City-Scale Traffic Simulation

-- Performance and Calibration

Yan Xu

(*B. Eng., HIT, China*)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
DEPARTMENT OF COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SINGAPORE

2014

SUMMARY

Road congestions in a city-scale (or urban) traffic system are largely determined by the equilibrium between the demand (people's requirement for travel) and the supply (the capacity of the traffic system). In general, there are two types of solutions to manage road congestions in a city-scale traffic system: transport planning and traffic control schemes. Due to the high complexity in a city-scale traffic system and the limited modelling capability of mathematical models, city-scale traffic simulation turns out to be an appealing toolkit to evaluate the holistic impact of transport planning and traffic control schemes to the entire city-scale traffic system. However, a successful deployment and maintenance of a city-scale traffic simulation is not trivial, and limits the feasibility of city-scale traffic simulations in real-world traffic systems. The thesis investigates and then proposes solutions on two challenges in the process of deploying and maintaining of a city-scale traffic simulation: the performance optimization of a city-scale traffic simulation and the calibration algorithm to estimate variables (e.g. model parameters and model inputs) in a city-scale traffic simulation.

The key problem in the performance optimization of city-scale traffic simulations is the lack of a systematic methodology to optimize the performance of city-scale traffic simulations. This thesis proposes a three-step methodology to improve the computational complexity and scalability of city-scale traffic simulations. These steps are framework optimization, serial bottleneck optimization, and scalability optimization. Following the three-step methodology, this thesis illustrates: 1) an Entry Time based Supply Framework (ETSF), which optimizes the performance and the computational complexity to simulate congested traffic scenarios; 2) an efficient two-dimensional spatial index (Sim-Tree), which reduces the time cost of a serial bottleneck in the spatial index and improves the scalability of parallel traffic simulations; and 3) a framework to execute city-scale traffic simulations on the CPU/GPU Platform. The three-step methodology is demonstrated to support the simulation of the Singapore expressway network from 7:00AM to 8:00AM with a total 106,386 vehicles. The total execution time is improved from 6690.2ms to 894.0ms. The three-step performance optimization methodology is suitable to be used as

a guideline to optimize the performance of both existing and ongoing city-scale traffic simulations.

The key problem in the calibration of city-scale traffic simulations is the lack of an effective algorithm to calibrate a large number of variables in demand models and supply models in city-scale traffic simulations (e.g. the OD Matrix). As the traffic road network size grows, we found that the state-of-the-art calibration algorithm (SPSA) deteriorates. The reason lies in the systematic error to incorporate uncorrelated measurements in the method of estimating gradients of calibration variables. Motivated by this, we propose W-SPSA ('W' means Weighted). The key idea of W-SPSA is to incorporate a 2-D weight matrix in the calibration algorithm, to assist the estimation of the gradient. The 2-D weight matrix represents correlations between variables and measurements. The idea is successfully demonstrated to calibrate 373,646 time-dependent OD flows in one day on Singapore Expressway Network. In the big research framework of traffic simulation calibration, the idea of weighted gradient approximation provides an effective methodology to calibrate variables in city-scale traffic simulations.

In summary, this thesis proposes frameworks, algorithms and data structures in computer science, to solve problems (execution performance and calibration) involved in the development and maintenance of city-scale traffic simulations.

DECLARATION

I hereby declare that the thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

_____

2014

ACKNOWLEDGEMENT

There are lots of people whom I would like to thank for a variety of reasons. I sincerely acknowledge all those whom I mention, and apologise to anybody who I might have forgotten.

Firstly, thanks to my supervisor A/P Gary Tan, for the guidance, support and friendship, to bring me up from an undergraduate to a computer science researcher. What impressed me most is that he never gives up on his students. Secondly, thanks to my parents, my wife and my sister, for supporting me during the last 5 years. I love them very much. Thirdly, thanks to A/P Constantinos Antoniou. "Do what I believe", that is a phrase I learned from him and I will always remember it.

Thanks to the guidance from Prof Li-Shiuan Peh and Prof Moshe E. Ben-Akiva. It is a great honor to work with them. Thanks to Prof Lee Der-Horng and A. Prof Chin Hoong Chor. Their modules led me from computing science to the transportation field.

Thanks to my wonderful friends (there is no order). Li Lu, Zhang Fan, Wang Pidong; Seth Hetu, Bogdan Marius Tudor, Carbunaru Cristina, Vinh An Vu, Le Duy Khanh, Saeid Montazeri, Linh Luong Ba; Lu Lu, Song Xiao, Melani Jayasuriya, Weng Zhiyong, Saber Hamishagi Vahid, Harish Loganathan, Randy Tandriansyah, Yao Jin, Wang Dong, Lu Yang, Zhang Huai Peng; Zhang Wenjia, Wang Bin and many friends in SMART. You made my PhD life easier and more enjoyable.

Lastly but not the least, thanks to the great Project Managers: Kakali Basak, Stephen Robinson and Francisco Pereira.

RELATED PUBLICATIONS

Journal Publications:

1. **Yan Xu**, Xiao Song, Zhiyong Weng and Gary Tan, "An Entry Time based Supply Framework (ETSF) for Mesoscopic Traffic Simulations", Journal: Simulation Modelling Practice and Theory, 2014.
2. Lu Lu, **Yan Xu**, Constantinos Antoniou and Moshe Ben-Akiva, "W-SPSA: An Enhanced SPSA Algorithm for the Calibration of Dynamic Traffic Assignment Models", Journal: Transportation Research Part C: Emerging Technologies, 2013.

Conference Publications:

1. **Yan Xu** and Gary Tan, "Sim-Tree: Indexing Moving Objects in Large-Scale Parallel Microscopic Traffic Simulation", ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS), 2014.
2. **Yan Xu**, Gary Tan, Xiaosong Li and Xiao Song, "Mesoscopic Traffic Simulation on CPU/GPU", ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS), 2014.
3. Kakali Basak, Seth Hetu, Zhemin Li, Carlos M. Lima Azevedo, Harish Loganathan, Tomer Toledo, Runmin Xu, **Yan Xu**, Li-Shiuan Peh, Moshe Ben-Akiva, "Modeling Reaction Time within a Traffic Simulation Model", In Proceedings of the 16th International IEEE Annual Conference on Intelligent Transportation Systems (ITSC), 2013 (Best paper award nominee).
4. **Yan Xu** and Gary Tan, "An Offline Road Network Partitioning Solution in Distributed Transportation Simulation", IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT), 2012.
5. **Yan Xu** and Gary Tan, "hMETIS-based Offline Road Network Partitioning", Asia Simulation Conference, 2012.

Poster and Short Papers:

1. **Yan Xu** and Gary Tan, "Workload Estimation Algorithms in Parallel Traffic Simulation", International Conference on Parallel and Distributed Systems (ICPADS), 2013, accepted as a poster paper.
2. **Yan Xu** and Gary Tan, "Offline Road Network Partitioning in Distributed Transportation Simulation", ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS), 2012, accepted as a short paper.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

1.1 The Need for City-scale Traffic Simulations

Road congestions in a city-scale (or urban) traffic system are largely determined by the equilibrium between the demand (people's requirement for travel) and the supply (the capacity of the traffic system). In general, there are two types of solutions to manage road congestions in a city-scale traffic system: transport planning and traffic control schemes. Transport planning deals with the long-term (e.g. 10 years) solutions (e.g. by building new highways and bridges), while traffic control schemes look for short-term solutions (e.g. by using adaptive signal control). However, the difficulty is how to evaluate the holistic impact of transport planning and traffic control schemes to the entire city-scale traffic system.

In general, mathematical models (e.g. link performance functions) cannot adequately capture the impact of transport planning and traffic control schemes, because of the high complexity in a city-scale traffic system. Thus, city-scale traffic simulation turns out to be an appealing toolkit to evaluate the holistic impact of transport planning and traffic control schemes to the entire city-scale traffic system with high accuracy, high confidence and visual demonstration. City-scale traffic simulation is useful in the following scenarios:

1) Evaluating the impact of existing or newly proposed road infrastructures, traffic control technologies, policies and events to the entire city-scale traffic system.
2) Real-time analysis and response assistance of special events (e.g. incidents).
3) Extracting surrounding traffic when working on a piece of a city-scale traffic system.
4) Visual demonstrations and studies of a city-scale traffic system.

City-scale traffic simulations have been receiving industrial attention in the last 10 years. First, in 2008, researchers in Minnesota (Hourdos & Michalopoulos, 2008) conducted a series of interviews with transport-related departments and firms about their need for a city-scale traffic

simulation platform. The interview found strong interests in a city-scale traffic simulation platform from engineers in the regional transportation management center and transportation consultants. Second, Aimsun deployed pioneer city-scale traffic simulation based traffic control systems in Singapore (in 2006) and Madrid (in 2008) for real-time decision making in support of the regional traffic management system. Caliper recently developed a city-scale traffic simulation model for the Maricopa Association of Governments (TransModeler, 2014).

There are several reasons that pushed the progress of city-scale traffic simulations. (1) City-scale traffic systems are becoming more and more complex with the emergence and development of traffic information systems, incident management systems, adaptive signal control systems and others. Mathematical models have limited capability to formulate the details of individual system and the complex interactions among these systems. It makes city-scale traffic simulation more attractive. (2) Researchers in traffic modeling are continuously pushing the capability of modeling people's decision making in route choice behaviors (Prato, 2009), parking behaviors (Boyles et al., 2014), response to traffic information (Ben-Elia & Shiftan, 2010), etc. It makes it more and more practical to simulate complex traffic scenarios in a city-scale traffic simulation platform. (3) The number of cores in the CPU and the GPU in one machine is increasing fast in the last 10 years while the prices of machines are stable. It makes the computational resources, which is required to support city-scale traffic simulation, much affordable than before. (4) New technologies to collect traffic data. Two pioneer technologies are camera-based traffic collection technology and smart-phone based traffic data collection technology. Combining these new technologies and traditional ways (e.g. loop detectors) increases the accuracy and the coverage of traffic data collection, making it possible to calibrate and evaluate city-scale traffic simulations.

The general work flow to deploy a city-scale traffic simulation system is shown in Figure 1. The first step is to define problems and objectives. As explained previously, problems can be to evaluate an innovative traffic control technology, to investigate the impact of a new train station, or to offer real-time operation suggestions to congestion management and incident response. Traffic simulators are not designed to replicate every detail in real-world traffic systems. In contrast, traffic simulators are designed to capture the most critical components which are important for answering the target problem. Thus, a clear definition of problem bounds the scope

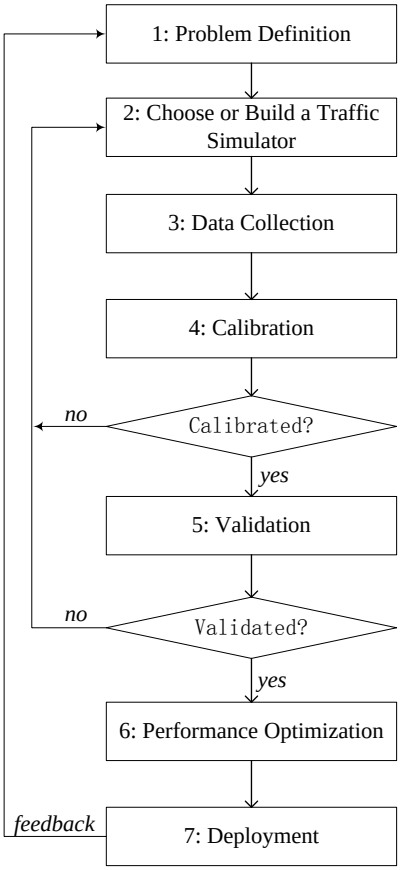


Figure 1: The work flow to deploy a city-scale traffic simulation

of a traffic simulator. The second step is to choose an existing traffic simulator or to build a new traffic simulator. There are a number of industrial and academic traffic simulators, e.g. AIMSUN (Barcelo, 2002), Commuter (Commuter, 2014), DynaMIT (Ben-Akiva et al., 1998), MITSIMLab (Yang et al., 2000), PARAMICS (Cameron et al., 1995), VISSIM (Fellendorf and Vortisch, 2014), etc. The classification of traffic simulators is introduced in Section 2.2. The third step is data collection, including the road network, traffic surveillance data, origin-destination (OD) matrix, drivers' characters and problem specified parameters, like road pricing setting and signal control setting. A traffic database, with a high accuracy, a high internal consistence, and a full spatial and temporal coverage, is vital to the success of deploying a city-scale traffic simulation system but also expensive and in many cases impossible. A data consistency checking method is introduced in Section 6.5.2. After that, variables (including model parameters and model inputs) are calibrated, in order that the traffic simulator has an acceptable accuracy to replicate the real-world traffic system. Then, the traffic simulator is validated, in order that the traffic simulator

has the capability to solve the target problem. The sixth step is performance optimization. Performance, which is the execution time of simulating the target traffic scenario, is also an important factor to choose a traffic simulator. Finally, the traffic simulator is deployed to solve the real-world problem. Note that the process from problem definition to deployment is not one-off, but that the deployed model provides feedback to the problem definition, so that the process can be refined and improved.

