and condition as the result of commuters's travelling. Based on the interaction between commuters, traffic assignment algorithms calculate route and link capacities and travel times at equilibrium condition. At equilibrium state, no driver has any incentive to change his current route.

Figure 2.1 illustrates the static traffic assignment in a one-shot simulation. In static traffic assignment, route set and flows are pre-planed and remain indifferent during simulation.

A more advanced approach has shortest routes frequently updated based on predominant traffic conditions and has these routes assigned to recently generated vehicles at the start of the trip. This is referred to as dynamic traffic assignment as shown in Figure 2.2.

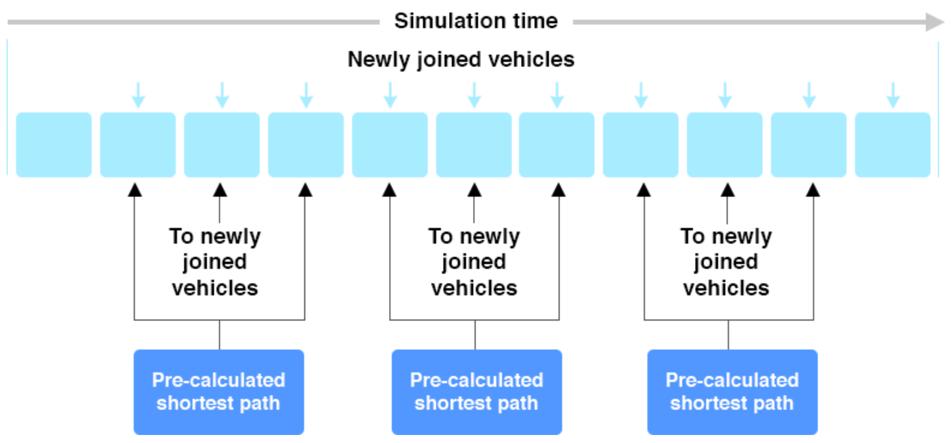

Figure 2.1: Static assignment in a one-shot simulation, Chiu et al. [12]

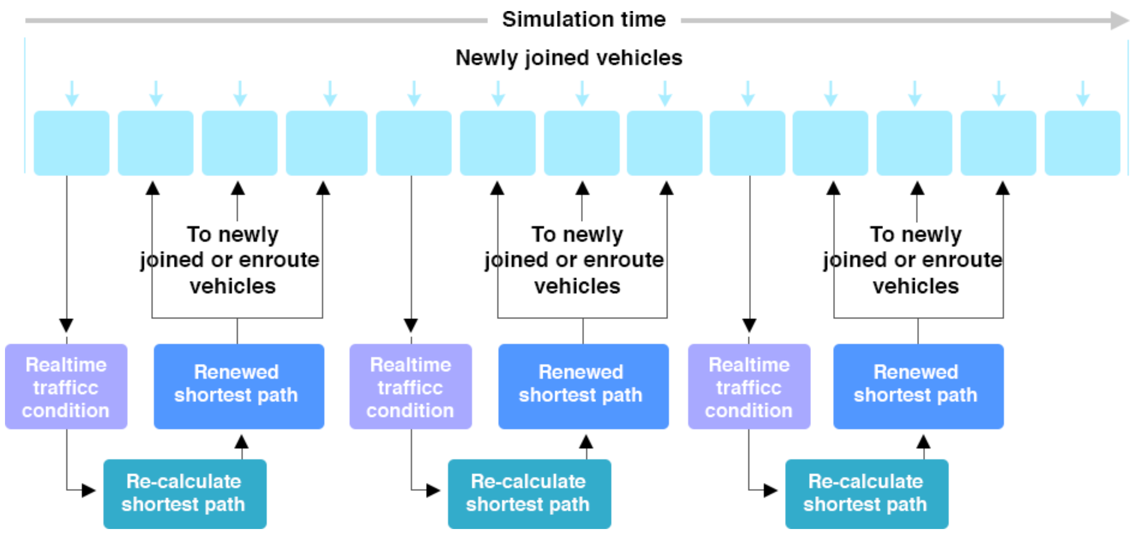
Simulation time
Newly joined vehicles
To newly joined or enroute vehicles
To newly joined or enroute vehicles
To newly joined or enroute vehicles
Realtime trafficc condition
Renewed shortest path
Realtime trafficc condition
Renewed shortest path
Realtime trafficc condition
Renewed shortest path
Re-calculate shortest path
Re-calculate shortest path
Re-calculate shortest path

Figure 2.2: Dynamic assignment in a one-shot simulation, Chiu et al. [12]

2.2.2 Static Traffic Assignment

The static traffic assignment problem was addressed by Beckmann [6], Nesterov, de Palma [29] and recently Chudak [13]. In [13], the static traffic assignment problem is defined formally as:

- A traffic newtwork $G = (N, A)$, where N is the set of nodes (intersections), an A is the set of arcs (roads).
- Each arc $a \in A$ has a capacity, c_a, which is the maximal number of cars that can go through the road a during a given period of time. An arc a also has a free

travel time $\bar{t}_a$, which is the minimum travel time needed to go through road a at maximal allowed speed.

The goal of the static traffic assignment problem is to assign routes to drivers in order to attain a Social Optimum (SO) state or an User Equilibrium (UE) state.

Definition 2.2.1 (Wardrop's First Principle [36])
User equilibrium (UE) is the state, at which no driver has any incentive to change his current route.

Definition 2.2.2 (Social Optimum)
Social Optimum (SO) is the state, at which the utilization of the transportation network is maximum (e.g. minimum total travel time).

The current *traffic pattern* of a traffic network is specified by a flow, f (the places of drivers in the network) and travel time t (total travel time of all drivers if they use the assigned routes).

2.2.2.1 Nestrov and de Palma Model

In Nesterov and de Palma model [13], [29], the capacity c_a of the road a in traffic network can not be exceeded, i.e., the drivers are able to travel with free-flow speed.

Let (f, t) be a traffic assignment, then (f, t) satisfy the following conditions:

- The number of vehicles on arc a (f_a) never exceeds the capacity of arc a, $f_a \leq c_a$.
- Below capacity c_a the travel time t_a on arc a is equal to its free travel time $\bar{t}_a$. At capacity limit, it can take any value larger or equal to the free travel time:

$$\text{if } f_a < c_a \Rightarrow t_a = \bar{t}_a$$
$$\text{if } f_a = c_a \Rightarrow t_a \geq \bar{t}_a$$

In Netsterov and de Palma model, calculating a traffic assignment at SO is equivalent to solving the *minimum linear cost multi-commodity problem*, i.e., minimise the total travel time: $\sum_{a \in A} f_a t_a$.

2.2.2.2 Price of Anarchy

The price of anarchy was first introduced by Koutsoupias and Papadimitriou [24] and it is the ratio between the total utility at UE and at SO. The total utility is the total travel time of a traffic pattern (f, t) and is denoted by $U(f, t)$. Then $U(f, t)$ is calculated as:

$$U(f,t) = \sum_{a \in A} f_a t_a$$

and the price of anarchy Pr is then formulated as follows:

$$Pr = \frac{U(f^{UE}, t^{UE})}{U(f^{SO}, t^{SO})} \tag{2.1}$$

where (f^{UE}, t^{UE}) corresponds to a traffic assignment at UE and (f^{SO}, t^{SO}) corresponds to a traffic assignment at SO.

2.2.2.3 Braess Paradox

The Braess paradox [9] occurs when adding more resources to a transportation network as more resources create worse delays for the drivers. In [9], the Braess paradox is stated as follows: "If every driver takes the path that looks most favourable to him, the resultant running times need not be minimal."

Let me consider an example of Braess paradox from [16]. Fig. 2.3 illustrates a road network, on which 4000 drivers desire to travel from point **START** to **END**. The travel time (in minutes) on links **START-A, B-END** is the number of travelers (T) divided by 100, and on links **START-B, A-END** is a constant 45 minutes.

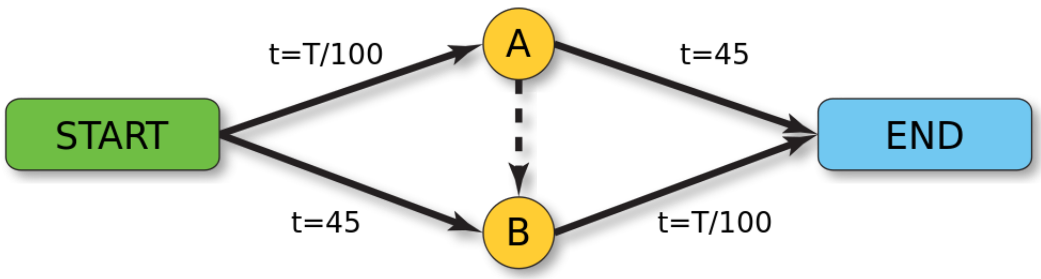

Figure 2.3: Example of Braess paradox

If there is not dashed road, the time needed to drive **START-A-END** route with A (a number) drivers would be $\frac{A}{100} + 45$. And the time needed to drive **START-B-END** route with B (a number) drivers would be $\frac{B}{100} + 45$.

If either route were shorter in terms of travel time, it would not be a Nash equilibrium: a rational driver would switch its route from the longer to shorter route. As there are 4000 drivers, the system is at user equilibrium state if $A = B = 2000$. Consequently, the travel time for each route is $\frac{2000}{100} + 45 = 65$ minutes.

Suppose the dashed line is a road with travel time of approximately 0 minutes. Therefore, the shortest route now is **START-A-B-END** because the link **START-A** will take at most 40 minutes to drive in comparison to constant 44 minutes for link **START-B**. Consequently, 4000 drivers switch their routes to **START-A-B-END** route and their travel time for arriving to destination location is $\frac{4000}{100} + \frac{4000}{100} = 80$ minutes, an increase from 65 minutes when the A-B road does not exist.

Finally, no drive has an incentive to switch because two original routes **START-A-END** and **START-B-END** now require 85 minutes to drive. The traffic system now is at user equilibrium but is far from system optimum. Moreover, when adding the dashed link **AB**, the performance of overall system, which is the total travel time of all drivers, decreased according to Braess paradox.

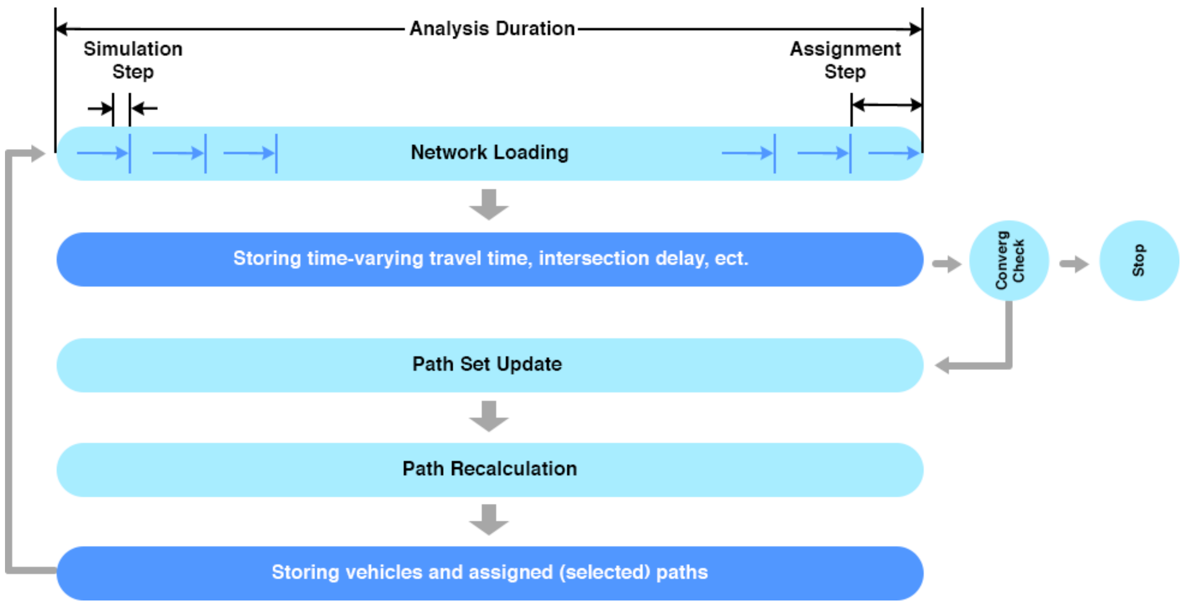

Figure 2.4: General DTA algorithmic procedure, Chiu et al. [12]

2.2.3 Dynamic Traffic Assignment

2.2.3.1 Overview of Dynamic Traffic Assignment

As shown in 2.4, the general method of finding an dynamic user equilibrium in DTA is apply three algorithmic steps in sequence iteratively, until traffic system state converged to DUE state:

- **Network Loading:** What are the resulting route travel times given a set of route choices?
- **Path Set Update:** What are the new shortest routes given the current route travel times?
- **Path Assignment Adjustment:** , how to assign routes to vehicles to better approximate a dynamic user equilibrium given the updated route sets?

2.2.3.2 Instantaneous and Experienced Travel-Times

Figure 2.5 and figure 2.6 illustrate an example that demonstrates the difference between instantaneous and experienced travel-times.

2.3 Distributed Constraint Programming

2.3.1 Definitions

Definition 2.3.1 (Constraint Optimisation Problem)
A Constraint Optimisation Problem (COP) is a tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{F} \rangle$ where:

- *$\mathcal{X}$ is a set $\{x_1, x_2, \ldots, x_n\}$ of variables,*
- *$\mathcal{D}$ is a set $\{d_1, d_2, \ldots, d_n\}$ of domains,*
- *$\mathcal{C}$ is a set $\{c_1, c_2, \ldots, c_m\}$ of constraints defined over a set $\mathcal{R}$ of relations $\{r_1, r_2, \ldots, r_m\}$ where r_i is the relation between $\{x_{1i}, x_{2i}, \ldots, x_{ni}\}$,*
- *$\mathcal{F}$ is a set $\{f'_1, f'_2, \ldots, f'_q\}$ of cost functions defined over $\mathcal{R}$.*

A constraint c_i is a pair $\langle t_i, r_i \rangle$, where $t_i \subset \mathcal{X}$ is a subset of k variables and r_i is an k-ary relation on the corresponding subset of domains d_i.