1.2 The Challenges and Thesis Scope

Even though the concept of a city-scale traffic simulation has been demonstrated in the real-world traffic system (in Singapore and Madrid), there is still a large research area to be investigated and fulfilled. A trend for city-scale traffic simulation is to enhance the functional capability and execution performance of traffic simulators, in order to lower the barrier and make it easier for end-users (e.g. government agencies, consultants and academic researchers) to develop and maintain a city-scale traffic simulation platform.

Currently, there are a number of challenges in the process of deploying and maintaining a city-scale traffic simulation:

1) The high modeling complexity in simulating a city-scale traffic system
2) An accurate, consistent and sufficient traffic database
3) The traffic simulator's execution performance
4) A calibration method to estimate variables in a city-scale traffic simulation

In this thesis, the traffic models and the traffic database are assumed given. Then, the thesis focuses on the following two challenges: performance and calibration. To be exact, the thesis is to propose frameworks, data structures and algorithms in computer science, to solve problems related to the performance optimization and the calibration of a city-scale traffic simulator.

In this thesis, the performance of a traffic simulator means the execution time of simulating a traffic scenario. Performance optimization means to optimize the underlying frameworks, data structures and algorithms in a traffic simulator to reduce the execution time. The execution performance is more important for simulating a city-scale traffic scenario than simulating a small

area traffic scenario. First, as the simulated trips increase and the simulated road network grows, the required execution time of simulating a traffic scenario increases rapidly. Second, the level of uncertainty (e.g. the randomness in the route choice model) in a city-scale traffic system is high. Thus, a larger number of simulation runs is required to get reliable simulation results. Third, for real-time city-scale simulation-based system, simulated-based assistance (e.g. a detour plan for incident management) has to be fed back to decision makers or drivers in a short period before the simulated-based assistance becomes invalid. However, performance optimization of a city-scale traffic simulator is complex. There is a theoretical lack of a systematic methodology to optimize the performance of a city-scale traffic simulator.

In this thesis, the calibration of a traffic simulator means to estimate variables (e.g. model parameters and model inputs), in order to match model outputs with real-world traffic surveillance measurements. This calibrated set of variables forms a historical database, which is the fundamental of the traffic simulation for future applications in the road network. However, as the simulated road network grows, the number of variables (e.g. OD flows) grows fast. To calibrate a large number of variables in a city-scale traffic simulation is not trivial. For example, when deploying a state-of-the-art academic traffic simulator DynaMIT (Ben-Akiva et al., 1998, 2001) in Beijing and Singapore, the calibration of the traffic simulator took more than one year; when deploying a state-of-the-art industrial traffic simulator Aimsun in Singapore in 2006, great effort and a long time were spent on calibrating hourly O/D matrices. There is a theoretical lack of an effective methodology to calibrate a large number of variables in city-scale traffic simulations.

1.3 Objectives and Contributions

This thesis has two objectives:

- To propose a systematic methodology to optimize the execution performance of a city-scale traffic simulator.
- To propose an effective algorithm to calibrate model parameters and model inputs in city-scale traffic simulations.

This thesis has two major contributions:

- We propose a systematic three-step performance optimization methodology. These steps are: framework optimization, serial bottlenecks optimization and scalability optimization. The three-step performance optimization methodology is successfully demonstrated to reduce the execution time to simulate the Singapore expressway road network from 7:00AM to 8:00AM with in total 106,386 vehicles.
- We propose an enhanced calibration algorithm for city-scale traffic simulations. The algorithm outperforms the state-of-the-art SPSA algorithm in terms of convergence rate and long run achievable goodness-of-fit. The proposed algorithm is successfully demonstrated to calibrate 373,646 time-dependent OD flows in one day in Project DynaMIT on Singapore expressway network.

The detailed contributions of the thesis are listed below:

- Simulating a city-scale congested traffic scenario is time costly. We propose an Entry Time based Supply Framework (ETSF) to reduce the execution time to simulate congested traffic scenarios. The computational complexity of the proposed ETSF framework is less sensitive to the level of congestions. Experiment results show that ETSF outperforms the current supply framework, by reducing the execution time by 50% - 95% in city-scale road networks and congested traffic scenarios.
- A spatial index is a data structure that manages locations of objects (e.g. a vehicle, a pedestrian, etc.) in a traffic simulation. When the number of objects increases in a city-scale traffic scenario, the maintenance cost to rebalance a spatial index becomes a serial bottleneck. We propose Sim-Tree, which is a more stable spatial index, whose balance depends only on the average road density and thus is insensitive to individual vehicles' changing locations. The results of experiments simulating a city-scale traffic scenario on a 6-core machine show that the Sim-Tree performs significantly better than the R*-tree family of spatial indexes.
- One trend in performance optimization is to execute a traffic simulation on multiple processing units (e.g. cores and machines) using parallel simulation technologies and distributed simulation technologies. A new type of hardware, the GPU, which has

hundreds of cores, is gaining popularity in high performance computing, because of its massive computational performance compared to the CPU. A research question is whether the GPU can be a potential high-performance platform for city-scale traffic simulations. This thesis proposes a comprehensive simulation framework to run the proposed ETSF framework on the CPU/GPU platform. The proposed simulation framework is demonstrated on a large-scale artificial grid road network and a real-world Singapore expressway road network.

- As the traffic road network size grows, we found that the state-of-the-art calibration algorithm (SPSA) deteriorates. The reason lies in the systematic error in the method to estimate the gradients. Motivated by this finding, we propose W-SPSA ('W' means Weighted). The key idea of W-SPSA is to incorporate a 2-D weight matrix in the calibration algorithm, to reduce the systematic error in the estimation of the gradients. The 2-D weight matrix represents the correlations between calibration variables and observed measurements. The proposed W-SPSA calibration algorithm is successfully demonstrated to calibrate 373,646 time-dependent OD flows in one day in Project DynaMIT on Singapore Expressway Network.

1.4 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 presents a review of the major components in a traffic simulation, the classification of traffic simulations, the performance optimization methodology and the calibration algorithms of a city-scale traffic simulation. Then, Chapter 3 introduces an Entry Time based Supply Framework (ETSF), which computational complexity and execution performance is less sensitive to the level of congestion. Chapter 4 proposes an efficient two-dimensional spatial index for city-scale traffic simulations. Chapter 5 investigates a comprehensive framework to run city-scale traffic simulations on the CPU/GPU Platform. After that, Chapter 6 presents an enhanced calibration algorithm for city-scale traffic simulations. Finally, conclusions and directions for further research are explained in Chapter 7.

2. Literature Review

2.1 Traffic Simulation

2.1.1 General Structure

A transportation system can be modeled in a software traffic simulator in many ways, depending on the target questions, the available computational resources and the coverage and the accuracy of road network data and traffic surveillance data. However, the "Demand-Supply" methodology has been proven to be successful in the development of a traffic simulator and the study of a traffic system. As shown in Figure 2, a traffic simulator consists of two components: ‘Demand’ and ‘Supply’. Modeling from the travelers’ point of view, the 'Demand' is to understand how travelers’ decisions are made, such as the choice of origin and destination, mode choice, departure time choice, route choice and response to information. Modeling from the road network's point of view, the 'Supply' is to understand the capacity of a road network, and also the corresponding traffic control systems, incident management systems. The interaction between the 'Demand' and the 'Supply' assigns the demand to available road resources to study the equilibrium status. The interaction between the 'Demand' and the 'Supply' can be modeled as: static traffic assignment, if the demand and the supply are assumed to be constant during the study period; and dynamic traffic assignment, if the demand and the supply are time-dependent.

Traffic simulators are mainly used for evaluating a target traffic scenario. A traffic scenario can be an innovative congestion management scheme to reduce the total travel delay in the AM peak of a typical day in a city or a new motorway to satisfy the increasing demand in future. A traffic scenario is configured by model parameters (e.g. parameters in the route choice model) and model inputs (e.g. the road network and the OD matrix). Generally, model parameters are pre-configured or calibrated by simulator developers and model inputs are set by end-users. The output of traffic simulation varies, depending on the required measures of effectiveness. Example measures of effectiveness are mean delay per vehicle and mean travel time per vehicle.

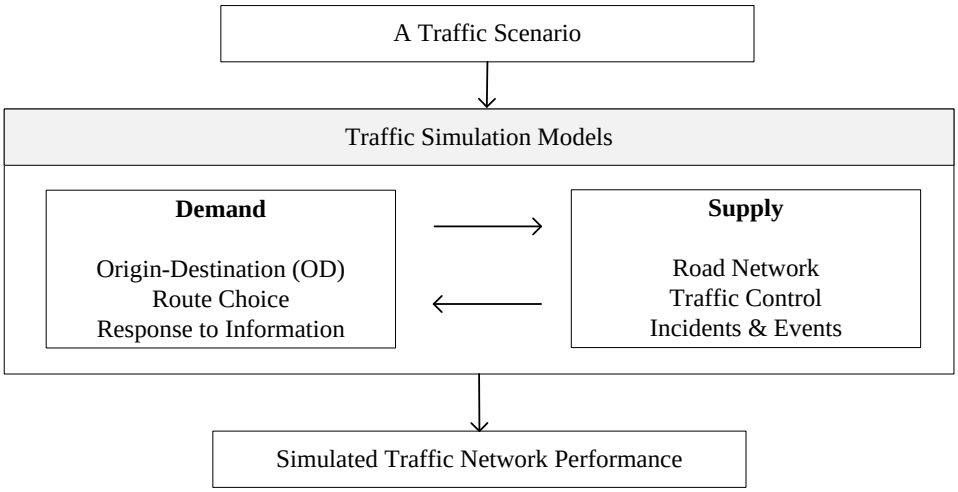


Figure 2: A general structure of traffic simulation models

2.1.2 Demand Models

The demand can be modeled in different ways. The activity-based demand modeling is a more detailed way of describing demand in term of the activities generating the need to move persons and freight. The activity-based demand modeling is introduced in (Bhat and Koppelman, 1999). However, so far, the most widely used way to model the demand in traffic simulations is in terms of an aggregate representation by means of an origin-destination (OD) matrix.

A traffic road network is divided into a number of traffic analysis zones (TAZs or zones). Each zone generates (or as the origin) and attracts (or as the destination) trips. The size and shape of a zone varies, depending on the size of the traffic road network and the target problem. Each zone is represented using a dummy node (or a centroid), and then connects to the traffic road network through dummy links (or centroid connections). The demand is then modeled as an OD matrix, whose rows are origin zones and whose column are destination zones. Each number in the matrix represents the number of trips going from an origin zone to a destination zone during a period. In most cases, the demand contains a number of OD matrix tables, representing different vehicle types in different periods. Table 1 synthesizes an OD representation of 3 zones from 07:00AM to 08:00AM. T_{ij} is the number of trips from the origin zone i to the destination zone j during the hour; T_{SS} is the total number of trips during the period. Given an OD matrix and an assumed distribution of the departure time (e.g. uniform distribution), vehicles are generated and inserted into the road network at the origin zone.

Table 1: An example Origin-Destination (OD) matrix from 07:00AM to 08:00AM

Origin \ Destination	1	2	3	Sum
1	T_{11}	T_{12}	T_{13}	T_{1S}
2	T_{21}	T_{22}	T_{23}	T_{2S}
3	T_{31}	T_{32}	T_{33}	T_{3S}
Sum	T_{S1}	T_{S2}	T_{S3}	T_{SS}

After a vehicle is generated, various travelers' decision makings are simulated, including departure time choice, mode choice and route choice for both pre-trip and en-route. Pre-trip decisions include travelers' choice of mode, departure time, as well as route under the expected travel conditions before taking the trip, while en-route decisions are made by travelers based on real time information and travel condition during the trip. The most widely used theory to model travelers' behavior models is "discrete choice method" (Ben-Akiva et al., 1985). Taking the route choice model as an example, drivers are modeled as independent decision makers, who choose one single route from a set of alternative routes from an origin to a destination in a road network. The first step is to generate alternative routes. In a real network, there might be a huge number of candidate routes between an OD pair. For computational efficiency, different choice set generation algorithms (Ben-Akiva et al., 1984; Azevedo, 1993; Bovy et al., 2006) are developed to build a smaller subset of routes that includes as much as possible the routes actually chosen by drivers in reality while eliminating routes that are never selected. Once the alternative routes are generated, a route choice model is developed. In the model, each driver is described by a vector of characteristics (e.g. driving age) and each alternative route in the choice set is described by a vector of attributes (e.g. total distance and historical time cost). Each driver perceives a utility associated with each route, which is a real number mapped from the characteristics of the driver and the attributes of the route. Discrete choice method assumes that each individual driver will select the route that has the maximum perceived utility. In reality, however, the utilities are not directly measured. Random utility theory captures the discrepancy between the systematic model utilities and the "true" utilities using an error term, where the systematic utility is calculated as a linear function of the characteristics of a driver and the attributes of an alternative route. Different models assume different distributions of the error term. The Multinomial Logit (MNL) model is one of the most attractive models in real applications due to its simple assumption on

identically and independently distributed Gumbel errors and its closed-form formula to compute the probability of selecting an alternative in the alternative routes. Then, C-Logit model (Cascetta et al., 1996) and Path Size Logit model (Ben-Akiva et al., 1999) were proposed to capture the overlapping between candidate routes. Travelers' behavior models are furthered explained in (Ben-Akiva et al., 1985).