A cost function is a function $f'_i(r_i) \rightarrow \Re$. A value u_i returns by a cost function f'_i is called an utility and an objective function is defined as:

$$\mathcal{O}(\mathcal{X}) = \sum_{i=1}^{q} u_i$$

A solution to COP is the set of all assignments to $x_i \in \mathcal{X}$ that satisfies $\forall c_i \in \mathcal{C}$ and minimise objective function $\mathcal{O}$ as:

$$\arg\min_{\mathcal{X}} \mathcal{O}(\mathcal{X})$$

Definition 2.3.2 (Distributed Constraint Optimisation Problem)
A Distributed Constraint Optimisation Problem (DCOP) is a tuple $\langle \mathcal{A}, \mathcal{COP}, \mathcal{C}', \mathcal{F}'' \rangle$ *where:*

- $\mathcal{A}$ *is a set* $\{a_1, a_2, \ldots, a_k\}$ *of agents,*
- $\mathcal{COP}$ *is a set* $\{\mathcal{COP}_1, \mathcal{COP}_2, \ldots, \mathcal{COP}_l\}$ *of* $\mathcal{COP}$*s such that* $\mathcal{X}_i^{\mathcal{COP}_i} \cap \mathcal{X}_j^{\mathcal{COP}_j} = \emptyset$ *and each agent* a_i *controls exactly one* $\mathcal{COP}_i$*.*
- $\mathcal{C}'$ *is a set* $\{c'_1, c'_2, \ldots, c'_g\}$ *of shared constraints. Each shared constraint* c'_g *defines over a subset* $\{\mathcal{COP}_1, \mathcal{COP}_2, \ldots, \mathcal{COP}_l\}$ *of* l $\mathcal{COP}$*s, where* $l \geq 2$*,*
- $\mathcal{F}''$ *is a set* $\{f''_1, f''_2, \ldots, f''_h\}$ *of cost functions defined over subsets of* $\mathcal{COP}$ *that shares constraints between them.*

The objective function for DCOP is defined as:

$$\mathcal{O}'(\mathcal{X}') = \sum_{i=1}^{h} u'_i,$$

where $\mathcal{X}' = \{\mathcal{X}_1^{\mathcal{COP}_1}, \mathcal{X}_2^{\mathcal{COP}_2}, \ldots, \mathcal{X}_l^{\mathcal{COP}_l}\}$ *and shared utility* u'_i *is a cost returned from* f''_i

A solution to DCOP is the set of assignments to all variables of $\mathcal{X}_i^{\mathcal{COP}_i}$*,* $i \in \{1, 2, \ldots, l\}$ *that satisfy* $\forall c'_i \in \mathcal{C}'$ *and minimise the objective function* $\mathcal{O}'$ *as:*

$$\arg\min_{\mathcal{X}'} \mathcal{O}'(\mathcal{X}')$$

Definition 2.3.3 (Dynamic Distributed Constraint Optimisation Problem)
A Dynamic Distributed Constraint Optimisation Problems (DynDCOP) is a sequence that consists of DCOPs as:

$$\langle DCOP_1, DCOP_2, \ldots, DCOP_n \rangle$$

, where $X'_{DCOP_i} \triangle X'_{DCOP_j} \neq \emptyset$*,* $C'_{DCOP_i} \triangle C'_{DCOP_j} \neq \emptyset$ *and* $F''_{DCOP_i} \triangle F''_{DCOP_j} \neq \emptyset$*. Note that given two sets* A, B *then* $A \triangle B = (A \cup B) \setminus (A \cap B)$*.*

Solving DynDCOP is maintaining solutions for all DCOPs that all constraints C'_i *must be satisfied and the objective function* $\mathcal{O}'_i$ *is minimised for every* $DCOP_i$*.*

2.3.2 Support-Based Distributed Optimisation Algorithm

SBDO algorithm [8] is designed for solving Dynamic Distributed Optimisation Problems based on complete asynchronous Support-Based Distributed Search [21]. SBDO employs argumentation as its mechanism. Agent sends a proposal to neighbour agents in order to influence these agents to accept it. Proposal is composed of assignments to variables controlled by itself and neighbour agents that satisfy local and shared constraints. This proposal is also associated with the total utility which is the sum of local and shared utilities. After receiving proposal from sending agent, neighbour agents check the consistency of assignments to variables in received proposal with assignments to their current variables. If consistent, then neighbour agent put the received proposal to the list of all received proposals associated with sending agents for considering who will be its supporter. Neighbour agent then choose an agent with maximum total utility as its supporter and compute the local solution based on supporter's proposal as its local view to global system. Therefore, neighbour agent sends proposal expressed its local view to all neighbour agents. This process will make the dynamic variable ordering of all agents.

In SBDO algorithm, each agent greedily selects what agent to be as its support and the values to assign to its own variables. Because an agent may have many variables, this agent requires its own centralised Dynamic COP solver. Agent that has chosen sub-optimal assignments may changes its assignment because of collection of agents when support is selected.

Each agent takes simple basic steps as followings. First in agent's message queue it processes all the messages. Then it chooses what values to assign to its own variables. Last it broadcasts all of its neighbours a message to tell them what values it has chosen for its variables.

All of the nogoods received should be taken when starting to work with processing messages. At the beginning, nogoods are processed if they are later become obsolete by a message from the environment and because one of them might invalidate one of the isgoods in the message queue. When receiving a nogood, it is added to the set of all known nogoods. When all nogoods are processed the received isgoods must be rechecked to detect if they are inconsistent with this agents assignment. If so, the isgoods sender must be informed by sending a nogood. This will compel the sender into changing their value in the next iteration. Next all environment messages are processed. The order within this group is not important, but they may affect how the isgoods are processed. Finally, the received isgoods are processed. First, recv(A) is updated with this most recent isgood, then it checks if there is a valid assignment to its own variable. If there isnt, a nogood is created and sent back to the agent that sent the isgood. This will force the sender to change their value in the next iteration.

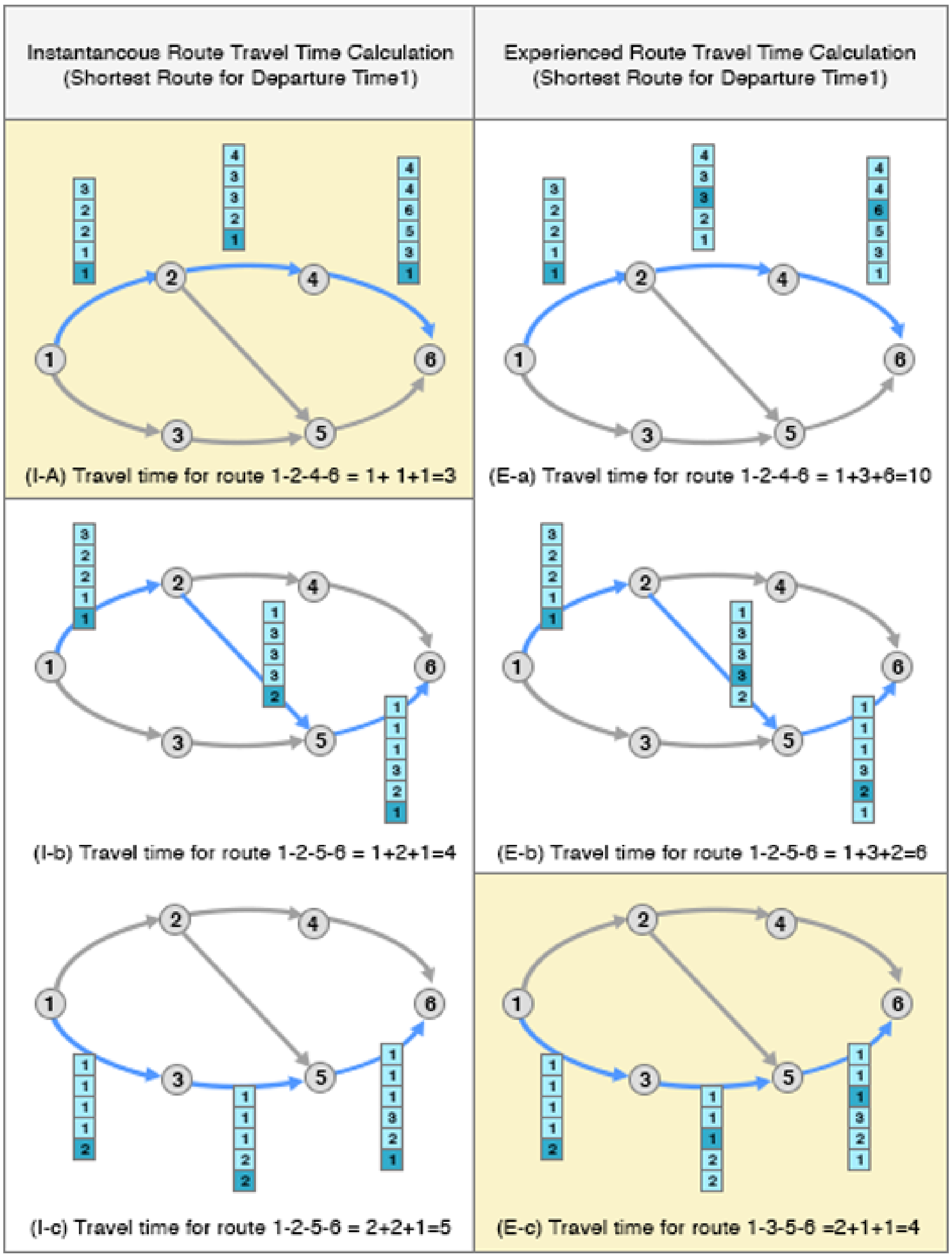
Instantaneous Route Travel Time Calculation (Shortest Route for Departure Time1)
Experienced Route Travel Time Calculation (Shortest Route for Departure Time1)
(I-A) Travel time for route 1-2-4-6 = 1+ 1+1=3
(E-a) Travel time for route 1-2-4-6 = 1+3+6=10
(I-b) Travel time for route 1-2-5-6 = 1+2+1=4
(E-b) Travel time for route 1-2-5-6 = 1+3+2=6
(I-c) Travel time for route 1-2-5-6 = 2+2+1=5
(E-c) Travel time for route 1-3-5-6 =2+1+1=4

Figure 2.5: Different shortest routes obtained by instantaneous travel-time and experienced travel-time approaches with departure time 1, Chiu et al. [12]

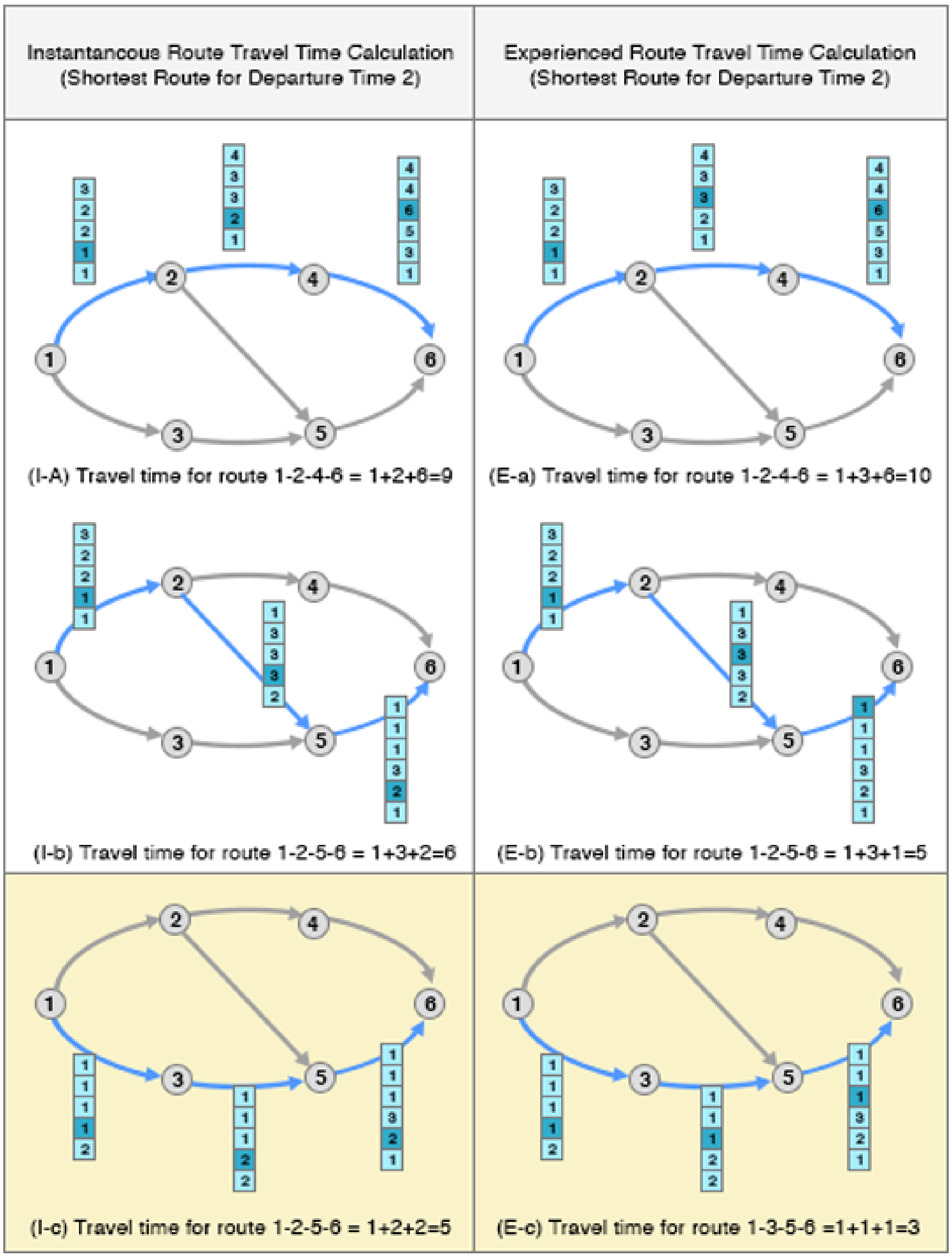
Instantaneous Route Travel Time Calculation (Shortest Route for Departure Time 2)
Experienced Route Travel Time Calculation (Shortest Route for Departure Time 2)
(I-A) Travel time for route 1-2-4-6 = 1+2+6=9
(E-a) Travel time for route 1-2-4-6 = 1+3+6=10
(I-b) Travel time for route 1-2-5-6 = 1+3+2=6
(E-b) Travel time for route 1-2-5-6 = 1+3+1=5
(I-c) Travel time for route 1-2-5-6 = 1+2+2=5
(E-c) Travel time for route 1-3-5-6 =1+1+1=3

Figure 2.6: Different shortest routes obtained by instantaneous travel-time and experienced travel-time approaches with departure time 2, Chiu et al. [12]

Chapter 3

Distributed Traffic Management

In this chapter, firstly I propose the distributed traffic management problem and its dynamic distributed constraint optimisation problem model. Secondly, I introduce an infrastructurelessly decentralised traffic information system, which is an alternative to the centralised traffic information system. Finally, I describe the decentralised multi-agent coordination algorithm for solving the proposed problem based on support-based distributed optimisation algorithm combined with auction theory.

3.1 Distributed Dynamic Traffic Assignment Problem

Essentially, the Distributed Dynamic Traffic Assignment (DDTA) Problem is a *multi-agent optimisation problem*, where the travellers in the road network are modelled as *autonomous vehicle agents* that are capable of making decision based on local view of global traffic system. In particular, such vehicle agents must coordinate their route plans in order to minimise the total travel time, which is the sum of travel times that all vehicle agents experienced during their trips.

Formally, the distributed traffic management problem is defined as follows:

- A *road network* is represented by a *directed graph* $G = (V, E)$, where V is a set of nodes, $V = \{v_1, v_2, \ldots, v_p\}$, that represents the *intersections* in the road network, E is a set of *edges*, $E = \{e_1, e_2, \ldots, e_q\}$, that represents the roads, which are referred as *links*. Each link e_i has a length l_{e_i}, a capacity c_{e_i} and a maximum allowed speed w_{max_i}. The capacity c_{e_i} of the link e_i is the maximal number of vehicle agents that are allowed to cross this link at a certain time window.

- A set $\mathcal{A}$ of *vehicle agents*, $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$, where each vehicle A_i is situated at start location s_i (a node in graph G) and desires to go to destination location z_i (another node in G) at a departure time, denoted by t_{s_i}.

The *time horizon* of the system T is discretised into a set of time slots, where each of them is denoted by t_i. Therefore, $T = \{t_1, t_2, \ldots, t_m\}$. Without loss of generality, the duration of time slot is set at 1 time unit in this proposed problem. For each time slot, vehicle agent A_i moves forward from one position to another position at its current speed. However, vehicle agent might stop and wait for the next move because of the traffic jams.

A *route* of vehicle agent A_i is denoted by $\mathcal{P}_i$, which consists of connected links, $\mathcal{P}_i = \{e_1, e_2, \ldots, e_{n'}\}$. I use v_{j_1}, v_{j_2}, where $v_{j_1}, v_{j_2} \in V$, to denote start node and end node of link e_j respectively. Therefore, two links e_j, e_k are supposed to be *connected* if v_{j_2} and v_{k_1} are identical. Vehicle agents A_i are capable of rerouteing in order to optimise their travel times and possibly, the total travel time of overall system.

An *experienced route* denoted by $\mathcal{P}_i^*$ of vehicle agent A_i is the actual route that was taken by this vehicle agent in order to arrive at destination location. An *experienced travel time* is an amount of time that was spent by vehicle agent following its experienced route.

The traffic flow Q on a link e_i is the number of vehicles (N) traversing this link during a time window $\triangle t_k$, where $\triangle t_k = [t_{k_1}, t_{k_2}]$, $t_{k_1}, t_{k_2} \in T$ and $t_{k_1} < t_{k_2}$.

$$Q_{\triangle t_k} = \frac{N}{t_{k_2} - t_{k_1}} \tag{3.1}$$

In the DDTA problem, the following *capacity constraint* on the traffic flow $Q_{\triangle t_k}$ must be satisfied to make sure that the number of vehicle agents on a link during a given time window $\triangle t_k$ does not exceed the link capacity c_{e_i}:

$$Q_{\triangle t_k} \leq \frac{c_{e_i}}{t_{k_2} - t_{k_1}} \tag{3.2}$$

Moreover, no any kind of central authority exists in the proposed model. Especially, vehicle agents must coordinate their routes by communicating with each other in a decentralised manner through either cellular network, 3G or Dedicated Short-Range Communications (DSRC). Each vehicle agent is supposed to be equipped with the following on-board hardware, which consists of:

- A *geographical information system* (GIS) with the global positioning system (GPS). Note that the maps, which are used by vehicle agents, must be identical.
- An *on-board navigation device.* This device is used for directing vehicle agent to the destination location.
- An *in-vehicle computing processor.* This processor is capable of processing received messages and computing a shortest path between two nodes of the map.