2.1.3 Supply Models

The current trend to model a road network is to translate a real-world road network into a directed graph in a two-dimensional coordinate space. However, a road network can be modeled with different granularities. For example, an intersection can be modeled as one node, ignoring detailed drivers' behaviors in the intersection. On the other hand, an intersection can also be modeled as a set of nodes and links, with the capability to capture drivers' behaviors at all turnings in the intersection. In this thesis, as shown in Figure 3, a road network is modeled as a directed graph including nodes, links, segments and lanes. The nodes correspond to intersections of the actual network, while links represent unidirectional pathways between nodes. Each link may be divided into many segments because of geometrical differences. Each segment contains a number of physical lanes. Nodes, links, segments and lanes have specific attributes, reflecting their facility characteristic. Centroid nodes (if existing) are modeled as a special type of nodes.

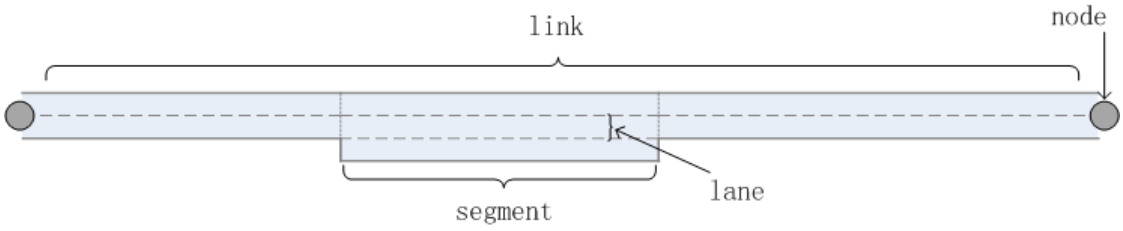


Figure 3: Road network related terminologies

Based on the level of detail, there are four general approaches to model traffic movements (or traffic dynamics) on a road network: macroscopic, microscopic, mesoscopic and nanoscopic. Macroscopic simulators, such as METANET (Wang et al., 2001), use an aggregate point of view based on a hydrodynamic analogy by representing traffic flow as a fluid process whose state is characterized by aggregate macroscopic variables: density, volume and speed. Microscopic simulators, such as MITSIMLab (Yang et al., 2000), PARAMICS (Cameron et al., 1995),

VISSIM (Fellendorf and Vortisch, 2014), AIMSUN2 (Barcelo, 2002), can provide more details by modeling vehicular interactions with surrounding individual vehicles, various transportation facilities and traffic information. Mesoscopic traffic simulators, such as DynaMIT (Ben-Akiva et al., 1998), DYNASMART (Mahmassani et al., 1992), Mezzo (Burghout et al., 2006), CONTRAM (Taylor, 2003), and MATSIM (Cetin, 2005), are designed as a combination of microscopic traffic simulators and macroscopic traffic simulators. Capturing the most essentials of the traffic flow dynamics, they are computationally more efficient than microscopic traffic simulators. Nanoscopic traffic simulators, such as Commuter (Commuter, 2014), can provide the most detail by capturing the decision making process in people's mind and modeling person-oriented door-to-door trips. These four approaches are further explained in Section 2.2.

Traffic control is another critical component in the Supply models. Examples of traffic control systems are: intersection control (or signal control), ramp metering control, road pricing control, variable message signs (VMS), variable speed limit signs (VSLS) and reversible lanes. In general, traffic control systems are implemented as optional modules in traffic simulators, which can dynamically modify the configuration (e.g. the speed limit), the capacity (e.g. red signal and lane close), the usage cost of the road network (e.g. road pricing), etc.

2.2 Classification of Traffic Simulations

This section discusses the four types of traffic simulators: macroscopic, microscopic, mesoscopic and nanoscopic. They have different capabilities and solve different problems.

2.2.1 Macroscopic Traffic Simulation

The idea of macroscopic traffic simulations is to model a traffic system as a particular fluid process and to capture the time-space evolution of the fluid. As shown in Figure 4, a traffic network is divided into a number of sections. Each section *x* at each time *t* is represented in a macroscopic way as its speed u(x,t), density k(x,t) and flow q(x,t). Vehicles are assumed to be evenly distributed in the section and moving using the same speed. Besides, each section might have a generation (or dissipation) flow.

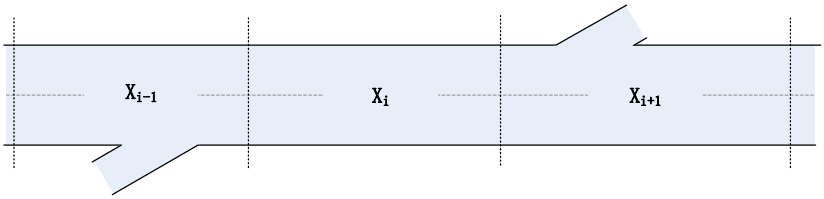


Figure 4: An example section of a road network in macroscopic traffic simulations

The simulation period is divided into a number of time intervals. The traffic movement in each time interval is conducted using three rules. The first rule is the fundamental relationship:

$$q(x,t) = u(x,t) * k(x,t) \tag{2.1}$$

The second rule is the hypothesis that speed is a function of density, which is also known as the speed-density relationship. A general form of the speed-density relationship (May et al., 1976) is shown below:

$$v = v_0 * \left\{ (1 - (\frac{k}{k_{jam}})^b)^a \right\} \tag{2.2}$$

where, v_0 is the free-flow speed, k_{jam} is the jam density, α, β are configurable parameters to be determined through calibration. However, the hypothesis is only valid in equilibrium conditions. Extensions and evaluations of this speed-density relationship are available in (Payne, 1979; Messmer et al., 1990). To account for dynamic efforts of flow behavior, the third rule represents the evolution of the density of a road section with the immediately adjacent sections (both the upstream and the downstream) at the current time step. An example formula is shown below:

$$k_j^{t+1} = \frac{1}{2}(k_{j+1}^t + k_{j-1}^t) - \frac{\Delta t}{2\Delta x}(q_{j+1}^t - q_{j-1}^t) + \frac{\Delta t}{2\Delta x}(g_{j+1}^t + g_{j-1}^t) \tag{2.3}$$

where, k_j^t, q_j^t are the density and flow on road section j at time t; Δt, Δx are the size of a time interval and the length of a road section; g_j^t is the generation (or dissipation) rate on section j at time t. If no sinks or sources exist on the section, g_j^t is 0.

A general macroscopic traffic simulation procedure is shown in Table 2. The procedure consists of 2 layers of loops. First, it is a loop of simulation time steps. Second, during each time step, the framework has a loop of sections. For each section, the density of the section is updated

using the third rule. After the density is determined, the speed at this time step is calculated using the second rule. Then, the flow is calculated using the fundamental relationship.

Table 2: A general procedure of a macroscopic traffic simulation

Inputs: A road network, a traffic scenario, parameters of flow models	
Outputs: Aggregate road measurements (e.g. speed) at each time step;	
1	Initialize a macroscopic traffic simulation environment
2	***for*** each time step ***do***
3	… load new vehicles;
4	… ***for*** each section in the network ***do***
5	… ... update density of the section; (using the third rule)
6	… ... update speed of the section; (using the second rule)
7	… ... update flow of the section; (using the first rule);
8	… ***end for*** // *section loop*
9	… output simulated traffic condition at this time step;
10	***end for*** // *time loop*

One example macroscopic traffic simulation is METANET (Wang et al., 2001), which was developed in 1989 for motorway network simulation. In METANET, a motorway network is converted into a directed graph with links and nodes. Links should have homogeneous geometric characteristics such as number of lanes. Links are divided into sections of equal length (typically 500 meters). A typical time step in METANET is 10 seconds. The demand is modeled as inflows in boundary links. The route choice behavior is described by use of splitting rates at nodes. The overall modeling approach allows for simulation of traffic conditions (e.g. free-flow, congested) and capacity-reducing events (e.g. incidents) in a motorway network. Due to low computational cost, METANET can be used for real-time simulation-based decision support. The drawback in macroscopic traffic simulations is the insufficient capability to model individual driver's route choice behavior.

2.2.2 Microscopic Traffic Simulation

The idea of microscopic traffic simulations is to capture the detailed actions (e.g. acceleration, deceleration, lane changing, and giving ways) of each individual driver in response to the surrounding traffic. The core behavior models include car following models (e.g. the Pipes' safe driving model in (Pipes, 1953), the Gipps car following model in (Gipps, 1981) and the NGSIM model in (Hwasoo et al., 2009)), lane changing models (e.g. the Gipps lane changing model in (Gipps, 1986)) and gap acceptance models (Ahmed et al., 1996).

A car following model captures drivers' response (acceleration or deceleration) to a stimulus (e.g. relative distance, relative speed) from the leading vehicle. The simplest linear car following model assumes that acceleration of the following car is directly proportional to the relative speed (the speed of the leading car - the speed of the following car). If the relative speed is positive, the response is acceleration; otherwise, the response is deceleration. Gazis et al. (1959) proposed an enhanced model assuming that the response is inversely proportional to the relative distance. Gipps (1981) developed an empirical model consisting of two components: the first component represents the intention of a vehicle to achieve its desired speed; the second component represents the limitation imposed by the leading vehicle to drive safely. The Gipps car following model is widely used in current traffic simulations (e.g. Aimsun). An overall view of advantages and disadvantages of different car following models are in Panwai and Dia (2005). A lane changing model is a decision model that approximates a driver's behavior to change lanes or not. The driver's lane changing behavior depends on the distance from the driver's locations to the next turning. If the distance is long, a lane changing only happens if the driver cannot achieve its target speed and one of the neighboring lanes provides a better situation downstream (It is also known as Discretionary Lane Change). If the distance is short, a lane changing happens if the driver is not moving on the correct lane in order to follow its path (It is also known as Mandatory Lane Change). Car following models and lane changing models are integrated in some traffic simulations (e.g. MITSIMLib). A gap acceptance model is used to model the give-way behavior. It determines whether a lower priority vehicle approaching an intersection can or cannot cross depending on the circumstances of high priority vehicles (positions and speed). The model takes into account of the distance of vehicles to the hypothetical collision point, their speeds and their acceleration. It then determines the time needed by the vehicles to clear the intersection.

A general microscopic traffic simulation procedure is shown in Table 3. The procedure consists of 4 layers of loops. First, it is a loop of simulation time steps. Second, during each time step, the framework has a loop of links and lanes. Then, each vehicle on a lane is updated. First, a vehicle fetches the surrounding traffic (e.g. the leading vehicle), applies corresponding models (e.g. car-following and lane-changing), and then updates its speed and location. The procedure also outputs aggregate attributes (e.g. density and queue) of links and lanes in the road network.

Table 3: A general procedure of a microscopic traffic simulation

Inputs: A road network, a traffic scenario, parameters of user behavior models	
Outputs: Trajectories of vehicles at each time step; Aggregate road measurements (e.g. density and queue) at each time step;	
1	Initialize a microscopic traffic simulation environment
2	***for*** each time step ***do***
3	… load new vehicles;
4	… ***for*** each link in the network ***do***
5	… ... ***for*** each lane in the link ***do***
6	… ... … ***for*** each vehicle on the lane ***do***
7	… ... … … fetch its surrounding traffic; //e.g. the leading vehicle
8	… ... … … apply corresponding behavior models: //e.g. car-following
9	… ... … … update the vehicle's speed and location;
10	… ... … ***end for***
11	… ... ***end for***
12	… ... update aggregate measurements (e.g. density and link travel time);
13	… ***end for*** // *end of link loop*
14	… output simulated traffic condition at this time step;
15	***end for*** // *end of time loop*

2.2.3 Mesoscopic Traffic Simulation

The idea of mesoscopic traffic simulations is to efficiently capture both individual drivers' trip behaviors and aggregate speed-density relationships. Figure 5 illustrates vehicles in a lane of a link in a mesoscopic traffic framework, which explicitly or implicitly split the lane into two parts: the moving part and the queue part. The location of a vehicle is modeled as <lane_ID, offset>. The moving part is the part of the link where vehicles are not yet delayed by the queue at the downstream node. In Figure 5, vehicles 1-4 are in the moving part. X1 is the offset of V1 and L1 means V1 is moving in lane L1. The queue part is the part of the link where vehicles are blocked in the downstream. Vehicles 5-9 are in the queue part. X5 is the offset of V5. In the process of simulating the vehicle movement, vehicles on a link are updated in a reverse order from the downstream end to the upstream end.

According to the vehicle's status (moving or in queue), its location is updated using different rules. If a vehicle is located in the moving part of a link, its speed is determined by a speed-density relationship on the density of the moving part and its location is then updated using the speed. If a vehicle is located in the queue part of a link, there are two possible conditions:

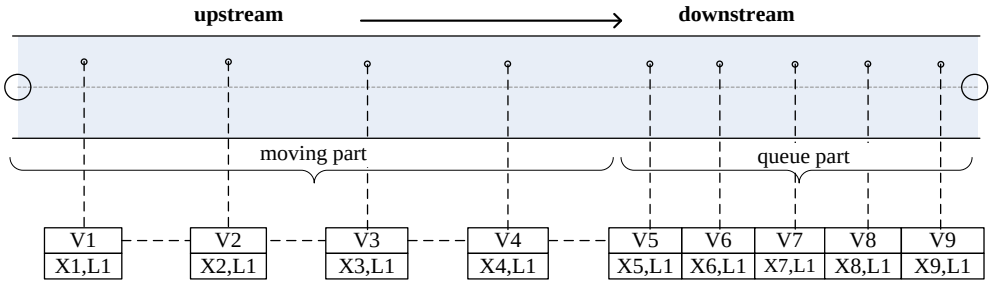


Figure 5: Vehicles in a lane of a link in a mesoscopic traffic framework

- If the vehicle is at the head of the queue (at the exit of the link), it can leave the queue only if the current link has output capacity left and the downstream link has sufficient empty space and sufficient input capacity. If conditions are satisfied, the vehicle can pass the current link to the downstream link. Otherwise, the vehicle stays in the queue.
- If the current vehicle is not first in the queue (i.e., there are other queuing vehicles ahead of it), it can only advance as far as vehicles in front of it do (assuming no space is left between any two consecutive queuing vehicles). The distance is then determined by how many vehicles have left the head of the queue during the same time step.