The *traffic pattern* at time slot $t_i \in T$ is comprised of the positions of all vehicle agents and their current routes. A *social cost* of a given traffic pattern at time slot t_i is the sum of expected travel times that vehicle agents will experience when following routes of the traffic pattern above. I use Ω_i, u_i to denote this traffic pattern and its associated social cost at time slot t_i respectively, then the *snapshot* problem of DDTA problem, denoted by p_i is defined as:

Definition 3.1.1 (Snapshot problem of DDTA problem)
The snapshot problem p_i of DDTA problem is an optimisation problem that is specific to the traffic pattern Ω_i at time slot t_i. The solution to the problem p_i is the set of routes for all en-route vehicle agents that satisfies the capacity constraint (Eq. 3.2) and minimise the social cost u_i given the traffic pattern Ω_i.

Therefore, the DDTA problem can be divided into a sequence of snapshot problems $\langle p_1, p_2, \ldots, p_m \rangle$, where each of them is appropriate for each time slot t_i, $t_i \in T$ and $|T| = m$.

Finally, the goal of DDTA problem is solving a sequence of snapshot problems $\langle p_1, p_2, \ldots, p_m \rangle$.

3.2 Dynamic Distributed Constraint Optimisation Model

In this section, firstly I model each snapshot problem of the DDTA problem as a DCOP and then the DDTA problem as a DynDCOP. Initially, let me consider the snapshot problem p_i at time slot t_i by taking account of the follows:

3.2.1 Variables

Vehicle agents in DDTA problem are referred to agents in DCOP model. Variable x_i within a set $X = \{x_1, x_2, \ldots, x_n\}$, which is controlled by vehicle agent A_i, represents a current route that this agent is following.

The domain d_i of variable x_i is a finite set of possible routes from its current location to destination location that vehicle agent can take. It is not necessary to enumerate all of these possible routes for the domain d_i. Therefore, d_i consists of the possible routes that include the shortest route and its alternatives. An alternative to shortest route is a route that has the same start and destination locations, but it replaced some links of shortest route by another ones.

3.2.2 Constraints

3.2.2.1 Capacity Constraint

The capacity constraint on traffic flow (Eq 3.2) is the n-ary shared constraint between routes assigned to variables that are controlled by vehicle agents. Actually, vehicle agents share the constraint on the number of vehicles on a link if they will enter this link within a same time window.

Formally, let me define the capacity constraint C_1^i over the set of variables $\{x_1, x_2, \ldots, x_m\}$ as following:

$$C_1^i : (d_1 \times d_2 \times \ldots \times d_m) \rightarrow \begin{cases} Satisfied & \text{if } N \leq c_{e_k} \\ Unsatisfied & \text{if } otherwise \end{cases} \tag{3.3}$$

where:

$$N = \sum_{i=1}^{m} g(\mathcal{P}_i), \; where \begin{cases} g(\mathcal{P}_i) = 1 & \text{if } e_k \in \mathcal{P}_i \; and \; \alpha_{e_k}^i \in \tau_k^j \\ g(\mathcal{P}_i) = 0 & \text{if } otherwise \end{cases}$$

and

- d_i is the domain of variable x_i controlled by vehicle agent A_i,
- N is the number of vehicle agents on link e_k at certain time window $[t_p, t_q]$,
- c_{e_k} is the capacity of link e_k,
- $\mathcal{P}_i$ is the route that is assigned to variable x_i of vehicle agent A_i from domain d_i,

- $\alpha^i_{e_k}$ is the estimated time of arrival at link e_k by vehicle agent A_i,
- τ^j_k is the considering time block for evaluating capacity constraint C^i_1,
- $e_k \in E$, $E = \{e_1, e_2, \ldots, e_q\}$.

3.2.2.2 Expected Travel Time Constraint

The purpose of defining the expected travel time constraint (unary constraint) in DCOP model of snapshot problem p_i is twofold. First, in this model individual travellers have the right to reject the suggestions on optimal routes from vehicle agents. For example, an individual traveller might try to across a link that is not allowed to travel through according to the optimal route suggested by the vehicle agent. Therefore, if the number of these drivers is large enough, it will cause the significant increase in travel times on a number of links in the road network.

Second, this model also considers the events that might happen suddenly on some links in the road network such as: road works, traffic accident, etc. Therefore, these events cause the traffic jams that increase the expected travel times on a number of links.

Finally, the expected travel time constraint denoted by C^i_2 on a route $\mathcal{P}_i$ of vehicle agent A_i is defined as following:

$$C^i_2 : d_i \rightarrow \begin{cases} Satisfied & \text{if } \forall e_k \in \mathcal{P}_i \mid \beta_{e_k} \leq \frac{l_{e_k}}{w_{f_k}} \\ Unsatisfied & \text{if } otherwise \end{cases} \tag{3.4}$$

where:

- d_i is the domain of variable x_i controlled by vehicle agent A_i,
- $\mathcal{P}_i$ is the route that is assigned to variable x_i of vehicle agent A_i from domain d_i,
- β_{e_k} is the expected travel time of link e_k,

- l_{e_k} is the length of link e_k,
- w_{f_k} is the free-flow speed on link e_k,
- $e_k \in E$, $E = \{e_1, e_2, \ldots, e_q\}$.

3.2.2.3 Route Valid Constraint

The route valid constraint, which is the unary constraint, is introduced to the DCOP model of snapshot problem p_i especially for the situation where the road closure event happened. When the road is closed, it means that this link is temporally eliminated from the road network. Therefore, the links of a vehicle agent's route need to be checked to determine whether or not they are connected.

The route valid constraint denoted by C_3^i on a route $\mathcal{P}_i$ of vehicle agent A_i is defined as following:

$$C_3^i : d_i \rightarrow \begin{cases} Satisfied & \text{if } \forall e_k \in \mathcal{P}_i \mid e_k \in E \\ Unsatisfied & \text{if } \exists e_k \in \mathcal{P}_i \mid e_k \notin E \end{cases} \tag{3.5}$$

where:

- d_i is the domain of variable x_i controlled by vehicle agent A_i,
- $\mathcal{P}_i$ is the route that is assigned to variable x_i of vehicle agent A_i from domain d_i,
- E is the set of all links in the road network.

3.2.3 Objective function

The *cost* of a route can be interpreted as the sum of expected travel times of all links of this route. The *objective function* O_i of a snapshot problem p_i is the sum of costs of all vehicle agents's routes at time slot t_i .

Let me define the objective function O_i over the set of variables $X_i = \{x_1, x_2, \ldots, x_m\}$ for snapshot problem p_i at time slot t_i as following:

$$O_i : (d_1 \times d_2 \times \ldots \times d_m) \rightarrow \sum_{j=1}^{m} \sum_{e_k \in \mathcal{P}_i} \beta_{e_k} \tag{3.6}$$

and

- d_i is the domain of variable x_i controlled by vehicle agent A_i,
- $\mathcal{P}_i$ is the route that is assigned to variable x_i of vehicle agent A_i from domain d_i,
- β_{e_k} is the expected travel time of link e_k,
- $e_k \in E$, $E = \{e_1, e_2, \ldots, e_q\}$.

The solution to snapshot problem p_i, which is modelled as a DCOP, is satisfying all the aforementioned constraints C_1^i, C_2^i, C_3^i and minimising the objective function O_i:

$$\arg\min_{X_i} O_i \tag{3.7}$$

As mentioned before, the DDTA problem is a sequence of snapshot problems, where each of them p_i is appropriate for each time slot t_i, $t_i \in T$ and $|T| = m$. Therefore, I model DDTA problem as DynDCOP, which is a sequence of DCOPs $\langle DCOP_1, DCOP_2, \ldots, DCOP_m \rangle$, where each $DCOP_i$ is the DCOP model of snapshot problem p_i.

3.3 Infrastructurelessly Decentralised Traffic Information System

The expected travel times of links are necessary for evaluating two constraints C_1^i (Eq 3.3), C_2^i (Eq 3.4) and the objective function O_i (Eq 3.6) in DynDCOP model of proposed DDTA problem. Typically, vehicle agents are able to calculate the expected travel times of links based on the real-time traffic condition updates received from central traffic information system, such as Advanced Traffic Information System (ATIS). However, such system is costly in terms of its installation, operation and maintenance.

In this section, an Infrastructurelessly Decentralised Traffic Information System (IDTIS) is presented especially for DDTA problem as well as its DynDCOP model. Taking advantage of V2V communication technologies such as DSRC, vehicle agents are committed to developing, operating and maintaining IDTIS in a decentralised manner. The traffic information system built by vehicle agents is completely independent from infrastructure and thus the cost of IDTIS is reduced substantially in comparison with ATIS. Moreover, because of the absence of centralised entity in DDTA problem, IDTIS becomes an appropriate and efficiency tool that supplements the approach described in section 3.5.1 for solving DDTA problem.

3.3.1 Broadcaster Agent

In order to develop IDTIS, the following assumptions can be made feasibly by exploiting the current DSRC technology developments:

1. Vehicle agent is able to determine is there any other vehicle agent that is also occupying the same link.
2. Vehicle agents, which are on the same link, are capable of identifying which is vehicle agent among them is closest to start node of this link.
3. For all vehicle agents traversing on the same link, the message sent by one of them will arrive immediately at others at the same time.

For developing IDTIS, a concept of *broadcaster agent* is defined as follows:

Definition 3.3.1 (Broadcaster agent)

*The **broadcaster agent** B_{e_k} of link e_k is a vehicle agent, which is responsible for broadcasting the estimated travel time of the link e_k to all vehicle agents in the traffic system.*

In IDTIS, there are two types of message, which are described as follows:

- Message M_A contains information about the experienced travel time and the link's occupation status of a broadcaster agent. This status is marked as *True* if this broadcaster agent is traversing on the link and *False* if it left this link. M_A is sent by a broadcaster agent to vehicle agents that occupy the same link.

- Message M_B contains information about the estimated travel time of a link. M_B is sent by broadcaster agents to all vehicle agents in the traffic system.

The formats of message M_A and message M_B will be described in Table 3.1 and Table 3.2 respectively.

Algorithm 1 describes a process of becoming a broadcaster agent of a vehicle agent. Lines 2-4 are about the situation where vehicle agent $self$ has recently entered link e_k. In this situation, if there is not any other vehicle agent that is occupying link e_k then $self$ becomes the broadcaster agent B_{e_k} of link e_k.

When occupying link e_k, B_{e_k} will send message M_A to all vehicle agents that are also on link e_k for every constant amount of time denoted by $UPDATE\_TIME$ (Lines 11-16). The value $True$ of message M_A's occupation status indicates that B_{e_k} is still on link e_k (Line 13). When finishing traversing link e_k, B_{e_k} will "announce" its completion of being the broadcaster agent of link e_k to all vehicle agents on link e_k by broadcasting message M_A with occupation status $False$ (Lines 17 -20).

After receiving the "completing message" M_A from B_{e_k}, which has recently left link e_k, vehicle agent A_i will determine whether or not it can become the broadcaster

```
   Data:
1  begin
2      if no $A_i$ occupies $e_k$ then
3          self becomes $B_{e_k}$
4      end
5      else if self is closest to $v_{k_1}$ and self received $M_A$ then
6          if $M_A$.occupation_status == False then
7              self becomes $B_{e_k}$
8          end
9      end
10     if self is $B_{e_k}$ then
11         for every UPDATE_TIME do
12             if self is on $e_k$ then
13                 $M_A$.occupation_status ← True
14                 Send $M_A$ to all other $A_i$ occupied $e_k$
15             end
16         end
17         if self left $e_k$ then
18             $M_A$.occupation_status ← False
19             Send $M_A$ to all other $A_i$ occupied $e_k$
20             Resign from broadcaster agent of $e_k$
21         end
22     end
23 end
```

Algorithm 1: Process of becoming a broadcaster agent and its operation

agent of link e_k. If current position of A_i is closest to start node v_{k_1} of link e_k then A_i becomes the broadcaster agent of this link (Lines 5-9). This process repeats over and over again to guarantee that there is only one broadcaster agent for link e_k at any time.

I use γ_{e_k} to denote the travel time of the link e_k experienced by a broadcaster agent and β_{e_k} to denote the expected travel time of link e_k. Recent broadcaster agent B_{e_k} is able to compute the expected travel time β_{e_k} of the link e_k based on its current travel speed w and travel time γ_{e_k} (extracted from message M_A) experienced by the previous

	Message M_A	
	`message_ID`	ID of message M_A
	`sender_ID`	ID of broadcaster agent (vehicle agent ID)
	`link_ID`	ID of link that broadcaster agent is occupying
	`status`	Occupation status of broadcaster agent (True/False)
	`exp_travel_time`	Travel time experienced by broadcaster agent
Time block 1	`registered_Vehicles`	List of vehicle agents registered to DA auction for time block 1
	`auctioneer_agent_ID`	ID of auctioneer agent for time block 1
Time block 2	`registered_Vehicles`	List of vehicle agents registered to DA auction for time block 2
	`auctioneer_agent_ID`	ID of auctioneer agent for time block 2
Time block 3	`registered_Vehicles`	List of vehicle agents registered to DA auction for time block 3
	`auctioneer_agent_ID`	ID of auctioneer agent for time block 3
...	...	...
	...	...
Time block n	`registered_Vehicles`	List of vehicle agents registered to DA auction for time block n
	`auctioneer_agent_ID`	ID of auctioneer agent for time block n

Table 3.1: Data structure of message M_A

broadcaster agent as follows:

$$\beta_{e_k} = \frac{1}{2}(\frac{l_{e_k}}{w} + \gamma_{e_k}) \tag{3.8}$$

where

l_{e_k} is the length of link e_k.

After computing β_{e_k}, B_{e_k} broadcast message M_B to all vehicle agents in the traffic system. This message includes the β_{e_k} and the format of this message will be described in more detail in Table 3.2. After receiving messages M_B, vehicle agents update their own expected travel time of link e_k, which is already stored in their storage devices.

	Message M_B	
	`message_ID`	ID of message M_B
	`sender_ID`	ID of sender agent (broadcaster agent)
	`link_ID`	ID of link
	`est_travel_time`	Estimated travel time of link
Time block 1	`auctioneer_agent_ID`	ID of auctioneer agent for time block 1
Time block 2	`auctioneer_agent_ID`	ID of auctioneer agent for time block 2
Time block 3	`auctioneer_agent_ID`	ID of auctioneer agent for time block 3
...	...	...
Time block n	`auctioneer_agent_ID`	ID of auctioneer agent for time block n

Table 3.2: Data structure of message M_B

3.4 Auctions

There are two types of auctions that are designed for solving DDTA problem using decentralised multi-agent coordination algorithm as follows:

- Auction 1, denoted by Φ, is used for determining which vehicle agent will become an auctioneer agent.
- Auction 2, denoted by Ψ, is used first for discovering constraints between vehicle agents and then granting permissions for them to cross a link at a certain time window.

Figure 3.1 illustrates different time points related to two aforementioned auctions in chronological order as follows:

- *Auction 1 open time.* Time point when Φ is opened for bidding.
- *Auction 1 close time.* Time point when Φ is closed for bidding and then is conducted.
- *Auction 2 register time.* Time point when Ψ is opened for vehicle agents to register their intentions of bidding. Note that *auction 2 register time* and *auction 1 close time* are identical.

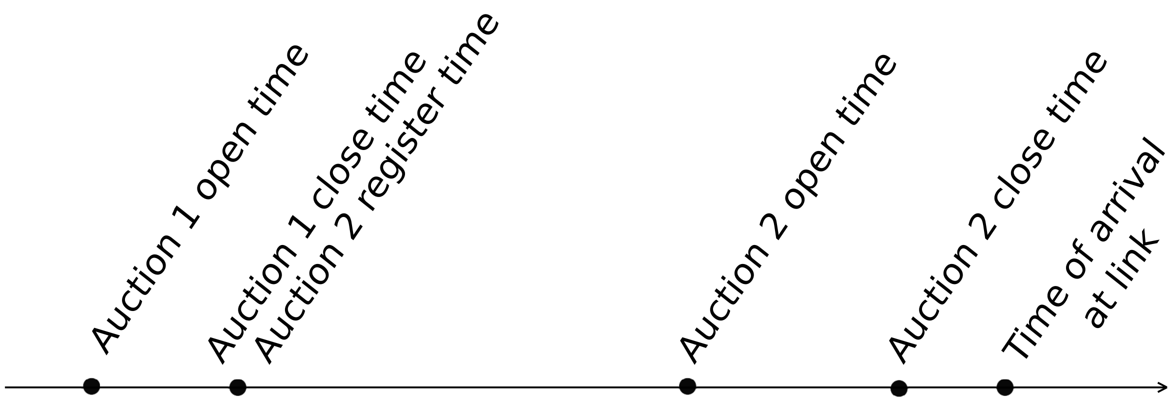

Figure 3.1: Timeline

- *Auction 2 open time.* Time point when Ψ is opened for bidding.
- *Auction 2 close time.* Time point when Ψ is closed for bidding and then is conducted.
- *Time of arrival at link.* Time point when vehicle agent starts crossing a link.

3.4.1 Auction of Determining Auctioneer Agent

Every vehicle agent needs to participate in an auction to become an auctioneer agent. The incentive of becoming an auctioneer agent is having the privilege to cross the link first among vehicle agents. Moreover, when an auctioneer agent is determined, it can assign a set of vehicle agents can cross the link leading to reduction of total travel time of all vehicle agents. Vehicle agent should not pay anything to become an auctioneer agent but for my approach every vehicle agent should make a bid for crossing a link. In this approach it is assumed that vehicle agent should follow the rule that it can cross the link if it is allowed by auctioneer agent.

Definition 3.4.1 (Auctioneer agent)
Auctioneer agent *Λ_k^j is a vehicle agent that holds the auction Ψ_k^j of granting permissions for a set of vehicle agents to travel through link e_k during time block τ_k^j.*

Following the optimal route returned by DynDCOP solving system, vehicle agent has to bid to auction Ψ_k^j for having the privilege of passing through link at during time block τ_k^j. Moreover, when vehicle agents registered their intentions of bidding to auction Ψ_k^j, the auctioneer agent, which controls Ψ_k^j, is able to determine the set of vehicle agents that share the capacity constraint on a number of vehicle agents on a link during time block τ_k^j (constraint 3.3). Theses vehicle agents become neighbouring agents of each other in DynDCOP model of DDTA problem. Therefore, SBDO agent starts finding the optimal value for its variable at the time when neighbouring agents are identified.

Definition 3.4.2 (Auction Φ)

Auction Φ_k^j *is an auction, which is held by a broadcaster agent* B_{e_k} *for determining the auctioneer agent* Λ_k^j *of link* e_k *associated with time block* τ_k^j.

For my approach, each vehicle agent A_i needs to bid to auction Φ_k^j for becoming the auctioneer agent of the link, across which it will travel during a time block. I use φ_i^{jk} to denote the bid of vehicle agent A_i to auction Φ_k^j, then the amount of this bid is defined as:

$$\varphi_i^{jk} = \frac{1}{\mu_{ik}} \tag{3.9}$$

where

μ_{ik} is the distance from current position of A_i to start node of link e_k.

After the auction Φ_k^j was conducted, the vehicle agent with the highest bid will become an auctioneer agent.

3.4.2 Auction of Granting Permission for Travelling

Definition 3.4.3 (Auction Ψ)

Auction Ψ_k^j *is an auction, which is held by auctioneer agent* Λ_i^j *for granting permissions* *for a set of vehicle agents to travel through link e_k during time block τ_i^j.*

Definition 3.4.4 (Expected total travel time)

The expected total travel time $\delta_{\mathcal{P}}$ of route $\mathcal{P}$ is the sum of expected travel times of all links within route $\mathcal{P}$.

$$\delta_{\mathcal{P}} = \sum_{e_k \in \mathcal{P}} \beta_{e_k} \tag{3.10}$$

Vehicle agents have to bid for having privileges of travelling through the link e during a time block. Let me use ψ_i^{jk} to denote the bid of vehicle agent A_i to auction Ψ_k^j. $\mathcal{P}^*$ is used to denote the shortest alternative route to current route $\mathcal{P}$ of the vehicle agent A_i that doesn't contain the link e. Then, the amount of ψ_i^{jk} is defined as:

$$\psi_i^{jk} = \delta_{\mathcal{P}*} - \delta_{\mathcal{P}} \tag{3.11}$$

3.5 Algorithms

3.5.1 Decentralised Multi-Agent Coordination Algorithm

Algorithm 2 presents the pseudocode of the decentralised multi-agent coordination (DMAC) algorithm, which is the combination of SBDO and auctions (auction Φ and auction Ψ). DMAC algorithm also describes the operation of a vehicle agent during its journey from start location to destination location. Besides acting as vehicle agent, vehicle agent participates in SBDO solving system as a SBDO agent. However, DMAC merges vehicle agent and SBDO agent into unique agent. Therefore, the term “vehicle agent” refers to vehicle agent as well as SBDO agent.

Lines 2-4 describe the initialisation process of a vehicle agent `self`. The shortest route and its alternatives are added to vehicle agent's domain of vehicle agents (Line 2). Moreover, according to SBDO algorithm, the value of variable `self.value` is the

```
Data: Road Network
begin
    Initialise self.domain
    Initialise self.value by choosing the best value from self.domain
    self.route ← Null; self.registered_auctioneer_agents ← $\emptyset$
    while self did not arrive at destination location do
        if self.route $\neq$ self.value then
            self.route ← self.value
            self.domain ← Update_Domain(self.route)
            foreach $e$, $\alpha_e$ in self.route do
                if self.time $\geq \alpha_e$ - AUCTION1_OPEN_TIME then
                    broadcaster_agent ← Get_broadcaster($e$, $\alpha_e$)
                    Bid $\varphi_i^{\alpha_e}$ to auction $\Phi_e^{\alpha_e}$ held by broadcaster_agent
                end
                if self.time $\geq \alpha_e$ - AUCTION2_REGISTER_TIME then
                    if self won auction $\Phi_e^{\alpha_e}$ then
                        auctioneer_agent ← self
                    else
                        auctioneer_agent ← Get_auctioneer_agent($e$, $\alpha_e$)
                    end
                    if auctioneer_agent not in self.registered_auctioneer_agents
                    then
                        Register self with auctioneer_agent
                        foreach agent in auctioneer_agent.registered_agents do
                        self.Add_Neighbour(agent)
                        Add auctioneer_agent to self.registered_auctioneer_agents
                    end
                else if self.value is not changed and self.time $\geq \alpha_e$ -
                AUCTION2_OPEN_TIME then
                    Bid $\psi_e^{\alpha_e}$ to auction $\Psi_e^{\alpha_e}$ held by auctioneer_agent
                else if self.time $\geq \alpha_e$ - AUCTION2_CLOSE_TIME then
                    if self won auction $\Psi_e^{\alpha_e}$ then
                        self is allowed to cross link $e$
                    else
                        Eliminate self.value permanently from self.domain
                        Change self.route to another self.value
                    end
                    Request reserve space from auctioneer_agent
                end
            end
        end
        foreach message in self.received messages do  Process message
        select_support()
        update_view()
        Update self.value from self.view
        foreach agent in neighbouring_agents do  send_update(agent)
    end
end
```

Algorithm 2: Decentralised Multi-Agent Coordination Algorithm

current best assignment of vehicle agent from its domain. Therefore, at the beginning of vehicle agent's journey, the shortest route is assigned to `self.value` (Line 3). Variable `self.route` represents the route that vehicle agent is following. Variable `self.registered_auctioneer_agents` represents the set of auctioneer agents, which hold the auctions that vehicle agent registered its intention to bid (or bid). Initially, `self.route` and `self.registered_auctioneer_agents` are set as null and empty respectively.

In lines 6-7, if the current route `self.route` of vehicle agent is different from the optimal value suggested by SBDO system, then this value is assigned to `self.route`. Based on the new value assigned to `self.route`, the `self.domain` is updated in line 8.

Lines 9-36 are about the interaction between vehicle agent and auctions (Φ and Ψ). In lines 10-13, vehicle agent bids to auction $\Phi_e^{\alpha_e}$ for becoming an auctioneer agent when $\Phi_e^{\alpha_e}$ is open for biding. First, vehicle agent finds the current broadcaster agent of link e based on α_e - estimated time of arrival at link e (Line 11). I use $\Phi_e^{\alpha_e}$ to denote the auction Φ, which is held by `broadcaster_agent` for determining the auctioneer agent of link e during the time block that includes estimated time of arrival α_e. $\varphi_i^{\alpha_e}$ is used to denote the amount of bid of vehicle agent A_i to auction $\Phi_e^{\alpha_e}$. Second, vehicle agent bids $\varphi_i^{\alpha_e}$ to auction $\Phi_e^{\alpha_e}$ for becoming the auctioneer agent of link e (Line 12).

Lines 14-35 are about the constraint discovering process and auction Ψ (auction 2) participating of vehicle agent `self`. First, vehicle agent should fine the auctioneer agent of link e at time α_e (Lines 15-19). If vehicle agent won the auction $\Phi_e^{\alpha_e}$ from last bidding, vehicle agent becomes the auctioneer agent of link e during time block that includes α_e. However, if vehicle agent lost this auction, it should find the auctioneer agent and register its intention of bidding to this agent (Line 21).

Now, auctioneer agent is able to determine the set of vehicle agents that intend to cross link e during a time block. In other words, the constraints on capacity of link e between these vehicle agents are discovered (Line 22). Then each vehicle agent starts sending and receiving messages to/from another neighbouring vehicle agents. At this time, SBDO solving system began working as each vehicle agent will choose the best value for `self.value` by V2V communication in a decentralised manner.

First, if the `self.value` is not changed by SBDO agent, vehicle agent follows the current route `self.route`. In contrast to this, vehicle agent postpones current process and starts again the procedure described in Lines 6-37. Next, if auction $\Psi_e^{\alpha_e}$ is opened for bidding, vehicle agent bid $\psi_e^{\alpha_e}$ for having permission to cross link e during time block that includes α_e.

After auction $\Psi_e^{\alpha_e}$ is conducted, if vehicle agent won this auction, then it has the privilege to go through link e (Line 29). In opposition to this, vehicle agent removes permanently link e from its domain and change `self.route` to another `self.value` (Lines 31-32). For a vehicle agent that was not able to bid to $\Psi_e^{\alpha_e}$, this agent can request the available space of link e from auctioneer agent (Line 34).

Lines 38-32 are about the operation of SBDO agent (referred as vehicle agent). First, in line 38, vehicle agent processes the messages received from neighbouring agents including messages `isgood`, `nogood`, `add_constraint`, `remove_constraint`, etc. Second, vehicle agent select supporter agent and then update its view based on the `isgood` of supporter (Lines 39-40). Next, in line 41, the `self.value` is updated by extracting the best assignment of vehicle agent from `self.view`. Finally, vehicle agent sends updates about its new `self.view` to neighbouring vehicle agents (Line 42).

3.5.1.1 Auctioneer Agent

Auctioneer agent Λ_k^j controls the traffic flow on link e_k, which is the number of vehicle allowed for travelling through during a time block τ_k^j. An auctioneer agent has three phases during its operating time: registering, auction opening and auction closing

```
   Data:
1  begin
2      if self.$time$ $\geq$ self.$auction\_time$ - AUCTION2_REGISTER_TIME then
3          Register vehicle agent to self.$registered\_agents$
4      end
5      if self.$time$ $\geq$ self.$auction\_time$ - AUCTION2_OPEN_TIME then
6          foreach $A_i$ in self.$registered\_agents$ do
7              Receive bid for $A_i$
8              Store $A_i$ with bid
9          end
10     end
11     if self.$time$ $\geq$self.$auction\_time$ - AUCTION2_CLOSE_TIME then
12         self.$winners$ $\leftarrow$ Select $c$ agents with highest bids
13         self.$loosers$ = self.$registered\_agents$ $\setminus$ self.$winners$
14         foreach $winner$ in self.$winners$ do
15             Send message to winner agent about winning auction
16         end
17         foreach $looser$ in self.$loosers$ do
18             Send message to looser agent about loosing auction
19         end
20     end
21 end
```

Algorithm 3: Auctioneer Agent

phases (Lines 2-20). In registering phase, auctioneer agent process request for registering from vehicle agent and add this agent to the list $self.registered\_agents$ of its registered agents (Lines 2-4). When auctioneer agent in auction opening phase, vehicle agents are allowed to bid for their privileges of travelling through the link during certain time block (Line 7). In line 8, auctioneer agent stores a list of registered agents with appropriate bids for processing in the last phase. In closing phase (Lines 11-20), auctioneer agent conducts auction and determines which vehicle agents are the winners by selecting c number of bidding agents with highest bids (c equal to the capacity of link). Next auctioneer agent informs all winners - vehicle agents by sending messages about their obtained privileges for travelling auctioneer agent's link (Lines 14-16). In lines 17-19, these loosers are also noticed about their losses of having their rights to travel their bidding links. These loosing vehicle agents will add this link to theirs list of forbidden links and would be forced to find a new route.

3.5.2 Decentralised Uncoordination Algorithm

In decentralised uncoordination (DECU) algorithm, vehicles plan their routes by considering the current traffic condition. Based on the estimated travel times on roads, A^* is used by vehicles for searching the best route with respect to minimum estimated travel time. Vehicles will follow their planned routes until they arrived at their destination locations ignoring any factor that might affect their travel time.

3.5.3 Centralised Coordination Algorithm

In centralised coordination (CECO) algorithm, central server is responsible for planning routes for all vehicles. Vehicles send their queries to central server for requesting the optimal routes from their start to destination locations. First, central server uses A^* search for computing shortest routes for all vehicles. Next central server analyses these routes to identify all points where the capacity of a link is exceeded. Then it can re-route vehicles until there is not any congestion on links of map. Finally, central server informs the vehicles of their optimal routes and vehicles take these routes until they reached to destination locations.

3.5.4 Example

In order to demonstrate how decentralised multi-agent coordination algorithm works, we provide an example of algorithm execution illustrated with Figure 3.2. In the map of this figure, five vehicle agents A_1, A_2, A_3, A_4, A_5 have to travel from their start locations to destination locations. In particular, vehicle agents A_1, A_2 start at location **A** and their destination locations are **H**. Next, vehicle agent A_3 begins a trip at **K** and its goal location is **Q**. The last three ones A_4, A_5 depart at **R** and want to arrive

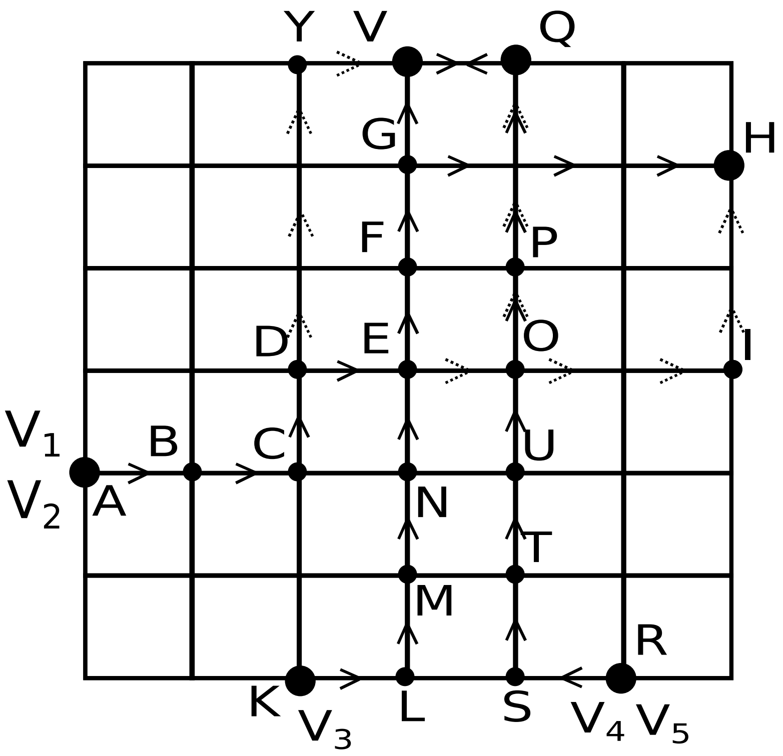

Figure 3.2: Example

at **V**.