A general mesoscopic traffic simulation procedure is shown in Table 4. The procedure consists of 4 layers of loops. First, the framework has a loop of simulation time steps. Second, during each time step, the framework has a loop of simulating the traffic dynamics in each link. Third, each link contains multiple lanes. Fourth, the framework computes and updates the locations and status of each individual vehicle in each lane. The process of simulating a lane is shown in steps 6-14. In the beginning of the process, the speed of the lane is calculated using Formula 2.2. Then, each vehicle on the lane is simulated to change its location on the current lane or pass to the next lane.

2.2.4 Nanoscopic Traffic Simulation

Nanoscopic traffic simulations are comparatively new. The general idea of a nanoscopic traffic simulation is to model the traffic system in the view of people, instead of vehicles. There are two main features: to capture the decision making process in people's mind and to capture the most detailed person's door-to-door behavior. First, rules in the decision making process are derived from the knowledge of human perception and cognition. Second, each person may travel using

different modes of transport (e.g. driving, walking, cycling, or be a passenger on a train, bus or taxi). One example nanoscopic traffic simulation is Commuter (Duncan et al., 2014), which is designed for evaluating alternative infrastructure plans. To the best of our knowledge, the computational framework of nanoscopic traffic simulations is not available in the literature and is thus not discussed in this section.

Table 4: A general procedure of a mesoscopic traffic simulation

Inputs: A road network, a traffic scenario, parameters of user behavior models	
Outputs: Aggregate road measurements (e.g. speed and queue) at each time step;	
1	Initialize a microscopic traffic simulation environment
2	***for*** each time step ***do***
3	… load new vehicles;
4	… ***for*** each link in the network ***do***
5	… ... ***for*** each lane in the link ***do***
6	… ... … update speed of the lane
7	… ... … ***for*** each vehicle on the lane ***do***
8	… ... … … ***if*** the vehicle is in the queue part ***then***
9	… ... … … … pass the vehicle to next lane or update its location in the queue part;
10	… ... … … ***else***//*then the vehicle is in the moving part*
11	… ... … … … update the vehicle's location in the moving part
12	… ... … … ***end if***
13	… ... … ***end for*** // *vehicle loop*
14	… ... ***end for*** // *lane loop*
15	… ... update aggregate measurements (e.g. density and queue);
16	… ***end for*** // *link loop*
17	… output simulated traffic condition at this time step;
18	***end for*** // *time loop*

2.2.4 Summary

Table 5 shows the comparison of these four traffic simulations in functional capability, use cases and simulation network scale. Macroscopic traffic simulation is the most time efficient to simulate the traffic flow movement. However, it lacks some critical functions, e.g. drivers' route choices. Microscopic traffic simulation has more functions, e.g. acceleration/deceleration, lane changing, giving way. It enables microscopic traffic simulation to be able to evaluate a range of existing traffic control systems (e.g. signal control, ramp metering control, road price control and incident response). Mesoscopic traffic simulation achieves a reasonable balance between functions and computational complexity. It reserves key functions (e.g. route choices) in

microscopic traffic framework, and performs much more efficiently than microscopic traffic simulation. Mesoscopic traffic simulation has been widely used in city-scale traffic simulation (Cetin, 2005; Ben-Akiva et al., 2012). Nanoscopic traffic simulation captures the most detail in decision making process in peoples' mind and peoples' door-to-door behavior. It is mainly used in traffic safety related study and infrastructure planning (e.g. to build a new airport). In this thesis, we focus on mesoscopic traffic simulations and microscopic traffic simulations.

Table 5: Functions and performance of three traffic simulation frameworks

	Functional Capability	Use Cases	Scale
Macroscopic	flow movement	traffic flow study	city-scale
Mesoscopic	drivers' route choices, flow movement	evaluate new technologies and policies (e.g. road pricing)	city-scale
Microscopic	acceleration/ deceleration, lane changing, drivers' route choices, flow movement	evaluate new technologies and policies (e.g. signal control)	region-scale
Nanoscopic	decision making process, acceleration/ deceleration, lane changing, multi-model route choices, flow movement	safety related study, infrastructure planning	small area

2.3 Performance Optimization of City-scale Traffic Simulations

2.3.1 Three-step Performance Optimization Methodology

The performance of a city-scale traffic simulation is measured by two criteria:

Computational Complexity: A typical peak-hour city-scale traffic scenario contains a large number of intersections, links and drivers. Computational complexity measures the sensitivity of the execution performance of simulating a traffic scenario to the number of intersections, links and drivers. The big-O notation is used in the analysis of the computational complexity. The big-O notation is explained in (Cormen, 2009).

Hardware Scalability: Hardware scalability is a measure of using additional processing units (cores and machines) to reduce the execution time of a traffic simulation. Time speed-up, which is equal to the ratio of the execution time of a traffic simulation on 1 processing unit and the execution time of the traffic simulation on N processing units, is utilized to measure the hardware scalability. In this thesis, scalability is the same as hardware scalability and speed-up is the same as time speed-up, unless otherwise specified.

If a city-scale traffic simulation has a low computational complexity, the traffic simulation is defined as efficient; if a city-scale traffic simulation has a high speed-up (e.g. near-to-linear), the traffic simulation is defined as scalable. If a city-scale traffic simulation is efficient and scalable, the traffic simulation is defined as high-performance.

In this thesis, a systematic three-step performance optimization methodology is proposed to optimize the performance of a city-scale traffic simulator. These three steps are:

1) Framework optimization
2) Serial bottleneck optimization
3) Scalability optimization

"Framework optimization" aims to optimize the computational complexity of simulating a city-scale traffic scenario to the number of intersections, segments and drivers. It is essentially a compromise between the functional capability of the traffic simulation and the computational complexity of the traffic simulation. An extremely high fidelity traffic simulation, which captures behaviors and motivations of all objects in a traffic scenario in a fine-grained time step (e.g. 0.1 seconds), is able to simulate complex phenomenon in a traffic system. However, such a framework is computationally expensive. On the other hand, an extremely low fidelity traffic simulation, which captures only the fluid process of traffic flows in a large time step (e.g. 60 seconds) and ignores behaviors of individual drivers, is computationally efficient. However, such a framework tends to be functionally limited to mimic practical solutions to real-world problems. "Framework optimization" aims to reduce the computational complexity of the framework, while the traffic simulation is still able to simulate the traffic scenarios with an acceptable accuracy.

"Serial bottleneck optimization" and "scalability optimization" aim to improve the hardware scalability of a city-scale traffic simulation on parallel & distributed platforms. Traditionally,

traffic simulations (e.g. PARAMICS (Cameron et al., 1995) and MITSIM (Yang, 1999)) have been written for serial computation. Traffic frameworks and models are constructed and implemented as a serial stream of instructions. These instructions are executed on a central processing unit (CPU) on one computer. Only one instruction may execute at a time - after that instruction is finished, the next is executed. One of the features of CPUs has been its steadily increasing computational performance in the last century. However, in the 2000s, this increase came to a stop. At the same time, parallelism appears to be a sustainable way of increasing computational performance. However, assigning more computational resources to a traffic simulation does not naturally reduce the execution time of simulating a traffic scenario. Amdahl's law is a model for the relationship between the expected speedup of parallelized implementations of an algorithm relative to the serial algorithm, under the assumption that the problem size remains the same when parallelized. Amdahl's law is shown as Formula 2.4.

$$S(n) = \frac{T(1)}{T(n)} = \frac{T(1)}{T(1) * B + T(1) * (1 - B) * \frac{1}{n}} = \frac{1}{B + \frac{1}{n}(1 - B)} \tag{2.4}$$

where *S(n)* is the speedup when using n processing units, T(1) is the execution time of simulating a traffic scenario on 1 processing unit, T(n) is the execution time on n processing units and B is the time cost of the serial part of the source code. As n tends to infinity, the maximum speedup tends to (1/B). As an example, if B is 10%, the maximum speedup is 10; if B is 90%, the maximum speedup is 10/9, no matter how many processing units are utilized. Serial bottleneck optimization is to reduce B. Amdahl's law also assumes the problem is evenly divided into available processing units, without causing additional cost. However, parallelism always comes with a cost *f(n).* As *n* becomes larger, *f(n)* tends to be more costly. The practical speedup is calculated as Formula 2.5.

$$S(n) = \frac{1}{B + \frac{1}{n}(1 - B) * \mathrm{f}(n)} \tag{2.5}$$

The aim of scalability optimization is to reduce the cost *f(n),*by improving load balance, reducing the communication cost and adapting new hardware (e.g. the GPU).

The order of these three steps in the performance optimization methodology is fixed. First, "framework optimization" is done before "serial bottleneck optimization". Otherwise, "serial bottleneck optimization" may become useless, because a serial bottleneck in the previous framework may not exist after framework optimization. Second, only if "framework optimization" and "serial bottleneck optimization" are completed, the maximum speedup can be estimated. The distance between the practical speedup and the maximum speedup is an important indicator of the effectiveness and completion of "scalability optimization".

2.3.2 Framework Optimization

There are three types of well-studied traffic simulation frameworks: macroscopic, microscopic and mesoscopic. These three types of traffic simulation frameworks were explained in Section 2.2. This section analyses the computational complexity of these three types of traffic simulation frameworks and then explains the existing researches in framework optimization. Besides, to the best of our knowledge, there is no clear definition of the computational framework of the nanoscopic traffic simulation framework, thus, the computational complexity of the nanoscopic traffic simulation framework is not included in this section.

The simulation framework of macroscopic traffic simulations was introduced in Section 2.2.1. The computational complexity of a macroscopic traffic simulation framework is shown as:

$$\theta(TT/\Delta t2 * NS * C_3) \tag{2.6}$$

$$\text{-> } \theta(TT/\Delta t2 * NS) \tag{2.7}$$

where, TT is the total simulation period, $\Delta t2$ is the simulation time step (e.g. 1-10 seconds), NS is the number of sections in a road network, and C_3 is the constant amortized time cost to update the density, speed and flow of a section following the three rules in Section 2.2.1. "An amortized time cost" is a concept in the big-O notation to describe that while certain operations may be extremely costly, they cannot occur at a high enough frequency to weigh down the entire framework. In this thesis, "an amortized time cost" can be understood as "a constant mean time cost". Thus, the computational complexity of a macroscopic traffic simulation framework is proportional to the number of sections, and is inversely proportional to the simulation time step.

The simulation framework of microscopic traffic simulations was introduced in Section 2.2.2. The computational complexity of a microscopic traffic simulation framework is shown as:

$$\theta(TT/\Delta t1 * \mathrm{NLi} * \mathrm{NLa} * (\mathrm{C}_1 + C_2 * \bar{n})) \tag{2.8}$$

$$\text{-> } \theta(TT/\Delta t1 * \mathrm{NLi} * \mathrm{NLa} * (\bar{n})) \tag{2.9}$$

$$\text{-> } \theta(TT/\Delta t1 * n) \tag{2.10}$$

where, TT is the total simulation period, $\Delta t1$ is the simulation time step (e.g. 0.1-1.0 seconds), NLi is the number of links, NLa is the average number of lanes in each link, $\bar{n}$ is the average number of vehicles on a lane, C_1 is the constant amortized time cost to update the status (e.g. density and speed) of a lane, C_2 is the constant amortized time cost to apply drivers' behavior models to update the status (e.g. speed and location) of a vehicle, $\mathrm{NLi} * \mathrm{NLa} * (\bar{n})$ is actually the aggregate number of vehicles in the road network. If we use n to denote this total number, the time complexity is $\theta(TT/\Delta t1 * n)$. Thus, the computational complexity of a microscopic traffic simulation framework is directly proportional to the number of vehicles, and is inversely proportional to the simulation time step.

The simulation framework of mesoscopic traffic simulations was introduced in Section 2.2.3. The computational complexity of a mesoscopic traffic simulation framework is shown as:

$$\theta(TT/\Delta t3 * \mathrm{NLi} * \mathrm{NLa} * (\mathrm{C}_4 + \mathrm{C}_5 * \bar{n})) \tag{2.11}$$

$$\text{-> } \theta(TT/\Delta t3 * \mathrm{NLi} * \mathrm{NLa} * (\bar{n})) \tag{2.12}$$

$$\text{-> } \theta(TT/\Delta t3 * n) \tag{2.13}$$

where $\Delta t3$ is the simulation time step (e.g. 1-10 seconds), NLi is the number of links, NLa is the average number of lanes of each link, $\bar{n}$ is the average number of vehicles on a lane, C_4 is the amortized time cost to update the status (e.g. density) of a lane, C_5 is the amortized time cost to update the status (e.g. location) of each vehicle. $\mathrm{NLi} * \mathrm{NLa} * (\bar{n})$ is actually the aggregate number of moving vehicles in the road network. If we use n to denote this number, the time

complexity is $\theta(TT/\Delta t3 * n)$. Thus, the computational complexity of a mesoscopic traffic simulation framework is directly proportional to the number of vehicles, and is inversely proportional to the simulation time step.