The map in Figure 3.2 is a Manhattan grid (6×6) and the travel time of every link is 2 seconds. For each link, there are a set of auctioneer agents and auctions Ψ that appropriate to the set of time blocks, whose duration is 2 seconds. AUCTION2_REGISTER_TIME, AUCTION2_OPEN_TIME and AUCTION2_CLOSE_TIME are 6s, 4s, 2s respectively. The capacity for each link is limited to 2, except 1 for link **OP**.

For simplicity's sake, suppose that the broadcaster agents and auctioneer agents are known for all links w.r.t different time blocks. Therefore, these aforementioned vehicle agents are unnecessary to bid to auctions Φ for becoming auctioneer agents.

At time $\mathbf{t = 0s}$, all vehicle agents initialise their routes, which are the shortest routes from their start locations to destination locations. Suppose we have the following first routes for these vehicles:

- $\mathcal{P}_1 = \mathcal{P}_2$: **A-B-C-D-E-F-G–H**
- $\mathcal{P}_3$: **K-L-M-N-E-F-G-V-Q**
- $\mathcal{P}_4 = \mathcal{P}_5$: **R-S-T-U-O-P–Q-V**

As we can see in figure 3.2, **EF, OP** is a link that has possibly the number of vehicle agents exceeds theirs capacities at time $t = 8s$.

At time $\mathbf{t = 2s}$, A_1, A_2, A_3 register to auctioneer agent Λ^8_{EF} for travelling through link **EF** at $t = 8s$. A_4, A_5 register to auctioneer agent Λ^8_{OP} for cross link **OP** at $t = 8s$ Because the AUCTION2_REGISTER_TIME for all auctioneer agents is 6s, so auctioneer agents Λ^8_{EF}, Λ^8_{OP} process registering requests from $\{A_1, A_2, A_3, A_4, A_5\}$. A_1, A_2, A_3 now become neighbours of each other as well as A_4, A_5 do. At this time, SBDO solving system starts working and returns the optimal value (route) for route variable of vehicle agent.

At time $\mathbf{t = 4s}$, A_1, A_2 continue travelling with their routes and A_3 is forced to reroute as value of its route variable is changed as A_3 accepted the suggestion from SBDO solving system. Suppose the new route for A_3 is **K-L-M-N-E-O-P–Q** and A_3 will follow this route until it reached to **Q**. In another group, A_5 follows its planned route and the route of A_4 is supposed to be changed to $\mathcal{P}^4_4$:**R-S-T-U-O-E-F-G-V**. $\mathcal{P}^4_4$ is used to denote the route of vehicle agent A_4 at time t = 4s.

At time $t = 4s$, auctioneer agent Λ^8_{EF} is opened and A_4 sends a bid to Λ^8_{EF} for its privilege of travelling through **EF**. The bid ψ^4_4 of A_4 is equal to marginal cost of not travelling through **EF**. Actually, suppose we have an alternative route $\mathcal{P'}^4_4$ for A_4 that doesn't contain **EF** is **R-S-T-U-O-E-D-Y–V**. Therefore $\psi^4_4 = cost(\mathcal{P'}^4_4) - cost(\mathcal{P}^4_4) = 20 - 16 = 4$. Vehicle agents A_1, A_2 also bid for their rights to travel on **EF** at $t = 8s$. Their alternative routes $\mathcal{P'}^4_1 = \mathcal{P'}^4_2$ that not included **EF** are **A-B-C-D-E-O-I-H**. Therefore, the bids ψ^4_1, ψ^4_2 of each A_1, A_2 is $\psi^4_1 = \psi^4_2 = cost(\mathcal{P'}^4_1) - cost(\mathcal{P}_1) =$

$cost(\mathcal{P'}_2^4) - cost(\mathcal{P}_2) = 18 - 18 = 0$. Auctioneer agent Λ_{EF}^8 at this time stored 3 bidders - vehicle agents A_1, A_2, A_4 with their bids 4, 0, 0 respectively.

At time $\mathbf{t} = \mathbf{6s}$, auctioneer agent Λ_{EF}^8 is closed for bidding and conducts the auction as Λ_{EF}^8 selects 2 bidders with highest bids from its list of bidders. In this case, the first highest bid belongs to vehicle agent A_4 and second one we choose randomly, suppose belongs to vehicle agent A_1. The winners of this auction are vehicle agents A_1, A_4, so that they will follow their routes $\mathcal{P}_1, \mathcal{P}_4^4$ until they reach their destination locations. The looser - vehicle agent A_2 changes its route to $\mathcal{P'}_2^4$ and will go along with this until it arrive in **H**.

The final routes for A_1, A_2, A_3, A_4, A_5 are followings, as they satisfy the condition that not exceed the capacity of every link in the map:

- $\mathcal{P}_1^*$: **A-B-C-D-E-F-G–H**,
- $\mathcal{P}_2^*$: **A-B-C-D-E-O-I-H**,
- $\mathcal{P}_3^*$: **K-L-M-N-E-O-P–Q**,
- $\mathcal{P}_4^*$: **R-S-T-U-O-E-F-G-V**,
- $\mathcal{P}_5^*$: **R-S-T-U-O-P–Q-V**.

Chapter 4

Experimental Results

This chapter first presents implementation details, then experiment settings and finally experimental results with three different traffic planners: decentralised uncoordination (DECU), centralised coordination (CECO) and decentralised multi-agent coordination (DMAC)

4.1 Implementation details

Three planners are implemented in Python language. For the simulation, I used Simulation of Urban Mobility (SUMO) [7] simulator. TraCI [37] is used for navigating vehicles simulated by SUMO simulator.

4.1.1 Map

Listing 4.1 illustrates a simplified XML code that is used for storing the map shown in Fig. 4.1. From its XML format, the structure of this map consists of the followings:

- **Node**. Each node has an identification `id`, a latitude `lat` and a longitude `lon`.
- **Way**. Each way (link) has an identification `id`, referenced nodes `nd`(start and end nodes), a length `length`, a maximum allowed speed `maxspeed` and a capacity

`capacity`.

Listing 4.1: Example of a XML map file

```
<?xml version="1.0" ?>
<map>
<config euclidean="True"/>
<node id="0" lat="0.0" lon="0.0015">
</node>
<node id="1" lat="0.00106066017178" lon="0.00106066017178">
</node>
<node id="2" lat="0.0015" lon="9.18485099361e-20">
</node>
...
<node id="23" lat="-0.00318198051534" lon="0.00318198051534">
</node>
<way id="1">
<nd ref="0"/>
<nd ref="1"/>
<tag k="highway" v="primary"/>
<tag k="length" v="127.657368569"/>
<tag k="maxspeed" v="70"/>
<tag k="capacity" v="13"/>
</way>
<way id="2">
<nd ref="1"/>
<nd ref="2"/>
<tag k="highway" v="primary"/>
<tag k="length" v="127.657368542"/>
<tag k="maxspeed" v="70"/>
<tag k="capacity" v="13"/>
</way>
...
```

```
<way id="40">
<nd ref="23"/>
<nd ref="15"/>
<tag k="highway" v="primary"/>
<tag k="length" v="166.792389876"/>
<tag k="maxspeed" v="70"/>
<tag k="capacity" v="17"/>
</way>
</map>
```

The XML file of map (Listing 4.1) then is converted to SUMO map format using NETCONVERT. The SUMO map is also stored using XML format and its simplified code example is shown in Listing 4.2. The SUMO map file will be used with with SUMO vehicle file as input to SUMO simulator.

Listing 4.2: Example of a SUMO XML map file

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- generated on Wed Feb 27 01:05:41 2013 by SUMO netconvert Version
    0.15.0
<?xml version="1.0" encoding="UTF-8"?>

<net version="0.13" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd/
    net_file.xsd">
    <edge id=":0_0" function="internal">
        <lane id=":0_0_0" index="0" speed="19.44" length="4.55" shape
            ="674.44,503.02 673.25,503.17 672.27,503.61 671.48,504.35
            670.88,505.39"/>
    </edge>
    <edge id=":0_1" function="internal">
```

```
        <lane id=":0_1_0" index="0" speed="19.44" length="12.36" shape="674.44,499.72 671.22,499.32 668.54,498.12 666.40,496.11 664.79,493.29"/>
    </edge>
     ...
    <edge id="-1" from="1" to="0" priority="9" type="highway.primary">
        <lane id="-1_0" index="0" speed="19.44" length="115.43" shape="617.32,608.11 661.75,501.58"/>
        <lane id="-1_1" index="1" speed="19.44" length="115.43" shape="620.37,609.39 664.79,502.85"/>
    </edge>
    <edge id="-10" from="10" to="9" priority="9" type="highway.primary">
        <lane id="-10_0" index="0" speed="19.44" length="232.96" shape="509.98,821.25 725.41,732.61"/>
        <lane id="-10_1" index="1" speed="19.44" length="232.96" shape="511.23,824.30 726.67,735.66"/>
    </edge>
    ...
    <junction id="0" type="priority" x="668.57" y="498.07" incLanes="17_0 17_1 8_0 8_1 -1_0 -1_1" intLanes=":0_0_0 :0_1_0 :0_2_0 :0_3_0 :0_4_0 :0_5_0 :0_11_0 :0_7_0 :0_8_0 :0_12_0 :0_13_0" shape="674.44,504.62 674.44,491.52 672.36,490.14 660.27,495.18 660.27,500.96 672.36,506.01">
        <request index="0" response="00000110000" foes="10000110000" cont="0"/>
        <request index="1" response="01110110000" foes="01111110000" cont="0"/>
       ...
        <request index="10" response="00000110001" foes="00000110001" cont="1"/>
    </junction>
   ...
    <junction id=":0_11_0" type="internal" x="666.95" y="494.18"
        incLanes=":0_6_0 -1_0 -1_1" intLanes=":0_1_0 :0_7_0 :0_8_0"/>
    <junction id=":0_12_0" type="internal" x="669.07" y="497.79"
        incLanes=":0_9_0 8_0 8_1" intLanes=":0_1_0 :0_2_0 :0_3_0
        :0_4_0 :0_5_0"/>
   ...
    <connection from="-1" to="-8" fromLane="0" toLane="0" via=":0_7_0
        " dir="r" state="M"/>
    <connection from="-1" to="-8" fromLane="1" toLane="1" via=":0_8_0
        " dir="r" state="M"/>
   ...
</net>
```

4.2 Traffic Demand

The traffic demand for experiment is generated and then will be converted to SUMO format. Listing 4.3 shows the traffic demand in XML format. Each vehicle agent consists of followings:

- A start location `source`, from which vehicle agent departs.
- A destination location `destination`, at which vehicle agent arrives and finishes its journey.
- A departure time `startTime`, when vehicle agent starts its trip.
- A beginning speed `speed` when vehicle agent departs from start location.
- A acceleration of speed of vehicle agent `acceleration`.

- A deceleration of speed of vehicle agent `decceleration`.
- The limited amount of carbon `emissions` that vehicle agent can emit during its journey.

For the experiment, a range from 400 to 1000 vehicle agents will be generated and converted to SUMO format XML files appropriately using utility NETCONVERT of SUMO suite.

Listing 4.3: Example of a vehicle file

```xml
<?xml version="1.0" ?>
<traffic>
    <vehicle acceleration="0.8" decceleration="4.5" destination="10"
        emissions="8.63121896361" source="5" speed="0" startTime="0"/>
    <vehicle acceleration="0.8" decceleration="4.5" destination="17"
        emissions="5.36890623795" source="21" speed="0" startTime="0"/>
    <vehicle acceleration="0.8" decceleration="4.5" destination="23"
        emissions="5.36890623795" source="19" speed="0" startTime="0"/>
    <vehicle acceleration="0.8" decceleration="4.5" destination="1"
        emissions="5.36890623795" source="12" speed="0" startTime="0"/>
    ...
</traffic>
```

Listing 4.4 illustrates the SUMO format of traffic demand. SUMO uses the *Krauss car-following model* for its simulation. The length of each car according to this model is set at 7.5 meters and the minimum allowed distance between two cars is 2.5 meters. Therefore,the capacity of a link can be calculated based on this information.

Listing 4.4: Example of a SUMO vehicle file

```xml
<routes>
     <vType id="vtype1" length="7.5" maxSpeed="70" minGap="2.5" vClass
         ="passenger" guiShape="passenger/sedan">
```

```
        <carFollowing−Krauss accel="0.8" decel="4.5" sigma="0.5" />
    </vType>
    <vehicle id="1" type="vtype1" depart="0" departPos="free"
        departSpeed="0">
        <route edges="−5 −4 −3 −19 " />
    </vehicle>
    <vehicle id="2" type="vtype1" depart="0" departPos="free"
        departSpeed="0">
        <route edges="−38 −22 −5 −4 −3 −2 −18 −34" />
    </vehicle>
    <vehicle id="3" type="vtype1" depart="0" departPos="free"
        departSpeed="0">
        <route edges="−36 −20 4 5 6 7 24 40" />
    </vehicle>
    <vehicle id="4" type="vtype1" depart="0" departPos="free"
        departSpeed="0">
        <route edges="21 −4 −3 −2" />
    </vehicle>
    ...
</routes>
```

4.3 Experiment Settings

The experiements is desgined to evaluate the efficeincy of decentralised multi-agent co-ordination (DMAC) algroithm in comparison with decentralised uncoordiation (DECU) and centralised coordination (CECO) algorithms. The criteria for evaluation include total travel time, total travel distance, percentage of used links and number of reroutes made by all vehicle agents. These criteria are explained as follows:

- *Total travel time.* Sum of experienced travel time of all vehicle agent in order to

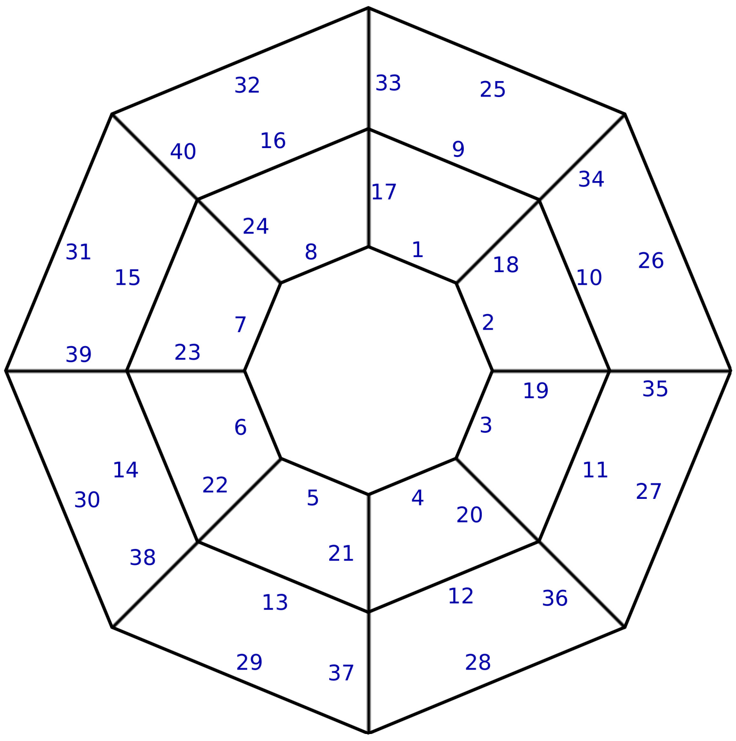

Figure 4.1: Map

arrive at destination locations.

- *Total travel distance.* Sum of distances that all vehicle agents travelled during their journeys.
- *Percentage of used links.* The percentage of links used by vehicle agents to travel on.
- *Number of reroutes.* Sum of times that vehicle agents changed their routes.

The map (ring road), which is used for the experiment, consists of 24 nodes and 40 links as shown in Fig. 4.1. The links of the map varies in length, capacity and maximum allowed speed. The length of these links are listed as follows:

- Links 1 - 8: **126.67** meters
- Links 9 - 16: **255.32** meters

Parameters		
parameter	meaning	value
`AUCTION1_OPEN_TIME`	Open time of auction Φ	80 seconds
`AUCTION1_CLOSE_TIME`	Close time of auction Φ	60 seconds
`AUCTION2_REGISTER_TIME`	Register time of auction Ψ	60 seconds
`AUCTION2_OPEN_TIME`	Open time of auction Ψ	40 seconds
`AUCTION2_CLOSE_TIME`	Close time of auction Ψ	15 seconds
`UPDATE_TIME`	Update time for sending M_A	2 seconds

Table 4.1: Parameters

- Links 25 - 32: **382.97** meters
- Links 17 - 24: **166.79** meters
- Links 33 - 40: **166.79** meters

For the traffic demand, the number of vehicle agents increased by 100 (vehicle agents) from 400 (vehicle agents) to 1000 (vehicle agents). The distance from start location to destination location of each vehicle agent was generated and maximised in order to simulate traffic congestion. Departure times of all vehicle agents are the same. However, if the number of vehicle agents on a link is too large, then simulator SUMO will control the order of departing for these agents.

In order to run simulation with DMAC planner, the values of parameters are set as shown in Table 4.1.

The amount of time block is calculated for each link according to the Definition **??**.

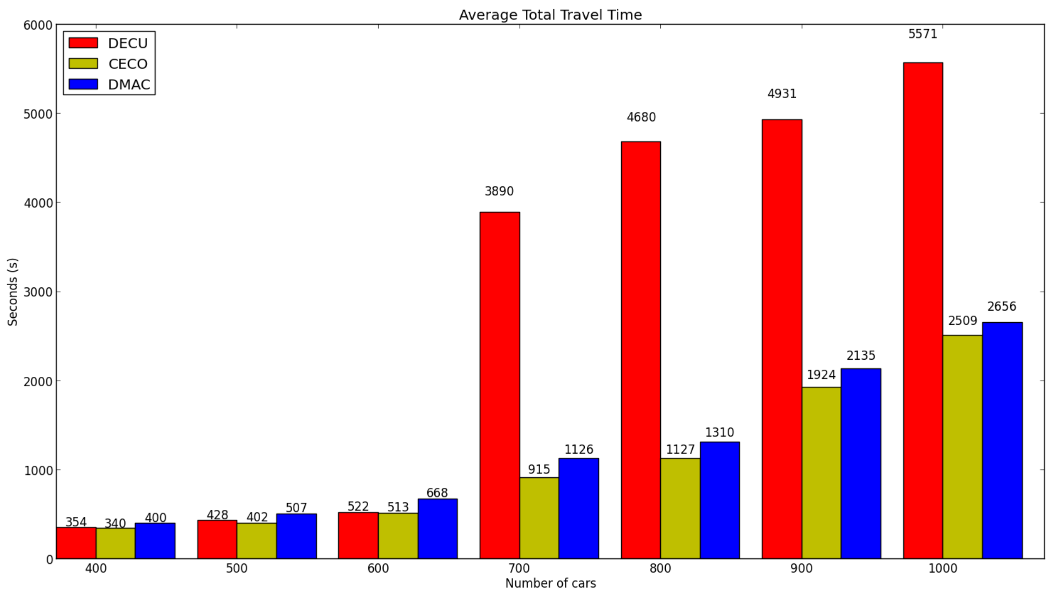

Figure 4.2: Average Total Travel Time

4.4 Experimental Results

Figure 4.2 discussion: In this section, I report averages over 10 experiments for the parameter settings in Table 4.1. Throughout the following sections, I evaluate three planners: DECU, CECO and DMAC based on four criteria:

- *Total travel time.* Sum of experienced travel time of all vehicle agent in order to arrive at destination locations.
- *Total travel distance.* Sum of distances that all vehicle agents travelled during their journeys.
- *Percentage of used links.* The percentage of links used by vehicle agents to travel on.
- *Number of reroutes.* Sum of times that vehicle agents changed their routes.

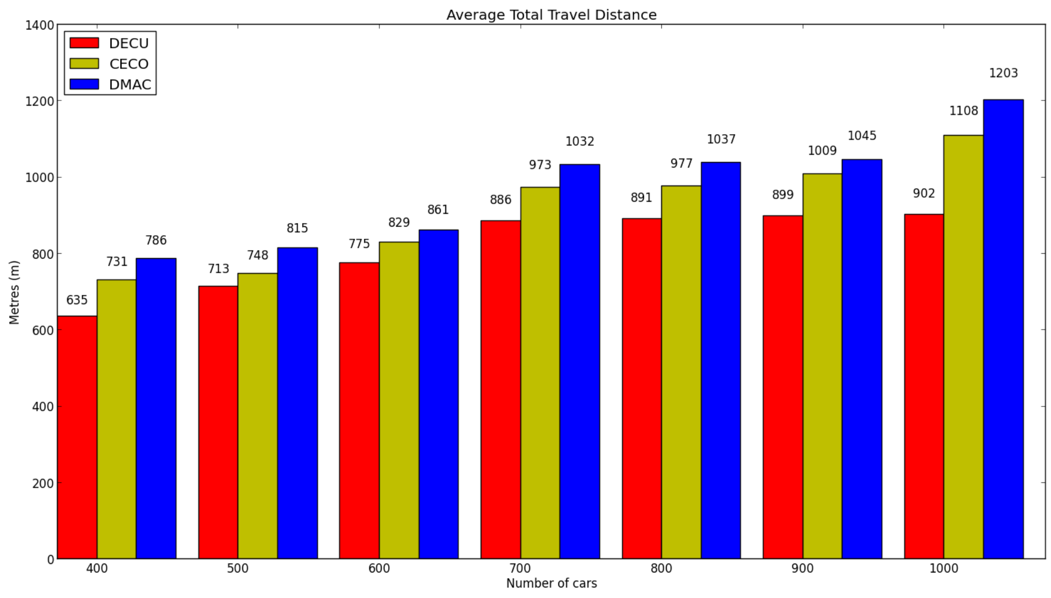

Figure 4.3: Average Total Travel Distance

4.4.1 Average Total Travel Time

Figure 4.3 discussion: Figure 4.2 presents the average total travel time of a number of vehicle agents ranging from 400 agents to 1000 agents. As shown in figure 4.2, the average total travel time dramatically increased from 552 seconds (600 agents) to 3890 seconds (700 agents) associated with DECU planner. This can be explained as traffic congestion occurred in traffic system with 700 vehicle agents.

When the number of vehicle agents is more than 700 agents, CECO and DMAC planners help the traffic system alleviate the traffic congestion. Specifically, DMAC reduced the average total travel time by 71% (700 agents), 72% (800 agents), 56.7% (900 agents), 52.3% (1000 agents) in comparison to 76.4% (700 agents), 75.9% (800 agents), 61% (900 agents), 54.9 % (1000 agents), by which CECO planner did. This result shows that the proposed DMAC planner outperforms DECU planner and its performance is close to CECO's one for congested network.

Surprisingly, when there was not traffic congestion, DECU planner works better

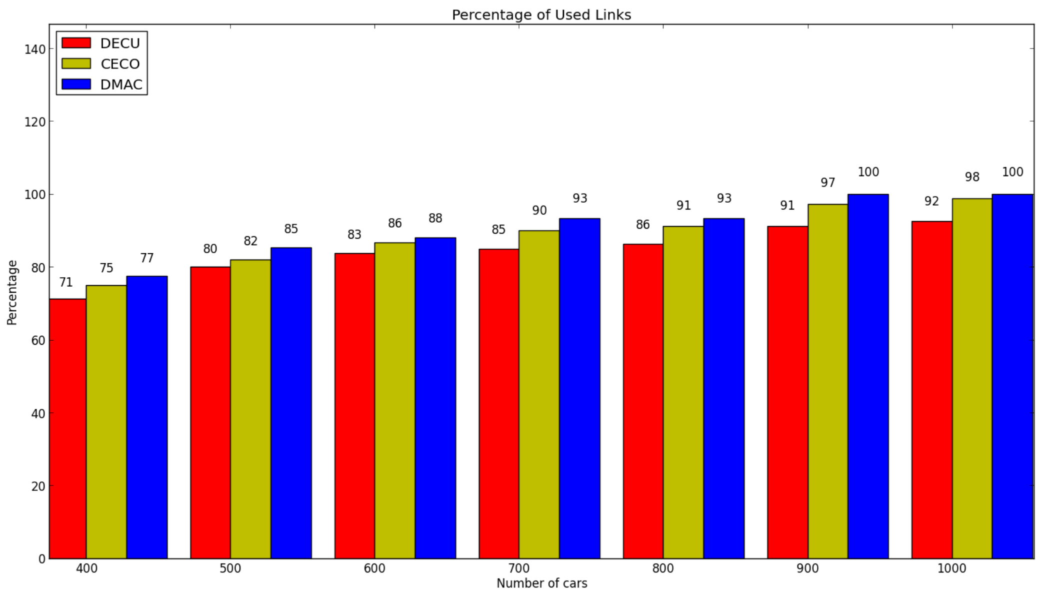

Figure 4.4: Percentage of Used Links

than DMAC planner ranging from 400 to 600 vehicle agents. However, the difference in average total travel time between these planners is not much. This can be explained as DMAC planner over-predicted the excess of number of vehicle agents on several links that made some vehicle agents rerouted to longer route.

4.4.2 Average Total Travel Distance

Figure 4.4 discussion: Figure 4.3 reports the average total travel distance of vehicle agents ranging from 400 to 1000 agents. As shown in figure 4.3, the distance that vehicle agent travelled with DMAC is longest in comparison to DECU (shortest) and CECO. Moreover, for 400-500 vehicle agents, the difference in average total travel distance between DECU and DMAC is not much in comparison to 700-1000 vehicle agents. This can be explained as when the traffic system was extremely congested, DMAC made vehicle agents to reroute more and therefore vehicle agent travelled longer than normal.

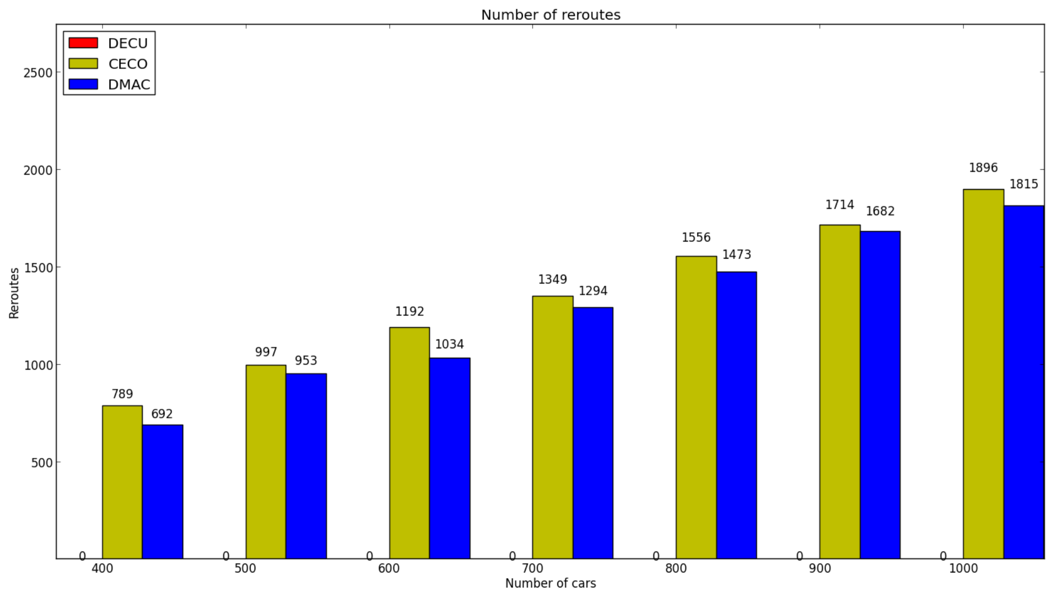

Figure 4.5: Number of Reroutes

4.4.3 Percentage of Used Links

4.4.4 Number of Total Reroutes

Figure 4.5 discussion: Figure 4.4 illustrates the percentage of used links within road network. The percentage of used links associated with DMAC is highest in comparison to DECU and CECO. For the extremely congested network with 900-100 vehicle agents, DMAC planner used all links of road network (100%). Having ability to predict the overcapacity of links, DMAC planner rerouted vehicle agents to another routes with less traffic in contrast to DECU planer. In other words, DMAC planner exploited links of traffic network better than DECU and CECO planners.

Figure 4.5 reports the number of reroutes made by all vehicle agents of traffic system. The number of reroutes made by vehicle agents with DECU planner is 0 as DECU planner does not allow vehicle agents to change routes. For DMAC planner, vehicle agents rerouted on average 1.73 (400), 1.91 (500), 1.72 (600), 1.85 (700), 1.84

(800), 1.87 (900) times in comparison to 1.97 (400), 1.99 (500), 1.98 (600), 1.93 (700), 1.95 (800), 1.9 (900), 1.89 (1000) times with CECO planner. The difference in number of reroutes between DMAC and CECO planners is relatively small as the quality of route plays important role in reducing the total travel time of overall system.

Chapter 5

Conclusion

In this chapter a summary of my thesis is presented and then the future work that I plan to carry out.

5.1 Summary

In this thesis I proposed the DDTA problem for optimising traffic system in terms of total travel time, total carbon emissions. Then DDTA problem is modelled as DynDCOP in order to solve it using DynDCOP solving algorithm. SBDO is used in combination with auctions to coordinate route plans of vehicle agents for solving DynDCOP of DDTA problem.

In this thesis, I also proposed IDTIS for vehicle agents to update current traffic conditions in a decentralised manner. For evaluating the efficiency of proposed coordination algorithm, I implemented three different planners (DECU, CECO, DMAC) and conducted the experiments for evaluating them with traffic simulator SUMO and TraCI. The experimental results shows that the performance of DMAC is relatively close to the performance of CECO.

5.2 Future Work

The first possible extension of the work described in this thesis is using learning algorithm for accurately predicting the expected travel times on future links.

The second extension would be using Dec-POMDP to model distributed traffic management problem. Dec-POMDP is able to handle uncertainty in expected travel time and capacity constraint discovery.

Finally, proposed DMAC algorithm would be run with large-scale network of hundred of thousands vehicle agents for simulating real city traffic such as Sydney CBD.

Appendix A

Program Code Listings

Listing A.1: SBDO_Vehicle_Agent class

```python
# -*- coding: iso-8859-1 -*-


import sbdo.sbdo
import sbdo.constraint
import sbdo.isgood
import constraints
import copy
from map import Map as Network
from link import Link
from node import Node
import sbdo_agent_route_planner
import datetime
from constants import *
import time
import itertools

class SBDO_Agent(sbdo.sbdo.Agent):
    def get_plan(self):
      count = 0
      while self.value is None:
        count += 1
        if count > 5:
          raise Exception('Timeout')
        self.handler.pass_message(self.name, self.name, SBDO_Agent.
            MESSAGE_NULL, None, 0)
        count += 1
        if count > MAX_PLANNING_TIME:
          raise RuntimeError('MAX_PLANNING_TIME exceeded')
        time.sleep(1)
      return self.value

class SBDO_Vehicle_Agent(SBDO_Agent):

  def __init__(self, handler, objective, name, road_network,
      start_node, end_node, planner, source):
    self.network = Network()
    self.network.nodes = road_network.nodes
    self.network.links = copy.copy(road_network.links)
    # roads that we have already considered and rejected
    self.blacklisted_links = set()
    self.blocked_paths = []
    # initially there are no constraints
    constraints = []
    # domain is defined by the road network
    domain = None
    # there is only one objective
    objectives = [objective]
    self.time = SIMULATION_START_TIME
    self.start_time = SIMULATION_START_TIME
    self.start_node = start_node
```