Even though the computational complexity formulas of a microscopic traffic simulation framework and a mesoscopic traffic simulation framework are similar, a microscopic traffic simulation tends to be much more time costly than a mesoscopic traffic simulation. First, the simulation time step in a microscopic traffic simulation is smaller. For example, in microscopic traffic simulations, the default time step in Paramics (Cameron et al., 1995) is 0.5 seconds and the default time step in Transmodeler (TransModeler, 2014) is 0.1 seconds. However, in mesoscopic traffic simulations, the default time step in DynaMIT (Ben-Akiva et al., 1998) is 5 seconds. Second, the time cost of updating individual vehicle's movement at each time step in microscopic traffic simulations (using disaggregate car-following model, lane changing model and gap acceptance model) is higher than the time cost of updating individual vehicle's movement at each time step in mesoscopic traffic simulations (using aggregate speed-density relationship).

Another noticeable methodology in framework optimization is the hybrid traffic simulation framework (Laval, 2004; Burghout et al., 2006). The idea is to choose a high-resolution simulation framework (e.g. microscopic) in the area of interests to evaluate new systems or technologies, which often require the detailed modeling of vehicles, combined with a low-resolution simulation framework (e.g. mesoscopic) in the surrounding areas to capture network effects of local phenomena.

2.3.3 Serial Bottleneck Optimization

There are three general serially costly components in a traffic simulation framework:

1) Synchronization
2) Inputs & outputs
3) Dynamic spatial index

Synchronization is common in parallel & distributed traffic simulations. Synchronization of processors (or threads) mean that multiple processes (or threads) are to join up or handshake at a certain point, in order to reach an agreement or commit to a certain sequence of action. There are two common types of synchronization: locks (or mutex) and barriers. First, locks are used to

protect against data races to allow thread-safe synchronization of data between threads. Each time a thread accesses data (e.g. a file or a memory space), the thread waits for the data's lock, locks the data, modifies the data and then unlocks the data. If the data's lock is not available, the thread is blocked. Lock reduces the level of parallelism and should be avoided if possible. (Kumaret al., 1994) discusses methods to efficiently use locks in parallel systems. Second, a barrier is another type of synchronization between multiple processes (or threads). A barrier is configured for a particular number of processes (*n*), and as processes reach the barrier they must wait until all *n* processes have arrived. Once the *n*-th process has reached the barrier, all the waiting processes can proceed, and the barrier is reset. Barriers widely exist in parallel & distributed traffic simulation frameworks. There are two general methods to optimize the performance of barriers. The first method is to increase the workloads between successive barriers. It is achievable by increasing the problem size or reducing the frequency of using barriers. The second method is to optimize the load balance of workloads in processes or threads. Load balancing algorithms are introduced in Section 2.3.4.

Input & Output (I/O) is a widely existing serial costly component in parallel & distributed traffic simulation frameworks. I/O sources include files (text or binary), XML, databases and the internet. For example, in the initialization phrase of a traffic simulation, a road network and the configuration parameters are loaded from files or databases. In each time step within a traffic simulation, the simulated results are written back to files or databases. In real-time simulation-based applications, the traffic surveillance data is periodically loaded from files or the internet. Generally, there are three methods to optimize the I/O performance. The first method is a connections pool (Ramakrishnan and Gehrke, 2003). Creating a connection to a file or a database is time costly. Connections pool is a technology that is widely used to avoid the creation and release of large number of connections. The second method is data buffer. If a write (or update) operation does not require an immediate effect (e.g. on the disc), buffering the operations can significantly improve the I/O performance. If the I/O performance is not satisfied, even if the disc or the internet reaches its maximum speed, parallel I/O technologies (Jin, 2001; May, 2001) should be considered. The idea of parallel I/O technologies is to spread the I/O operations across multiple machines and each machine executes a portion of the I/O operations simultaneously.

Dynamic spatial index is another serial costly component. Dynamic spatial index is a data structure that manages locations of moving objects (e.g. a vehicle) in a simulated road network. The major responsibility of a dynamic spatial index is to allow any object to query its surrounding traffic (e.g. the leading vehicle), which is a fundamental function to support various traffic behavior models to determine a vehicle's speed and location. Based on the number of dimensions in objects' locations, the spatial index is classified into one-dimensional spatial indexes (e.g. linear referencing (Noronha et al., 2002)) and two-dimensional spatial indexes. In one-dimensional spatial indexes, the location of an object is formulated as <lane_ID, offset>. Each object's location is associated with a lane. In two dimensional spatial indexes, the location of an object is formulated as <latitude, longitude>, which does not depend on its underlying geometries. One-dimensional spatial indexes are widely used in both mesoscopic simulation frameworks and microscopic simulation frameworks. However, two dimensional spatial indexes are receiving more attention recently, in order to model more complex traveler's behaviors (e.g. cooperation driving, driving in an intersection, the interaction between pedestrians and drivers). An example two-dimensional region query is shown in Figure 6.

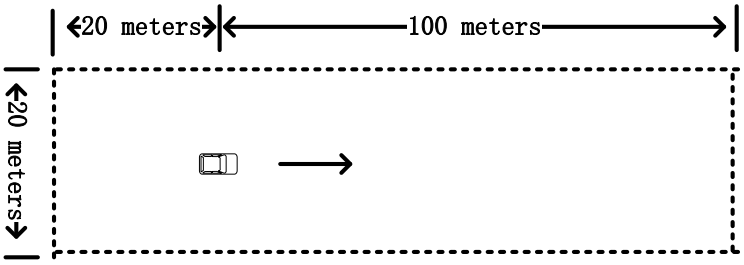


Figure 6: A region query using a two-dimensional spatial index (a plan view)

Indexing moving objects in a two-dimensional space is a hot research topic in spatial simulations (Koh et al., 2011; Othman et al., 2013), computer games and spatial databases (Navathe et al., 2010). R-Tree (Guttman, 1984), R*-Tree (Beckmann et al., 1990) and extended B+-Tree (Jensen et al., 2004) are three of the most popular two-dimensional tree-based spatial indexes and remain a focus of attention in the research community. The key idea of tree-based two-dimensional spatial index is to map spaces into nodes on a tree, and objects within a space are linked to the corresponding node. A region query in a city-scale road network is then transferred into an efficient region query on a tree, which scans only a smaller portion of the road

network. Considerable work has been done to reduce the I/O cost of tree-based spatial indexes (Navathe et al., 2010). In current city-scale microscopic traffic simulations, in most cases, data can fit into the main memory. So, this thesis focuses on two-dimensional spatial indexes in the main memory. The major drawback of using tree-based spatial indexes in main memory (e.g. R-Tree and R*-Tree) is the heavy time cost of update operations. In the case of frequent location update operations in city-scale traffic simulations, there will be lots of node splitting and node merging, making the time cost of update operations high. To support frequent updates in the R-family trees, (Kwon et al., 2002) proposes the LUR-tree. The main idea of the LUR-tree is to update the structure of the index only when an object moves out of the corresponding node. If a new position of an object is in the same leaf node, it changes only the position of the object in the same leaf node. Besides, a secondary index (a hash table) is introduced in the LUR-tree to allow starting an update operation from the bottom. This method can update positions of an object quickly and reduce the update cost. The work in (Lee et al., 2003) proposes a similar bottom-up update function to reduce the cost of frequent updates. Another branch of work (Jensen et al., 2004) maps two-dimensional data to a one-dimensional space by using a recursive space-filling curve and then inserts the data to a B+-Tree. As these works are based on B+-Tree, it can be easily integrated to existing DBMSs. The most relevant recent work to our research is MOVIES (Dittrich et al., 2009). This approach does not require a sophisticated index structure to be adjusted for each incoming update. Instead, it constructs conceptually simple short-lived throwaway indexes which are only kept for a very short period of time (sub-seconds). A recent summary of tree-based spatial index can be found in (Ilarri et al., 2010).

A particular city-scale traffic simulation tends to have specific serially costly components. For example, in DynaMIT-R, which is a real-time simulation-based traffic flow prediction system, real-time OD estimation is a specific serially costly component. Optimization of system-dependent serial components is not disscussed in this thesis. Besides, (Wen, 2009) introduces some general optimization thoughts, which are also important in serial bottleneck optimization.

2.3.4 Scalability Optimization

Parallel & distributed traffic simulation received much attention in the last decade in both academic traffic simulations (Lee et al., 2002; Liu et al., 2004; Wen, 2009; Aydt et al., 2013) and

commercial traffic simulations (Cameron et al., 1996; Nagel at al., 2001; Nokel et al., 2002). The target in parallel & distributed traffic simulations is "scalability optimization". There are four key areas in scalability optimization: map decomposition, workload estimation, boundary processing and investigating emerging hardware.

Map Decomposition Algorithms

Map decomposition algorithms are fundamental to scalability optimization of a city-scale traffic simulation. The purpose of map decomposition algorithms is to divide a large traffic road network into small partitions, so that each thread or process simulates only a partial road network. As shown in Figure 7, the problem of map decomposition can be divided into four layers. The bottom layer is the road network level. This layer represents traffic facilities that constitute the road network. This layer can be modeled as nodes and links. The second layer is the vehicle movement level. This layer represents vehicle behaviors on the road network. This layer can be modeled as speed, flow and density on links (or segments). The third layer is the traffic control level. This layer represents various traffic control systems to control the demand and the supply. The top layer is the partitioning level. This layer represents the solution to divide the road network. This layer can be modeled as the list of cells in each partition. Different types of cells can be defined for different requirements. In this example, each cell is centered by one node and cuts neighbor links in the middle. The benefit of road network partitioning is that each computer simulates only the vehicle movement and traffic control in its own partial road network, which might reduce the total simulation time. The cost of road network partitioning comes from the need to send messages between partitions, if for example, one vehicle moves from one partition to another, or the traffic signal control in one partition needs the data in neighbour partitions. A good road network partitioning solution should maximize the benefits and minimize the costs at the same time.

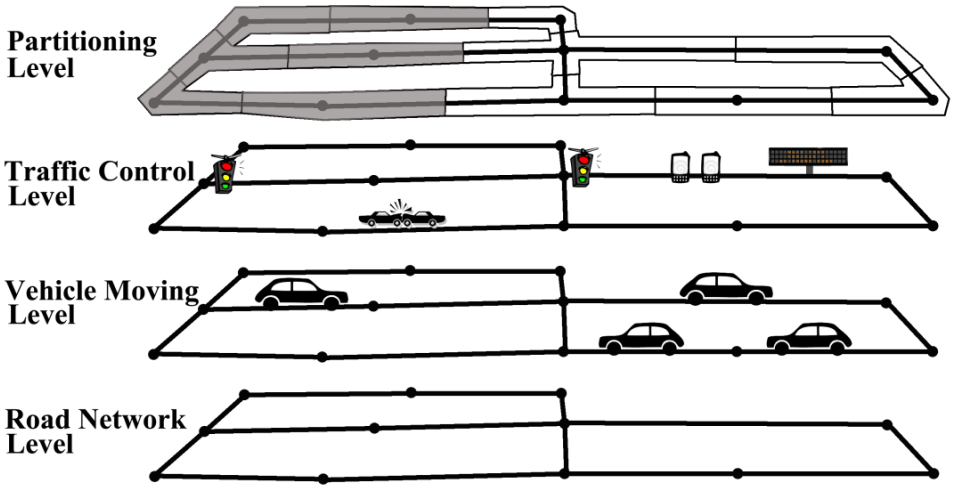
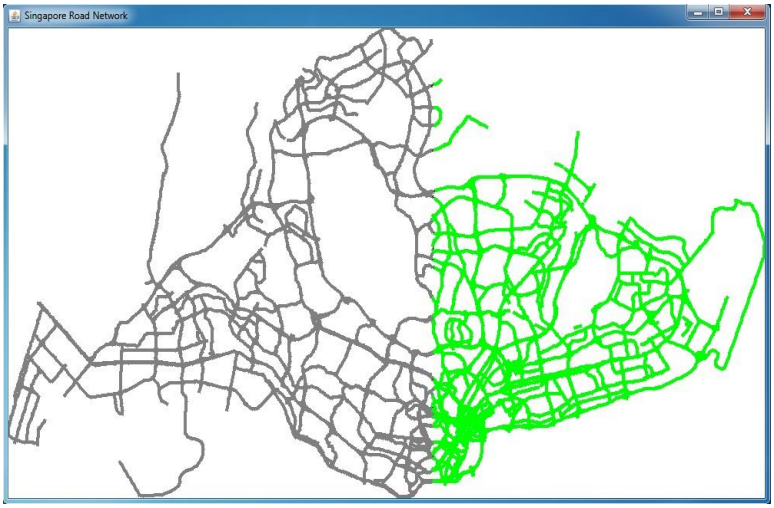


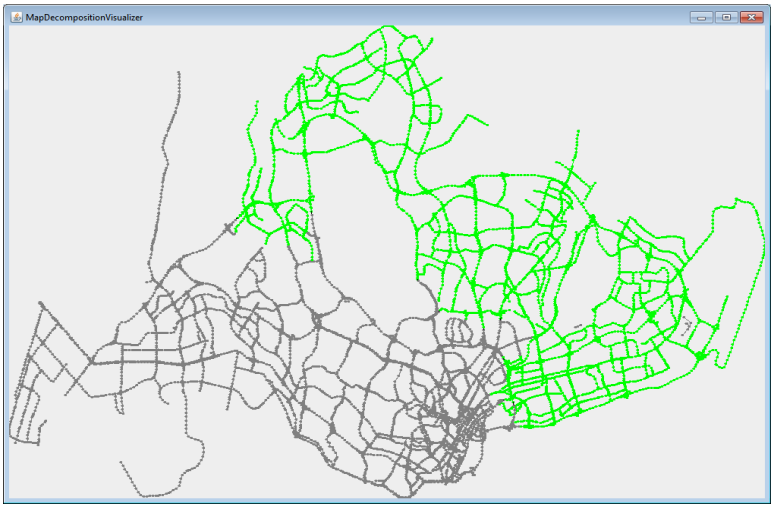
Figure 7: Four layers in map decomposition algorithms

While getting an optimal map decomposition solution is difficult, there are three algorithms that work pretty well in most cases: the orthogonal recursive bisection algorithm, METIS algorithms (Karypis et al., 1999; LaSalle et al., 2013), and hMETIS algorithms (Karypis et al., 1999; Xu et al., 2012). In the orthogonal recursive bisection algorithm, a road network is divided into a number of cells and each cell is centered by one node and cuts its neighbor links. Each node gets a weight corresponding to the workload of all of its attached half-links. Nodes are located at their geographical coordinates. A vertical or horizontal straight line is searched so that roughly half of the computational load is on each half. Then the larger of the two pieces is picked and cut again, by a vertical or horizontal line. This is recursively done until a sufficient number of partitions are generated. The major benefit of the orthogonal recursive bisection algorithm is its simplicity. However, it has three weaknesses. First, the orthogonal recursive bisection algorithm is not efficient for partitions which are not a power of 2. Second, the workloads of nodes are inaccurate, which deteriorates the load balance when the number of partitions is large (e.g. >10). Third, the orthogonal recursive bisection algorithm does not try to minimize the communication cost between partitions. An example map decomposition of the Singapore network using the orthogonal recursive bisection algorithm is shown in Figure 8(A). METIS algorithms, and hMETIS algorithms are a group of advanced and complex algorithms, compared to the orthogonal recursive bisection algorithm. Similar to the orthogonal recursive bisection algorithm, a road network is divided into a number of cells and each cell is centered by one node and cuts its neighbor links in the middle. However, the output of METIS algorithms and

hMETIS algorithms are a mapping table between node IDs and partition IDs, instead of vertical or horizontal lines in the orthogonal recursive bisection algorithm. An example map decomposition of the Singapore network using hMETIS algorithms is shown in Figure 8(B). As shown in Figure 8, hMETIS algorithms succeed to avoid cutting the Singapore road network through the city center. METIS algorithms can always quickly produce high-quality partitions for city-scale road networks. However, the limitation is that it does not guarantee that the partitions are contiguous, even though in most cases they will be.



(A) An example map decomposition using the orthogonal recursive bisection algorithm



(B) An example map decomposition using hMETIS algorithms

Figure 8: Map decomposition of the Singapore network using two algorithms

Workload Estimation Algorithms

Network decomposition algorithms take weights of links, nodes, or both as inputs. To obtain the best results, the weights should accurately represent the actual workload associated with each link and/or node. However, it is generally difficult to get precise measurements of workloads. Even if one can break down the simulation into low level machine instructions, and compute exactly how many instructions are used to move individual vehicles at all circumstances, the actual computation time is still unknown. This results from factors such as the cache and the branch prediction technology, which are common to modern processors.

There are two practical methods to estimate workloads in a traffic scenario: agent-based estimation and network-size based estimation. In agent-based workload estimation, the workload of simulating a link is assumed to be linear with the number of agents (e.g. vehicles) on the link. (Nagel at al., 2001; Wen, 2009) give evidence that the number of vehicles on a link is a good indicator of the computational cost of the link. The number of vehicles on a link is commonly estimated by multiplying the historical road density and the length of the link. In network-size based workload estimation, the workload of simulating a link is assumed to be linear with the total lane length of the link. The second method is simpler. However, it is effective only if the workload is evenly distributed on all lanes in the road network.

When simulating a traffic scenario with a long period, the distribution of workloads on the road network might change significantly during the traffic scenario. Thus, the simulation period should be divided into multiple periods (e.g. AM-Peak, Non-peak, PM-Peak and Mid-Night). A separate set of workloads are estimated for each period, and the corresponding workloads are used to generate network partitions for that period. Wen, (2009) proposed an adaptive workload estimation method, which changes the network partition periodically (every 15 minutes), and concluded that adaptively changing the partitions on-the-fly significantly improved the load balancing especially when unusual events occur.

Boundary Processing Method

In parallel & distributed traffic simulations, multiple network partitions are simultaneously simulated. One key problem is that objects in different partitions should be visible to each other. It means a vehicle in one partition should be able to access nearby objects, even if the objects are

in another partition. Besides, if a vehicle moves from one partition to another partition, all information of the vehicle should be transferred automatically. As shown in Figure 9, a road network is divided into two partitions. In this example, the road network is cut by a line crossing the middle of links. The left side is named the upstream partition and the right side is named the downstream partition. The boundary area (the red dash links) is the area between these two partitions, which should be "seen" by agents in both sides. The size of the boundary area depends on the looking forward distance (e.g. in car-following models), the looking backward distance (e.g. in lane changing models) and the synchronization frequency (e.g. 2 time steps).

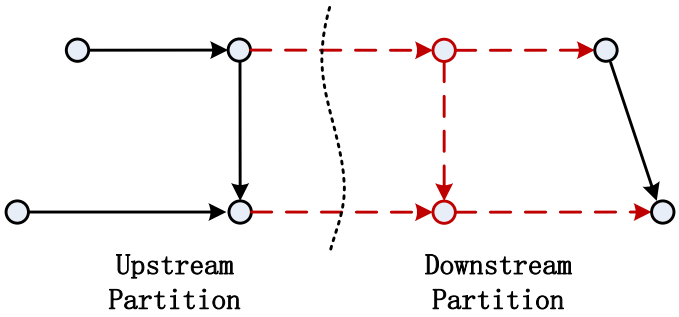


Figure 9: The boundary area between two partitions

There are three types of communication between network partitions: forward ownership transfer, backward proxy transfer and global information interchange. First, when a vehicle crosses a cut line (e.g. the dash line in Figure 9), the ownership of the vehicle is transferred from the upstream partition to the downstream partition. All information of the vehicle is transferred. Second, when a vehicle needs to access the information of another vehicle, which is located in another partition, the information of that vehicle will be backward transferred to the partition. However, only a few information (e.g. speed and location) of the vehicle is transferred. Third, the global information (like: link-based travel time) is exchanged between partitions periodically.

Boundary processing forces processors to stop to synchronize and communicate, which makes the system vulnerable to load imbalance. The faster processors have to wait until the slower ones finish their work. Wen, (2009) investigated ways to reduce the frequency of boundary processing. First, the size of the boundary zone is enlarged, in order that the upstream partition knows more traffic conditions on the downstream partition. Second, since boundary processing does not happen at each time step, vehicles on the upstream partition are simulated

without a perfect knowledge of the downstream partition. A feedback method is designed to reduce the bias. However, unless the boundary processing takes place at each time-step, the result could be inconsistent with the sequential simulation, where perfect knowledge about the previous time-step is available before moving any vehicle in the current time-step.

Emerging Hardware

Traditionally, traffic simulations were written to be executed on single-core hardware. However, the computational performance of the CPU is increasing slowly during the last decades. At the same time, parallelism appears to be a sustainable way of increasing computational performance.

PARAMICS is one of the earliest parallel traffic simulators. As explained in (Gordon et al., 1996), the PARAMICS project is originally designed for a Thinking Machines CM-200, a 16-K processor SIMD machine. In the SIMD model, performance is gained by exploiting the very high number of simple processors that are connected in a tightly coupled, high speed network. The important thing is that each processor on the machine is executing the same piece of code. The authors describe that, to make good use of the CM-200, the data must be in a parallel array form so that operations can occur in parallel on the elements of the array. The approach used to build a parallel data framework for the simulation process was to associate a number of queues with each of these unidirectional links. A queue was used as the parallel item of data on the CM-200. The parallel array is a one-dimensional array consisting of a large number of these queues, and can be operated concurrently. The concepts in this pioneering work are also applied in recent works on traffic simulations on GPUs. AIMSUN (Advanced Interactive Microscopic Simulator for Urban and Non-Urban Networks) is another pioneering parallel traffic simulation program, which was originally developed as a sequential simulator but later ported to parallel computers (Barcelo et al., 1998). A road network is divided into blocks and layers. Firstly, entities that are updated together are grouped into blocks that may be allocated to a single thread. Blocks are grouped into layers, in order that blocks in the same layer can be simulated simultaneously. Thus, different layers are simulated in a sequential order, while within a layer, multiple threads can be started simultaneously.

Computer clusters, which consist of a set of loosely/tightly connected computers that work together, are also receiving attention in this decade. Wen, (2009) investgated to run a mesoscopic

traffic simulation in a distributed way. First, algorithmic analyses are first used to identify the system bottlenecks in a serialized large-scale traffic simulation. Then, scalable approaches are developed to solve the bottlenecks. Distributed traffic simulation with an adaptive network decomposition framework is proposed to achieve better load-balancing and improved efficiency on a computer cluster. Besides, a synchronization-feedback mechanism is designed to ensure the consistency of traffic dynamics across computers while keeping communication overheads minimal. Cetin, (2005) studied the parallel queue model, which includes domain decomposition, message exchanging and communication. A parallel queue model was used to implement distributed event-driven traffic simulations (e.g. MATSIM). Another branch of research work, which does not require modifying the source code of current traffic simulations, is studied in (Lee et al., 2002; Liu et al., 2004). They both presented a general distributed traffic simulation architecture. It involves dividing the network into regions and simulating each region under a separate instance of the program (e.g. PARAMICS). Inter-process communication techniques are employed to exchange data between different regions and synchronize time among the different regions. By using the API supported by off-the-shelf PARAMICS software, the computational load of microscopic simulation is distributed to multiple single-processor PCs without accessing the proprietary source codes of the simulation program. Besides, cloud-based traffic simulation (like: Cube Cloud) might greatly reduce the cost (both time and money) of deploying a city-scale traffic simulation, and encouranges collaboration between teams in different regions and countries. Cloud-based traffic simulations can be viewed as traffic simulations on a large loosely connected cluster.

Graphics processing unit (GPU) is gaining popularity, because of its massive performance compared to the CPU. While GPUs were primarily meant to do three-dimensional rendering in graphics applications, rapid developments in their architectures have enabled their use in scientific computing (Michalakes et al., 2008), computational finance (Buck et al., 2007), computational biology, simulations (Denis et al., 2012; Park et al., 2011) and high performance computing (Fan et al., 2004). Moreover, the development of direct computing application protocol interfaces (APIs), such as CUDA and OpenCL, has greatly reduced the programming effort. However, fundamental differences in the GPU and the CPU architectures mean that the traditional technique of converting serial implementations to parallel using standards such as

OpenMP and MPI is inapplicable. Thus, a research question is whether the GPU can be a potential high-performance platform for traffic simulations. There has been some research work to enhance the solution to traffic simulation on GPUs. Perumalla et al., (2009) introduced a method to simulate the vehicle movement on GPU by using a field based model. This model maps the real world road data onto a 2D lattice, with each element in lattice representing the possibility of turning either left/right or up/down. By using this possibility data, the vehicles will be directed from one position in the 2D array to another position. The proposed field based model is similar to the classic Cellular Automata Model. However, the contribution of this work in the global traffic simulation research framework is not clarified. The MATSIM team recently released a research work to implement an event-driven mesoscopic traffic simulation framework on the GPU (Strippgen et al., 2009). The paper introduced two kernel functions ("moveLink" and "moveNode") to implement the core queue simulation. They also talked about three different implementations of the vehicle array in the GPU memory. Their work is pioneering and obtained a speedup over serial applications between 5.5 and 60 times depending on different data structures and NVIDIA GPU series. However, the paper did not confirm the correctness of running MATSIM on the GPU and also did not explain how to migrate a real-world mesoscopic traffic simulation from the CPU to the GPU.

2.4 Calibration of City-scale Traffic Simulations

The calibration of a traffic simulator is essentially a process of determining variables (e.g. model parameters and model inputs) in order to minimize the difference between simulated outputs and observed measurements. The procedure of calibrating a city-scale traffic simulator consists of two phases: disaggregate calibration and aggregate calibration. In the disaggregate calibration phase, parameters of a particular model (e.g. the route choice model) or a particular model input (e.g. the OD matrix) are calibrated using specified types of observed measurements. In the aggregate calibration phase, different types of model parameters and model inputs are jointly calibrated using mixed types of observed measurements.