```
      self.end_node = end_node
      self.cur_node = self.start_node
      self.cur_link = None
      self.cur_eta = SIMULATION_START_TIME
      self.position = 0
      sbdo.sbdo.Agent.__init__(self, handler, objectives, constraints,
          name, domain)
      self.planner = planner
      self.source = source
      self.add_domain(None, None)

    def blacklist(self, link_id):
      if link_id in self.network.links and self.network.links.index(
          link_id) not in self.network.links:
        return False
      if link_id == self.cur_link or (link_id in self.network.links and
            self.cur_link.source == self.network.links[self.network.links
          .index(link_id)].source):
        return False
      new_blacklisted_links = copy.copy(self.blacklisted_links)
      new_blacklisted_links.add(link_id)
      new_network = Network()
      new_network.nodes = self.network.nodes
      new_network.links = copy.copy(self.network.links)
      try:
        del new_network.links[new_network.links.index(link_id)]
      except ValueError:
        print ("Warning, link %s has already been blacklisted for
            vehicle %s" %(link_id, self.name))
      router = sbdo_agent_route_planner.SBDO_Agent_Route_Planner(
          new_network, sbdo.isgood.Isgood(), self.planner, self.source)
          # thieu source gay ra loi
      route = router.AStar(self.cur_link, self.end_node, self.cur_eta, self.position)
      if route[0] is not None:
        self.blacklisted_links = new_blacklisted_links
        self.network = new_network
        self.handler.pass_message(self.name, self.name, SBDO_Agent.MESSAGE_NULL, None, 0)
        return True
      return False

    def add_neighbour(self, src, message):
      link_id = message[0]
      agent_id = message[1:]
      # if we already have a constraint for this link
        # add this agent to the constraint
        # add this agent as a neighbour
      # else
        # add a new constraint for this link
      found = False
      for constraint in self.constraints:
        # Relying on there only being one type of constraint
        if constraint.con.link_id == link_id: # change id to link_id
          found = True
          insert = True
          for agent in agent_id:
            self.neighbours.add(str(agent))
            for a in constraint.agents:
              if a == agent:
                insert = False
                break
            if insert:
              #constraint.agents.append(agent)
              constraint.agents = constraint.agents + (str(agent),)
          break
      if not found:
        for link in self.network.links:
          if link.id == link_id:
            link = link
            break
        if link is None:
          raise ValueError("link id %s does not exist in the traffic network" %(link_id))
        constraint = sbdo.constraint.Constraint(constraints.vehicle_capacity(link), self.name)
        for ID in agent_id:
          constraint.agents = constraint.agents + (str(ID),)
          self.neighbours.add(str(ID))
        self.constraints.append(constraint)

    def remove_neighbour(self, src, message):
      link_id = message[0]
      agent_id = message[1:]
      # remove this agent from the constraint
      # if it was the last agent
        # remove the constraint
      # maybe remove this agent from our neighbours
      for constraint in self.constraints:
        if constraint.con.link_id == link_id:
          try:
            if len(agent_id) > 0:
              for agent in agent_id:
                agents_list = list(constraint.agents)
                agents_list.remove(str(agent.ID))
                index = self.constraints.index(constraint)
                del self.constraints[index].agents
                self.constraints[index].agents = tuple(agents_list)
              if len(constraint.agents) == 1:
                self.constraints.remove(constraint)
            else:
              self.constraints.remove(constraint)
          except ValueError:
            pass
          break
      # rebuild the list of neighbours
      self.rebuild_neighbours()

    def add_domain(self, src, message):
      if message is not None:
        route = message[0]
        etas = message[1]
      else:
        route = None
        etas = None

      if self.cur_link is not None:
        del self.domain[:]
      domain_values = []
      if route is not None and etas is not None and self.cur_link is not None:
        shortest_route, shortest_etas = route, etas
      else:
        if self.support is None:
          router = sbdo_agent_route_planner.SBDO_Agent_Route_Planner(self.network, sbdo.isgood.Isgood(), self.planner, self.source)
        else:
          router = sbdo_agent_route_planner.SBDO_Agent_Route_Planner(self.network, self.recv[self.support], self.planner, self.source)
        # get a set of possible routes from original to destination
        # get first shortest route
        if self.cur_link is None:
          shortest_route, shortest_etas = router.AStar(self.cur_node, self.end_node, self.time, 0)
        else:
          shortest_route, shortest_etas = router.AStar(self.cur_link, self.end_node, self.time, self.position)
        if self.cur_link is not None and (len(shortest_route) == 0 or shortest_route[0] != self.cur_link):
          shortest_route.insert(0, self.cur_link)
          shortest_etas.insert(0, self.cur_eta)
      domain_values.append((shortest_route, shortest_etas))
      # remove each link of shortest route from network and calculate again shortest route without this link
      new_map = Network()
      new_map.nodes = self.network.nodes
      new_map.links = copy.copy(self.network.links)
      # Add possible routes from current position to destination locations to domain of vehicle agent
      tmp_route = []
      temp_route = copy.deepcopy(shortest_route)
      first_link = temp_route[0]
      del temp_route[0]
      del_set_links = []
      for i in range (1, len(temp_route)+1):
        gen_set_links = list(itertools.combinations(temp_route, i))
        del_set_links.extend(gen_set_links)
      for remove_tuple in del_set_links:
        for j in range(0, len(remove_tuple)):
          del new_map.links[new_map.links.index(remove_tuple[j])]
        if self.support is None:
          new_router = sbdo_agent_route_planner.SBDO_Agent_Route_Planner(new_map, sbdo.isgood.Isgood(), self.planner, self.source)
        else:
          new_router = sbdo_agent_route_planner.SBDO_Agent_Route_Planner(new_map, self.recv[self.support], self.planner, self.source)
        if self.cur_link is None:
          possible_route, possible_etas = new_router.AStar(self.cur_node, self.end_node, self.time, 0)
        else:
          possible_route, possible_etas = new_router.AStar(self.cur_link, self.end_node, self.time, self.position)
        if self.cur_link is not None and (len(possible_route) == 0 or possible_route[0] != self.cur_link):
          possible_route.insert(0, self.cur_link)
          possible_etas.insert(0, self.cur_eta)
        if possible_route != shortest_route and (possible_route, possible_etas) not in domain_values and possible_route[0] == first_link and (possible_route not in self.blocked_paths):
          domain_values.append((possible_route, possible_etas))
        elif possible_route == shortest_route:
          index_route = domain_values.index((shortest_route, shortest_etas))
          del domain_values[index_route]
          domain_values.append((possible_route, possible_etas))
        for j in range(0, len(remove_tuple)):
          new_map.links.append(remove_tuple[j])
      self.domain = domain_values
```

```

  def get_alternate_route(self, link_id, time):
    network = Network()
    network.nodes = self.network.nodes
    network.links = copy.copy(self.network.links)
    try:
      del network.links[network.links.index(link_id)]
    except ValueError:
      # link has already been deleted
      pass
    if self.support is None:
      router = sbdo_agent_route_planner.SBDO_Agent_Route_Planner(
          network, sbdo.isgood.Isgood(), self.planner, self.source)
    else:
      router = sbdo_agent_route_planner.SBDO_Agent_Route_Planner(
          network, self.recv[self.support], self.planner, self.source)
    route, etas = router.AStar(self.cur_link, self.end_node, time,
        self.position)
    if route is None:
      return None
    if self.cur_link is not None:
      route.insert(0, self.cur_link)
      etas.insert(0, self.cur_eta)
    return (route, etas)

  def stop(self):
    self.handler.pass_message(self.name, self.name, SBDO_Agent.
        MESSAGE_TERMINATE, 'DIE!', 0)
```

Listing A.2: Vehicle_Agent class

```
# -*- coding: iso-8859-1 -*-

# This class defines a smart vehicle agent.

import vehicle
import sbdo_agent
import sbdo
import sbdo_agent_route_planner
import datetime
import time as time_sys
from constants import *

class Vehicle_Agent(vehicle.Vehicle):
  def __init__(self, vehicle, planner, sbdo_message_handler, ID,
      objective, road_network):
    for att in vehicle.__dict__.iterkeys():
      self.__dict__[att] = vehicle.__dict__[att]
    self.planner = planner
    self.ID = ID
    self.sbdo_agent = sbdo_agent.SBDO_Vehicle_Agent(
        sbdo_message_handler, objective, ID, road_network, self.source
        , self.destination, self.planner, self.source)
    self.sbdo_agent.start()
    self.route = None
    self.time = SIMULATION_START_TIME
    self.objective = objective
    self.registered_auctions = []
    self.network = road_network

  def vehicle_registered(self, link_id, *vehicle_id):
    message = [link_id]
    message.extend(vehicle_id)
    self.sbdo_agent.handler.pass_message(str(self.ID), str(self.ID),
        self.sbdo_agent.MESSAGE_ADD_NEIGH, message, 0)
```

```

def vehicle_deregistered(self, link_id, *vehicle_id):
  message = [link_id]
  message.extend(vehicle_id)
  self.sbdo_agent.handler.pass_message(str(self.ID), str(self.ID),
      self.sbdo_agent.MESSAGE_REM_NEIGH, message, 0)

def auction_open(self, link_id):
  current_cost = self.objective.cost(self.route, self.
      estimatedTimesOfArrival)
  start_time = self.estimatedTimesOfArrival[0]
  alternate_route = self.sbdo_agent.get_alternate_route(link_id,
      start_time)
  if alternate_route is None:
    difference = 65545
  else:
    alternate_cost = self.objective.cost(*alternate_route)
    difference = alternate_cost - current_cost
  for link, time, auctioneer in self.registered_auctions:
    if link == link_id:
      break
  auctioneer.register_bid(self, difference)

def auction_won(self, link_id):
  for link, time, auctioneer in self.registered_auctions:
    if link == link_id:
      break
  auctioneer.deregister(self)

def auction_lost(self, link_id):
  auctioneer.deregister(self)
  self.sbdo_agent.blacklist(link_id)

  def time_tick(self, new_time):
    assert isinstance(new_time, datetime.datetime)
    self.sbdo_agent.time = new_time
    self.time = new_time
    self.sbdo_agent.position = self.position

    if self.route is None:
      self.sbdo_agent.cur_node = self.source
      self.sbdo_agent.cur_link = None
      self.sbdo_agent.cur_eta = self.time
    else:
      self.sbdo_agent.cur_link = self.route[self.routePosition]
      self.sbdo_agent.cur_node = self.sbdo_agent.cur_link.destination
      if self.routePosition + 1 < len(self.route):
        self.sbdo_agent.cur_eta = self.estimatedTimesOfArrival[self.routePosition + 1]

  def update_plan(self, auctions=True):
    try:
      print (self.ID, "update plan")
      new_route = self.sbdo_agent.get_plan()
      assert len(new_route[0]) > 0 or self.route[0][-1].destination == self.destination
      if self.route is None:
        self.routePosition = 0
        self.route, self.estimatedTimesOfArrival = new_route
        self.route_changed = True
      else:
        if new_route[0] != self.route:
          router = sbdo_agent_route_planner.SBDO_Agent_Route_Planner(self.network, 0, self.planner, self.source)
```

```
new_etas = router.construct_etas(new_route[0], self.time, self.position)
new_route = new_route[:-1]+(new_etas,)

if self.route[self.routePosition] in new_route[0]:
  if new_route[0].index(self.route[self.routePosition]) == 0:
    self.routePosition = 0
    self.sbdo_agent.blocked_paths.append(self.route)
    self.route, self.estimatedTimesOfArrival = new_route
    self.route_changed = True
    message = [new_route[0], new_route[1]]
    self.sbdo_agent.handler.pass_message(str(self.ID), str(self.ID), self.sbdo_agent.MESSAGE_ADD_DOMAIN, message, 0)
  else:
    positionNewRoute = new_route[0].index(self.route[self.routePosition])
    del new_route[0][:positionNewRoute] # remove traveled links
    del new_route[1][:positionNewRoute] # remove assoc. etas
    self.routePosition = 0 # set routePosition to 0
    self.sbdo_agent.blocked_paths.append(self.route)
    self.route, self.estimatedTimesOfArrival = new_route
    self.route_changed = True
    message = [new_route[0], new_route[1]]
    self.sbdo_agent.handler.pass_message(str(self.ID), str(self.ID), self.sbdo_agent.MESSAGE_ADD_DOMAIN, message, 0)
else:
  if self.route[self.routePosition].source == new_route
```

```
                 [0][0].source:
              self.routePosition = 0
              self.sbdo_agent.blocked_paths.append(self.route)
              self.route, self.estimatedTimesOfArrival = new_route
              self.route_changed = True
              message = [new_route[0], new_route[1]]
              self.sbdo_agent.handler.pass_message(str(self.ID), str(
                  self.ID), self.sbdo_agent.MESSAGE_ADD_DOMAIN,
                  message, 0)
      print ("finish update plan")
      if auctions:
        return self.update_auctions()
      print (self.ID, "finish update plan + update auction")
    except AssertionError:
      print ('WARNING: Tried to change to a non-contiguious route')
      print ('self.route =', self.route)
      print ('self.routePosition =', self.routePosition)
      print ('new_route =', new_route[0])
    return True

  def update_auctions(self):
    if self.route is None:
      print ("warning, vehicle_agent.update_auctions, no route")
      return False
    # check to see if we should deregister from any auctions
    if not not self.registered_auctions:
      for link_id, time, auctioneer in self.registered_auctions:
        found = False
        for i in xrange(len(self.route)):
          if self.route[i] == link_id and self.
              estimatedTimesOfArrival[i] >= time and self.
              estimatedTimesOfArrival[i] < time + AUCTION_BLOCK_TIME:
```

```
            found = True
            break
        if not found:
          auctioneer.deregister(self)

    # check to see if we should register for any new auctions
    for i in xrange(len(self.route)):
    #for r_link_id, r_time in self.route:
      found = False
      for link_id, time, auctioneer in self.registered_auctions:
        print len(auctioneer.registered_vehicles)
        if self.route[i] == link_id and self.estimatedTimesOfArrival[i] >= time and self.estimatedTimesOfArrival[i] < time + AUCTION_BLOCK_TIME:
          found = True
          break
      if not found:
        if self.time > (self.estimatedTimesOfArrival[i] - AUCTION_CLOSE_TIME):
          auctioneer = self.planner.get_auctioneer(self.route[i], self.estimatedTimesOfArrival[i])

          reserve = auctioneer.request_reserve()
          if not reserve:
            self.registered_auctions.append((self.route[i], self.estimatedTimesOfArrival[i], auctioneer))
          else:
            # remember that we have reserved a spot on this link
            self.registered_auctions.append((self.route[i], self.estimatedTimesOfArrival[i], auctioneer))
        elif self.time >= self.estimatedTimesOfArrival[i] - AUCTION_LEAD_TIME and self.time < self.
```

```
            estimatedTimesOfArrival[i] - AUCTION_CLOSE_TIME:
          auctioneer = self.planner.get_auctioneer(self.route[i], self.estimatedTimesOfArrival[i])
          current_cost = self.objective.cost(self.route, self.estimatedTimesOfArrival)
          start_time = self.estimatedTimesOfArrival[0]
          alternate_route = self.sbdo_agent.get_alternate_route(self.route[i], start_time)
          if alternate_route is None:
            difference = 65545
          else:
            alternate_cost = self.objective.cost(*alternate_route)
            difference = alternate_cost - current_cost
          auctioneer.register_bid(self, difference)
        elif self.time > (self.estimatedTimesOfArrival[i] - AUCTION_REGISTER_TIME):
          auctioneer = self.planner.get_auctioneer(self.route[i], self.estimatedTimesOfArrival[i])
          auctioneer.register(self)
          self.registered_auctions.append((self.route[i], self.estimatedTimesOfArrival[i], auctioneer))
    # delete obsolete received proposal
    message = []
    self.sbdo_agent.handler.pass_message(str(self.ID), str(self.ID), self.sbdo_agent.MESSAGE_REM_OBS_PROPL, message, 0)
    return True

  def __eq__(self, other):
    if type(other) != type(self):
      return False
    if self.ID != other.ID:
      return False
    return True

  def __hash__(self):
    return self.ID

  def get_plan(self):
    return self.sbdo_agent.get_plan()

  def start(self):
    self.sbdo_agent.start()

  def stop(self):
    self.sbdo_agent.stop()

  def __getstate__(self):
    return {'ID':self.ID}
```

Listing A.3: Auctioneer agent class

```
# -*- coding: iso-8859-1 -*-
import vehicle_agent
from constants import *
import math

class Auctioneer:
  def __init__(self, link, time, planner):
    self.registered_vehicles = []
    # link for bidding to traverse
    self.link = link
    # time for which privileges are already allocated
    self.time = time
    # simulator's time
    self.simulator_time = None
    # lists for storing bidders, winners, losers of auction
    self.bidders = []
    self.reserve_capacity = 0
    self.auction_open = False
    self.auction_done = False
    self.planner = planner

  def register(self, agent):
    # check to see if the agent is already registered
    for a in self.registered_vehicles:
      if a == agent:
        return
    if len(self.registered_vehicles) != 0:
      id_list = []
      for reg_vehicle in self.registered_vehicles:
        id_list.append(reg_vehicle.ID)
      agent.vehicle_registered(self.link.id, *id_list)

      for a in self.registered_vehicles:
        a.vehicle_registered(self.link.id, agent.ID)
      # add this vehicle to the list
    self.registered_vehicles.append(agent)

  def deregister(self, agent):
    agent_list = []
    found = False
    for a in self.registered_vehicles:
      if a == agent:
        self.registered_vehicles.remove(a)
        found = True
        break
    if found:
```

```python
      agent.vehicle_deregistered(self.link.id, *self.registered_vehicles)
      for a in self.registered_vehicles:
        a.vehicle_deregistered(self.link.id, agent)

  def register_bid(self, agent, bid):
    # NOTE: assuming each vehicle only makes one bid
    self.bidders.append((bid, agent))

  def conduct_auction(self):
    self.bidders.sort()
    self.bidders.reverse()
    losers = 0
    len_winners = len(self.bidders)
    if len_winners <= self.link.capacity:
      for i in xrange(0, len_winners):
        self.bidders[i][1].auction_won(self.link.id)
      self.reserve_capacity = self.link.capacity - len_winners
    else:
      for i in xrange(0, self.link.capacity):
        self.bidders[i][1].auction_won(self.link.id)
      for i in xrange(self.link.capacity, len(self.bidders)):
        self.bidders[i][1].auction_lost(self.link.id)
        losers += 1
    self.auction_done = True
    return losers

  def request_reserve(self):
    # Check if it's possible to give privilege to vehicle agent if auction is already closed
    if self.reserve_capacity > 0:
      self.reserve_capacity -= 1
      return True
    else:
      return False

  def time_tick(self, time):
    self.simulator_time = time
    if self.simulator_time >= self.time - AUCTION_LEAD_TIME and not self.auction_open:
      self.open_auction()
    if self.simulator_time >= self.time - AUCTION_CLOSE_TIME and not self.auction_done:
      self.conduct_auction()

  def open_auction(self):
    for agent in self.registered_vehicles:
      agent.auction_open(self.link.id)
    self.auction_open = True
```

Listing A.4: SUMO simulator, TraCI and traffic planner

```python
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import sys
sys.path.append("TrafficPlanner/")
from xml.dom import minidom
from map import Map
from node import Node
from link import Link
from vehicle import Vehicle
from planner import Planner
from aStarPlanner import AStarPlanner
from centralisedTrafficPlanner import CentralisedTrafficPlanner
from decentralisedTrafficPlanner import DecentralisedTrafficPlanner
from timeEstimatingPlanner import TimeEstimatingPlanner
from sbdo_vehicle_planner import SBDO_Vehicle_Planner
from sbdo_link_planner import SBDO_Link_Planner
from constants import *
import random
import time
import subprocess
import traci
import traci.constants as tc
import os
os.environ['XERCESC_NLS_HOME'] = '/usr/share/xerces-c/msg'

def main():
  activeCars = []
  completedCars = []
  map = Map()
  #Variables for program run
  finished = False
  cur_time = 0
  mapFN = "map.xml"
  vehiclesFN = "vehicles.xml"

  if len(sys.argv) > 1:
    mapFN = sys.argv[1]
  if len(sys.argv) > 2:
    vehiclesFN = sys.argv[2]
  map = ParseMapFile(mapFN)
  print ("Map loaded")
  activeCars = ParseVehiclesFile(vehiclesFN, map)
  print ("Vehicles loaded")
  if len(sys.argv) > 3:
```

```
  if sys.argv[3].lower() == "ctp":
    routePlanner = CentralisedTrafficPlanner(map)
  elif sys.argv[3].lower() == "dctp":
    routePlanner = DecentralisedTrafficPlanner(map)
  elif sys.argv[3].lower() == "tetp":
    routePlanner = TimeEstimatingPlanner(map)
  elif sys.argv[3] == 'svtp':
    routePlanner = SBDO_Vehicle_Planner(map)
  elif sys.argv[3] == 'sltp':
    routePlanner = SBDO_Link_Planner(map)
  elif sys.argv[3] == 'astar':
    routePlanner = AStarPlanner(map)
else:
  routePlanner = AStarPlanner(map)
# ensure sumo has the same map we have
try:
  os.unlink(mapFN + '.xml')
except OSError:
  pass
result = subprocess.call((NETCONVERT, '--osm-files', mapFN, '-o', mapFN + '.xml', '--tls.join', '--remove-edges.by-vclass', 'rail_slow,rail_fast,bicycle,pedestrian', '--proj.utm', '--junctions.join' ))
if result != 0:
  print ("Error: netconvert failed, aborting")
routePlanner.Setup (map, activeCars)
routePlanner.InitialPlanning(map, activeCars)
print ("Initial Planning Completed")
write_sumo_vehicles(activeCars, vehiclesFN)
# start sumo
sumo = subprocess.Popen((SUMO, '--net-file', mapFN + '.xml', '--remote-port', '32000', '--route-files', vehiclesFN + '.xml', '--
    summary-output', os.path.join(OUTPUT_DIR, 'summary'), '--
    tripinfo-output', os.path.join(OUTPUT_DIR, 'tripinfo'), '--
    vehroute-output', os.path.join(OUTPUT_DIR, 'vehroute'), '--
    vehroute-output.exit-times', 'true' ))
# setup traci
traci.init(32000)
print ("Starting")
all_cars = activeCars
activeCars = {}
started = False
num_reroutes = 0
  #This is to record the load balancing
load_balance = []
absolute_balance = float(len(all_cars))/float(len(map.links))
while not finished and sumo.poll() is None:
  #The load balance for this step
  this_balance = []
  # sumo simulation step
  traci.simulationStep(0)
  results = traci.simulation.getDepartedIDList()
  if len(results) > 0:
    started = True
    for car_id in results:
      for car in all_cars:
        if str(car.id) == car_id:
          activeCars[car_id] = car
          current_pos = traci.vehicle.getRoadID(str(car.id))
          car.routePosition = car.route.index(int(current_pos))
          car.position = traci.vehicle.getLanePosition(str(car.id))
          if car.routePosition == -1:
            car.route = [current_pos]
            car.routePosition = 0
  results = traci.simulation.getArrivedIDList()
  #print 'DEBUG: arrived =', results
  for car_id in results:
    if car_id in activeCars:
      del activeCars[car_id]
  if cur_time % SIMULATION_UPDATE_INTERVAL == 0:
    for car in activeCars.values():
    #for car in all_cars:
      try:
        current_pos = traci.vehicle.getRoadID(str(car.id))
        car.routePosition = car.route.index(int(current_pos))
        car.position = traci.vehicle.getLanePosition(str(car.id))
        assert car.routePosition != -1
      except ValueError:
        # car is currently on an intersection, mark the car as being on the previous road
        try:
          node = int(current_pos.split('_')[0][1:])
        except ValueError:
          # car is probably teleporting
          continue
        for n in map.nodes:
          if n.id == node:
            for edge in n.incomingLinks:
              if edge in car.route:
                car.routePosition = car.route.index(edge)
                car.position = 0
                break
  if cur_time % SIMULATION_PLANNING_INTERVAL == 0:
    print ("planning")
    routePlanner.Plan(map, activeCars.values(), cur_time)
    for car in activeCars.values():
      if car.route_changed:
        # tell sumo about the new route
        car.route_changed = False
        new_route = [str(edge.id) for edge in car.route]
        curr_route = traci.vehicle.getRoute(str(car.id))
        if new_route != curr_route:
          print ("car = {0}, current route = {1}, current position = {2}, new route = {3}".format(car.id, curr_route, traci.vehicle.getRoadID(str(car.id)), new_route))
          traci.vehicle.setRoute(str(car.id), new_route[car.routePosition:])
          num_reroutes += 1
  if len(activeCars) == 0 and started:
    finished = True
  #Record the load balancing for this step
  for link in map.links:
    link.occupants = []
  for car in activeCars.values():
    try:
      current_pos = int(traci.vehicle.getRoadID(str(car.id)))
    except ValueError:
      # car is currently on an intersection, or teleporting
      # it doesn't count for any of the links
      break
    for link in map.links:
      if link.id == current_pos:
        link.occupants.append(car)
  for link in  map.links:
    this_balance.append(len(link.occupants))
  load_balance.append(this_balance)
  # get the list of active vehicles
  if cur_time % SIMULATION_STATISTICS_INTERVAL == 0:
          print ("At time:", cur_time, " amount of active cars:", len(
              activeCars))
      cur_time += 1
  traci.close()
  count = 0
  while sumo.poll() is None:
      time.sleep(1)
      if count > 10:
          sumo.terminate()
  print ("Percentage of links used: ", (float(roads_used.count(1))/
      float(len(roads_used))) * 100.0)
  print ('Reroutes:', num_reroutes)
  print ('MainThread finished, Safe to kill any child threads')

def write_sumo_vehicles(vehicles, vehiclesFN):
  # have to sort the vehicles by departure cur_time
  vehicles.sort(key=lambda v: v.startTime)
  fd = open(vehiclesFN + '.xml', 'w')
  fd.write('<routes>\n')
  fd.write('    <vType id="vtype1" length="7.5" maxSpeed="70" minGap
      ="2.5" vClass="passenger" guiShape="passenger/sedan">\n')
  fd.write('        <carFollowing-Krauss accel="0.8" decel="4.5"
      sigma="0.5" />\n')
  fd.write('    </vType>\n')
  for veh in vehicles:
      fd.write('    <vehicle id="{0}" type="vtype1" depart="{1}"
          departPos="free" departSpeed="0">\n'.format(veh.id, veh.
          startTime))
      fd.write('        <route edges="')
      for edge in veh.route:
          fd.write(str(edge.id))
          fd.write(' ')
      fd.write('"  />\n')
      fd.write('    </vehicle>\n')
  fd.write('</routes>\n')
  fd.close()

if __name__ == "__main__":
  main()
```

Bibliography

[1] Sheng-hai An, Byung-Hyug Lee, and Dong-Ryeol Shin. A survey of intelligent transportation systems. In *Computational Intelligence, Communication Systems and Networks (CICSyN), 2011 Third International Conference on*, pages 332–337. IEEE, 2011.

[2] Richard Arnott and Kenneth Small. The economics of traffic congestion. *American Scientist*, 82(5):446–455, 1994.

[3] World Road Association. Its handbook. `http://road-network-operations.piarc.org/`. Accessed: 12/02/2013.

[4] Ana L.C. Bazzan, M. de Brito do Amarante, and F.B. Da Costa. Management of demand and routing in autonomous personal transportation. *Journal of Intelligent Transportation Systems*, 16(1):1–11, 2012.

[5] Ana L.C. Bazzan and Franziska Klugl. *Multi-Agent Systems for Traffic and Transportation Engineering.* IGI Global, 2009.

[6] Martin Beckmann, CB McGuire, and Christopher B Winsten. Studies in the economics of transportation. Technical report, 1956.

[7] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. Sumo - simulation of urban mobility: An overview. In *SIMUL 2011, The Third International Conference on Advances in System Simulation*, Barcelona, Spain, 2011.

[8] G. Billiau, C. Chang, and A. Ghose. Sbdo: A new robust approach to dynamic distributed constraint optimisation. *Principles and Practice of Multi-Agent Systems*, pages 11–26, 2012.

[9] Dietrich Braess, Anna Nagurney, and Tina Wakolbinger. On a paradox of traffic planning. *Transportation science*, 39(4):446–450, 2005.

[10] Ennio Cascetta. *Transportation systems engineering: theory and methods*, volume 49. Springer, 2001.

[11] Bo Chen and Harry H Cheng. A review of the applications of agent technology in traffic and transportation systems. *Intelligent Transportation Systems, IEEE Transactions on*, 11(2):485–497, 2010.

[12] Yi-Chang Chiu, Jon Bottom, Michael Mahut, Alex Paz, Ramachandran Balakrishna, Travis Waller, and Jim Hicks. Dynamic traffic assignment: A primer. *Transportation Research E-Circular*, (E-C153), 2011.

[13] Fabian A. Chudak, Vania Dos Santos Eleuterio, and Yurii Nesterov. Static traffic assignment problem: A comparison between beckmann (1956) and nesterov and de palma (1998) models. In *Proceedings of 7th Swiss Transport Research Conference*, Monte-Verita, Ascona, Switzerland, 2007.

[14] Denise de Oliveira, Ana L.C. Bazzan, and Victor Lesser. Using cooperative mediation to coordinate traffic lights: a case study. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 463–470. ACM, 2005.

[15] Prajakta Desai, Seng W Loke, Aniruddha Desai, and Jugdutt Singh. Multi-agent based vehicular congestion management. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 1031–1036. IEEE, 2011.

[16] David Easley and Jon Kleinberg. *Networks, crowds, and markets.* Cambridge Univ Press, 2010.

[17] Lino Figueiredo, Isabel Jesus, JA Tenreiro Machado, Jose Rui Ferreira, and JL Martins de Carvalho. Towards the development of intelligent transportation systems. In *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pages 1206–1211. IEEE, 2001.

[18] Michael Florian, Michael Mahut, and Nicolas Tremblay. A hybrid optimization-mesoscopic simulation dynamic traffic assignment model. In *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pages 118–121. IEEE, 2001.

[19] Google. Google maps hompage. `http://maps.google.com/`. Accessed: 10/01/2013.

[20] Cristian Gratie and Adina Magda Florea. Alleviating urban traffic congestion by means of adaptive routing. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2009 11th International Symposium on*, pages 361–367. IEEE, 2009.

[21] P. Harvey, C. Chang, and A. Ghose. Support-based distributed search: a new approach for multiagent constraint processing. *Argumentation in Multi-Agent Systems*, pages 91–106, 2007.

[22] Bruce N Janson. Dynamic traffic assignment for urban road networks. *Transportation Research Part B: Methodological*, 25(2):143–161, 1991.

[23] Robert Junges and Ana L.C. Bazzan. Evaluating the performance of dcop algorithms in a real world, dynamic problem. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages

599–606. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[24] Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *STACS 99*, pages 404–413. Springer, 1999.

[25] Terrence Lee. Cooperative travel planning in city networks. University of Wollongong, 2011. Honours Thesis.

[26] Phil Magney. Real-time traffic service revenue to boom over the next five years. `http://www.isuppli.com`, June 2009. Accessed: 10/01/2013.

[27] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *International Conference on Autonomous Agents: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-*, volume 1, pages 438–445, 2004.

[28] P.J. Modi, W.M. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1):149–180, 2005.

[29] Yurii Nesterov and Andre De Palma. Stationary dynamic solutions in congested transportation networks: summary and perspectives. *Networks and spatial economics*, 3(3):371–395, 2003.

[30] Global Health Observatory World Health Orgnaization. Urban population growth. `http://www.who.int/gho/urban_health/situation_trends/urban_population_growth_text/en/index.html`. Accessed: 10/02/2013.

[31] Brammert Ottens and Boi Faltings. Coordinating agent plans through distributed constraint optimization. In *Proceedings of the ICAPS-08 Workshop on Multiagent Planning*, 2008.

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

[33] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *International Joint Conference on Artificial Intelligence*, volume 19, page 266, 2005.

[34] Ramesh Thangarajoo and Hoong Chuin Lau. Distributed route planning and scheduling via hybrid conflict resolution. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 02*, WI-IAT '10, pages 374–378, Washington, DC, USA, 2010. IEEE Computer Society.

[35] TomTom. Tomtom hd traffic. `http://www.tomtom.com/en_gb/services/live/hd-traffic`. Accessed: 10/01/2013.

[36] JG Wardrop. Some theoretical aspects of road traffic research. 1952.

[37] Axel Wegener, MichałPiórkowski, Maxim Raya, Horst Hellbrück, Stefan Fischer, and Jean-Pierre Hubaux. Traci: an interface for coupling road traffic and network simulators. In *Proceedings of the 11th communications and networking simulation symposium*, CNS '08, pages 155–163, New York, NY, USA, 2008. ACM.

[38] Tomohisa Yamashita and Koichi Kurumatani. New approach to smooth traffic flow with route information sharing. *Multi-Agent Systems for Traffic and Transportation. IGI Global*, 2009.

[39] Xu Yang and Wilfred W. Recker. Modeling dynamic vehicle navigation in a self-organizing, peer-to-peer, distributed traffic information system. *Journal of Intelligent Transportation Systems*, 10(4):185–204, 2006.