2.4.1 Calibration Variables

Calibration variables are divided into variables in demand models and variables in supply models. The main calibration variables in demand models are the time-dependent OD matrix and parameters in the travel behavior models. As explained in Section 2.1.2, an OD matrix is a 2-D table, which indicates the number of vehicles to generate for each origin-destination pair during a period in a traffic simulation. Calibration variables in travel behavior models include parameters in choice set generation algorithms, utility functions (e.g. the weights of time and money in the utility function) and choice making functions (e.g. parameters in Path Size Logit model). The calibration variables in supply models depend on the used traffic models (e.g. mesoscopic, microscopic, macroscopic and nanoscopic). In microscopic traffic simulations, supply-side calibration focuses on parameters in car-following models, lane-changing models and gap acceptance models (these three models were introduced in Section 2.2.2). In mesoscopic and macroscopic traffic simulations, supply-side calibration focuses on segments' capacity and parameters in the speed-density relationship (they were introduced in Section 2.2.1). Finally, a traffic simulation tends to have specific calibration variables, such as the length of vehicle, the default vehicles' speed in queues, the default turning speed, reaction time, etc.

There are three major types of data used in the calibration of a traffic simulator: survey data (e.g. Household Travel Survey (Cheong et al., 2008) in countries like Singapore and Australia), point-based surveillance data (e.g. loop detectors and cameras) and route-based trajectory data (e.g. on-board units and GPS devices). In this thesis, point-based surveillance data (segment-based traffic flow) in the Singapore expressway network is the major data source for calibration.

2.4.2 Disaggregate Calibration

In the disaggregate calibration phrase, the OD matrix, parameters in travelers' behavior models, and parameters in supply-side models are calibrated separately. Travel survey is a common and effective way to give prior values of an OD matrix. However, travel survey has limitations due to its high cost of data gathering, low population coverage and limited accuracy. A high fidelity OD matrix is often obtained by aggregate calibration. Currently, a direct city-scale measurement of travelers' departure times is missing. A practical solution is to assume that travelers' departure times follow a Normal distribution or a Poisson distribution. Parameters in the distribution are

estimated based on field studies in small regions. Disaggregate data that reveal individuals' route choice behaviors are used to estimate parameters in route choice models. The sources of data include traditional surveys, such as mail, telephone, and the Internet (Ben-Akiva et al., 1984; Prato,2004), or emerging technologies, such as GPS trajectories (Frejinger, 2007). The parameters to estimate are the weights in the systematic utility function and they are estimated by maximizing the probability of the actual selected route in the choice set. More rigorous mathematical derivations in discrete choice analysis can be found in (Ben-Akiva et al., 1985).

The OD estimation studies can be divided into two groups: static OD estimation and time-dependent OD estimation. In the static OD estimation, it assumes that the OD flows are static across a time period (e.g. the AM peak hours). Example studies on static OD estimation include (Cascetta, 1984), (Hazelton. 2000), and (Li, 2005). This thesis focuses on time-dependent OD estimation, where the OD flows may change significantly across different time periods. The most widely applied dynamic OD estimation approach is the generalized least square (GLS) framework proposed in (Cascetta et al. 1993; Cascetta et al. 2013). The OD estimation problem is expressed through a fixed-point model and a programming-based optimization method is proposed to solve it. The authors propose two methods under the GLS framework. The sequential method solves for the OD flows one interval at a time and the simultaneous method optimizes the OD flows in multiple intervals simultaneously. Although the simultaneous method is able to capture the influence of the OD flow in one interval on all subsequent intervals, it requires the calculation and storage of a large assignment matrix and has been found to have significant computational overhead on large networks in (Toledo et al., 2003) and (Bierlaire et al., 2004). Then, Ashok (1996) formulates the OD flow estimation problem with a state-space representation, where the interval-to-interval evolution of network states is captured by transition equations (autoregressive process) and the sensor counts are incorporated using measurement equations that links states with measurements. Then, the author further proposes a Kalman Filter solution approach. In both the GLS and the state-space framework, the assignment matrix is a key element for OD flow estimation. There are two approaches to obtain assignment matrices outlined in Ashok (1996): 1) load the current best OD flows into a traffic simulator, keep tracking the vehicles from different OD pairs, record and count their appearances at sensor locations, and 2) analytically calculate the fractions using knowledge of network topology, time-

dependent travel times as well as route choice models. Balakrishna (2006) argues that the latter approach can provide more accurate results because the simulation based method requires small starting flows to prevent artificial bottlenecks and therefore results in highly stochastic and small fractions due to the limited number of vehicles being loaded into the network.

Recent researchers proposed a number of new OD calibration methods. (Nie et al., 2008) formulates the OD estimation problem as a single-level framework based on variational inequality. Travel time is incorporated into this framework by (Qian et al., 2011) to further improve its performance. (Djukic et al., 2012) applies principal component analysis (PCA) to pre-process OD data. This methodology can significantly reduce the size of time series of OD demand without sacrificing much of the accuracy, which leads to an impressive reduction of computational costs. There are also attempts to formulate the OD estimation problem as a stochastic optimization problem and use meta-heuristic approaches as the solution algorithms. For example, (Stathopoulos et al., 2004) experiments with three different meta-heuristic optimization algorithms: a genetic algorithm (GA), a simulated annealing algorithm (SA), and a hybrid algorithm (GASA) based on GA and SA. The authors argue that GASA outperforms the other two algorithms in terms of convergence rate and final goodness-of-fit; (Kattan et al., 2006) applies evolutionary algorithms (EA) in a parallel computing framework to improve the computation efficiency as well as the solution quality.

In mesoscopic and macroscopic traffic models, capacities are mostly estimated using observed flows. For example, in the approach proposed by (Mufioz et al., 2004) to estimate the parameters in a Cell Transmission Model (CTM), capacities for non-bottleneck cells are chosen to be slightly larger than the maximum observed flows while capacities for bottleneck cells are computed to match the observed mainline and ramp flows. The Highway Capacity Manual has also been widely used as a reference for determining the capacities for different types of links. Parameters in the speed-density relationship are estimated by fitting appropriate curves to observed traffic data. Due to the usually low level of sensor coverage rate in reality, (Van Aerde et al., 1995) proposes an approach to group links in the network based on their characteristics.

In microscopic traffic models, parameters in driving behavior models are estimated based on surveys and historical projects in similar cities. (Darda, 2002) applies the Box-Complex (Box, 1965) algorithm to calibrate car-following and lane-changing models in MITSIMLab given fixed

demand model parameters. However, convergence is not able to be ascertained in this study. (Brockfeld et al., 2005) calibrates a small number of supply parameters in a variety of microscopic and macroscopic simulation systems using a gradient-free downhill algorithm. (Kim et al., 2004) applies genetic algorithms (GA) to calibrate driving behavior parameters in CORSIM and TRANSIMS. However, given the small scale of the problem in their studies, there are still computational constraints. More studies on applying these methods on real-world scale networks should be done to test their efficiency and ability to handle city-scale problems.

2.4.3 Aggregate Calibration

There are two types of aggregate calibration methods: iterative demand-supply joint calibration and simultaneous demand-supply joint calibration. In the first method, demand calibration and supply calibration are done iteratively until convergence is reached with the observed values. In demand calibration, the parameter values in the supply model are fixed when OD flows and parameters in behavior models are calibrated using aggregate traffic data, for example, sensor counts, travel times, etc. Then, using the calibrated demand parameters, supply models are calibrated and simulated traffic conditions that are used in the calibration of demand models are updated, for example, the weight of time delay in route choice models. In the second method, demand calibration and supply calibration are done simultaneously. The second method has received significant attention due to its ability to fully consider the interactions between demand models and supply models as well as their huge savings in computation time. This thesis focuses on simultaneous demand-supply joint calibration.

The simultaneous demand-supply joint calibration is modeled as an optimization problem. The objective function is to maximize the goodness-of-fit of simulated traffic quantities to their observed values. The objective function is explained in Section 6.1. This stochastic optimization framework was first proposed by (Balakrishna, 2006). The author applied this simultaneous demand-supply calibration methodology in DynaMIT, a mesoscopic simulation simulator. Parameters considered in the decision vector included time-dependent OD flows, parameters in travel behavior models, parameters in speed-density functions, and segment capacities. Goodness-of-fit to sensor flows and segment speed data are used in the objective function. As in most cases, the number of measurements is far less than the number of parameters to estimate,

the distances between estimated parameter values and their priori, or historical values are also included as a part of the objective function. In this work, three algorithms were tested as the solution algorithm of the stochastic optimization problem: the Box-Complex (Box, 1965), stable noisy optimization by branch and fit (SNOBFIT) (Huyer et al., 2008), and simultaneous perturbation stochastic approximation (SPSA) (Spall, 1998). A synthetic case study and a full scale real world case study in Los Angeles, California were used to test the simultaneous demand-supply calibration framework as well as the three different solution algorithms. The results showed that SPSA was the most suitable algorithm due to its extremely high efficiency and satisfactory convergence performance. Successful attempts have been made to modify or extend the existing SPSA algorithm to improve its performance. Balakrishna et al., (2007) extended this methodology to the off-line calibration of microscopic traffic simulation models by applying it in MITSIMLab; Cipriani et al. (2011) proposed asymmetric estimation and adopting polynomial interpolation to the step size in the SPSA algorithm; Cantelmo et al. (2014) proposed a second order SPSA algorithm to deal with difficulties in optimizing functions with variables of different magnitude orders.

Appiah et al., (2010) presents a framework that jointly calibrates OD flows and driving behaviors in microscopic supply simulators using aggregate intersection turning movement counts. Similar to (Balakrishna, 2006), the framework formulates the calibration as an optimization problem. However, a GA was adopted as the solution algorithm. A case study in a small arterial network using VISSIM, a microscopic simulation software package was conducted in this study. The results showed appropriate final goodness-of-fit but the GA algorithm showed high computational cost as it took the algorithm 2 months and 18,000 iterations to converge, given the small scale of the problem. Ben-Akiva et al., (2012) applied the simultaneous calibration methodology to a city-scale highly congested urban network in the city of Beijing. The network consists of 1,698 nodes connected by 3,180 directed links in an area of approximately 35 square miles. There are as many as 3000 OD pairs in the study network. Time dependent OD flows, route choice parameters, speed-density relationships and segment capacities were calibrated. Sensor counts in expressways and travel times from GPS equipped floating cars were used as measurements in the calibration. However, the experiment results showed room for improvement in terms of the goodness-of-fit to observed values.

3. ETSF: An Entry-Time based Supply Framework for City-scale Traffic Simulation

The computational complexity of mesoscopic traffic simulations is directly proportional to the number of vehicles in the simulated traffic scenario (Wen, 2009). It makes simulating congested traffic scenarios much more costly than simulating uncongested traffic scenario. To tackle the problem of high computational cost when simulating congested traffic scenarios, we proposed a novel simulation supply framework named 'Entry Time based Supply Framework (ETSF)'. ETSF has lower a computational complexity than the current mesoscopic supply framework and more importantly, the computational complexity of ETSF is not sensitive to the number of vehicles in the simulated traffic scenario (Xu et al., 2014). This chapter firstly introduces the key concepts and the simulation procedure in ETSF. Then, the trade-off in ETSF is discussed. Finally, ETSF is investigated and evaluated in an artificial road network and a real-world city-scale road network.

3.1 Key Concepts in ETSF

Entry Time based Supply Framework (ETSF) is an extension of time-stepped mesoscopic traffic framework. Similar to the mesoscopic traffic simulation framework (in Section 2.2.3), a road network in ETSF is modeled as nodes, links, segments and lanes. Figure 10 shows an example of vehicles moving in a lane in ETSF. The key idea is to model vehicles as a traffic fluid, when vehicles stay in the lane, and as individual vehicles, only when vehicles need to change lanes. Key concepts include: a speed table of a lane, entry time of a vehicle, the accumulated movement distance $D_{acm}(t)$, entry_time_to_pass of a lane (t_p) and entry_time_to_queue of a lane (t_q).

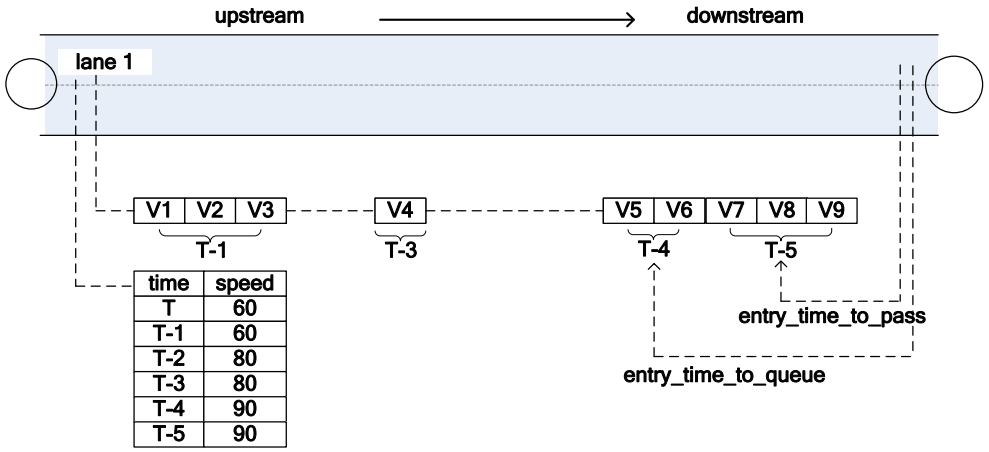


Figure 10: An example of vehicles in a lane of a link in the ETSF

First, each lane of the link has a speed table, which contains the speed of the lane at recent time steps. In a time step, all vehicles on the lane move using the corresponding speed in the table. For instance, at simulation time T, vehicles' speeds are 60 km/h. Second, each vehicle on the lane has an attribute: entry_time, which refers to the simulation time when the vehicle enters the lane. Third, given the speed table of a lane and the entry time of a vehicle on the lane, the accumulated movement distance (or offset) of the vehicle is calculated by:

$$D_{acm}(t') = (\sum_{\lceil t' \rceil \le t \le T} v(t) * \Delta t) + (v(t') * (\lceil t' \rceil - t')) \quad (3.1)$$

where, *t'* is the entry time of the vehicle, $\lceil t' \rceil$ is the following time step after the vehicle enters the lane, *T* is the current time, Δt is the simulation time step, and *v(t)* is the speed of the lane at time *t*. The first half represents the movement distance between $\lceil t' \rceil$ and T; The second half represents the movement distance during the time step when the vehicle enters the lane. Fourth, vehicles in a lane are ordered based on their entry time. Since all vehicles are moving using the same speed, the vehicle order is not changed. Besides, lane changing behaviours are not allowed in ETSF. Fifth, each lane has a key attribute: entry_time_to_pass (t_p), which means that "*if the entry time of a vehicle is earlier (or smaller) than t_p, its accumulated movement distance is bigger than the length of the lane and then the vehicle has the potential to pass the current lane"*. It is used to determine whether a vehicle can pass to the next link. The accumulated movement distance at time t_p is equal to the length of the lane. For example, in Figure 10 the current time is T and t_p on the lane is T-5. It means that vehicle V9 might pass to the next link if other conditions are also

satisfied (these other conditions are explained later). Sixth, each lane has another key attribute: entry_time_to_queue (t_q), which means that "*if the entry time of a vehicle is earlier (or smaller) than t_q, the vehicle either passes the lane or enters the queue*". It is used to determine whether a vehicle is in a queue. The accumulated movement distance at time t_q is equal to the length of the non-queue part of the lane. For example, in Figure 10 the current time is T and t_q on the lane is T-4. It means that vehicles 5-9 are in the queue. Note that the accumulated movement distance of V5 is smaller than the lane's length, but V5 is in the queue, because it reaches the end of the queue.

Figure 11 further explains the concept: entry_time_to_pass of a lane. The x-axis corresponds to time (but in a reversed direction), and the y-axis corresponds to the accumulated moving distance of a vehicle. The current time step is T. At each time step t, there is a speed v(t) on the lane, which is also the slope during the time step in Figure 11. The height of the gray area is equal to the accumulated movement distance, which is also the length of the lane. The time in the right border of the gray area is named as the entry_time_to_pass of a lane at time T. Vehicles (e.g. V9), whose entry time is in the right side of (or smaller than) t_p, might be able to pass the lane.

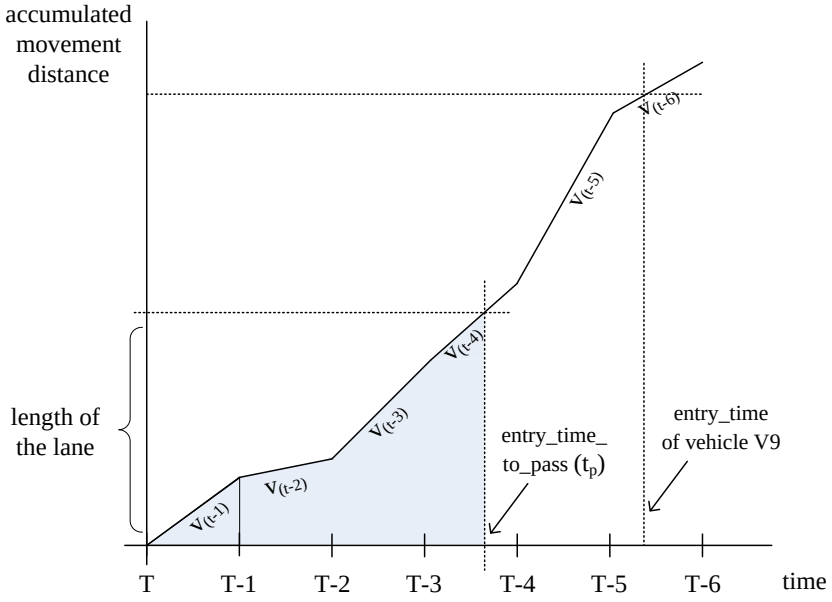


Figure 11: The entry_time_to_pass (t_p) of a lane

3.2 The Simulation Procedure

The simulation procedure in ETSF is shown in Table 6. The simulation procedure consists of 4 layers of loops. First, ETSF has a loop of simulation time steps. Second, during a time step, ETSF has a loop of simulating each link. Third, each link contains multiple lanes. Fourth, ETSF simulates the traffic dynamics in each lane. These 4 layers of loops are similar to the current mesoscopic simulation framework (see Table 4). Lanes are basic processing units in ETSF. This section explains the calculations of t_p and t_q of a lane, queue length and empty space of a lane, which are unique in ETSF.

Table 6: The simulation procedure in Entry Time based Supply Framework (ETSF)

Inputs: A road network, a traffic scenario, parameters of flow models	
Outputs: Aggregate road measurements (e.g. density and queue) at each time step;	
1	Initialize a macroscopic traffic simulation environment
2	***for*** each time step ***do***
3	… load new vehicles;
4	… ***for*** each link in the network ***do***
5	… ... ***for*** each lane in the link ***do***
6	… ***... ...*** calculate density and speed of the lane and then update speed table;
7	… ... ... calculate entry_time_to_pass (t_p);
8	… ... ... ***for*** i ← 1 to the output capacity ***do*** *//only few vehicle in a lane*
9	… ... ... reversely check vehicle passing from the downstream end;
10	… ... ... ***end for***
11	… ... ... update entry_time_to_queue (t_q) and the queue length;
12	… ... ... update empty space of the lane;
13	… ... ***end for*** // *loop lanes*
14	… ***end for*** // *link loop*
15	… output simulated traffic condition at this time step
16	***end for*** // *time loop*

The speed of a lane is calculated using the speed-density relationship, which was explained in Section 2.2.1. After that, the entry_time_to_pass (t_p) of a lane is updated in Step 7. When the simulation starts, the entry_time_to_pass (t_p) of a lane is initialized to be invalid (e.g. -1), as the accumulated movement distance is 0. When the simulation time advances, the accumulated movement distance at time t_p increases and becomes larger than the lane's length. Then, t_p is increased, so that the accumulated movement distance at t_p is equal to the lane's length. Continuing the example in Figure 11, Figure 12 shows the update of the entry_time_to_pass (t_p)

of a lane. As the simulation time goes from T to T+1, t_p is updated that the distance between $D_{acm}(t_p^T)$ and $D_{acm}(t_p^{T+1})$ is equal to the movement distance during [T, T+1]. There is another way to understand the procedure of updating the entry_time_to_pass (t_p) of a lane. In the beginning, when the traffic simulation starts, since no vehicle can pass the lane, t_p is invalid (e.g. -1). In the end, when the traffic simulation ends, most vehicles are able to pass the lane (if not blocked), it means t_p is near to the end of the simulation period. During the simulation period, the procedure of updating the entry_time_to_pass (t_p) of a lane is just to check whether t_p can be increased to allow more vehicles on the lane to pass.

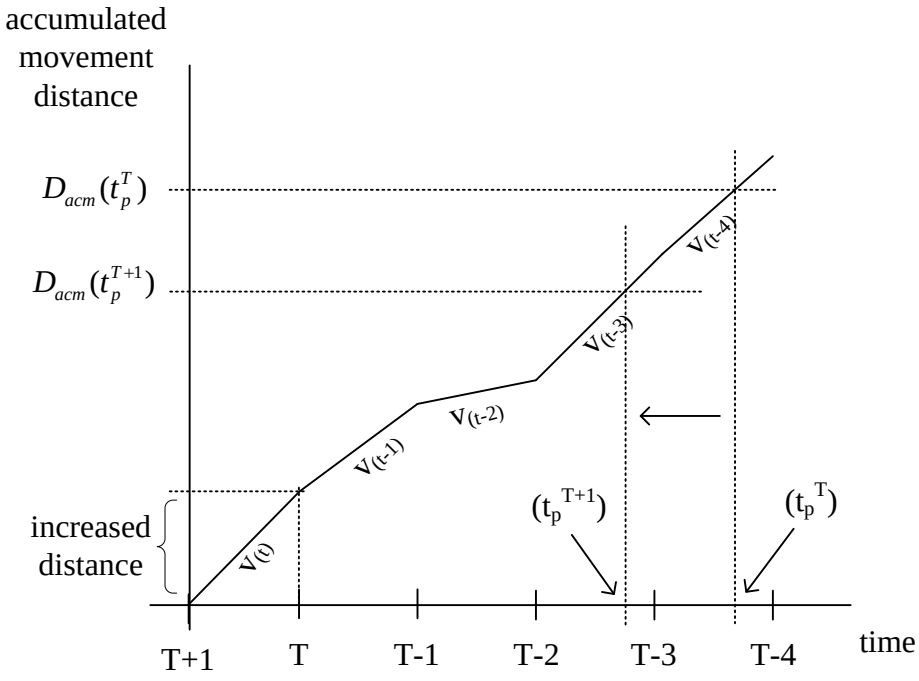


Figure 12: The update of the entry_time_to_pass (t_p) of a lane

Steps 8-10 explain how to pass a vehicle from a lane to the next lane. First, vehicles in a lane are checked in a reverse order, from the downstream end to the upstream end. It is because the behavior of an upstream vehicle depends on whether downstream vehicles can pass the current lane or not. Second, the maximum number of passing vehicles is the output capacity of the lane. Thus, only a smaller number of downstream vehicles are checked during a time step. For example, when simulating a freeway lane, whose lane capacity is 1800 v/h, the theoretical maximum number of passing vehicles during a time step (e.g. 2 seconds) is only 1. It means only

the most downstream vehicle is checked during a time step. Third, as shown in Figure 13, there are four rules to determine whether a vehicle can pass a lane:

1) There is available output capacity in the current lane.
2) There is available input capacity in the next lane.
3) There is available empty space in the next lane.
4) The vehicle's entry time is smaller than the entry_time_to_pass (t_p) of the lane.

Fourth, if one vehicle cannot pass the current lane, all upstream vehicles are blocked. Thus, there is no need to check the upstream vehicles. Fifth, if multiple vehicles can pass lanes and go to the same next lane at the same time, the vehicle with the maximum waiting time, which is (entry time - t_p), is processed first. Finally, In ETSF, a vehicle's status (e.g. entry time and next lane) is updated only if the vehicle passes from the current lane to the next. Compared to updating each individual vehicle's status (e.g. locations) at each time step in the current mesoscopic supply framework, ETSF tends to be more time efficient and less sensitive to the level of congestion.

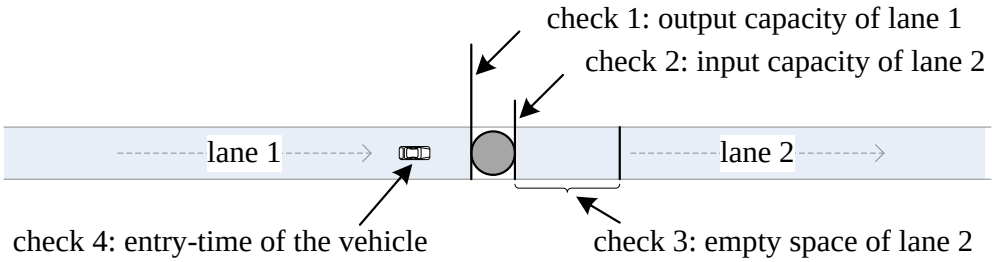


Figure 13: Four rules to determine whether a vehicle can pass a lane

Step 11 is the calculation of the entry_time_to_queue (t_q) and the queue length of a lane. The accumulated movement distance at time t_q is equal to the length of the non-queue part of the lane. Thus, if there is no queue on the lane, $t_q = t_p$; if there is a queue on the lane, $t_q > t_p$. In the queue's expansion phase, the first upstream vehicle, which is not in the queue, is checked whether the vehicle can reach the end of the queue. If yes, t_q of the lane is updated to be the entry time of the vehicle and the queue length is increased by the length of the vehicle. Otherwise, t_q of the lane and the queue length are not changed. In the queue's shrink phase (when vehicles in the queue pass to the next lane), t_q of the lane is not changed and the queue length is reduced by the length of the vehicle.

During a traffic simulation, the entry_time_to_pass (t_p) of a lane is increased from 0 to the end time step, thus, the amortized update cost at a time step is $\theta(1)$. Second, the number of simulated individual vehicles at a time step is similar to the flow of the lane during the time step, and constrained by the output capacity of the lane. Third, the calculation of t_q and the queue length of a lane require one to to scan the vehicles on the lane. However, each vehicle is checked only once on the lane. Thus, the aggregate cost is actually linear to the aggregate flow of the lane. In summary, the time complexity of ETSF framework is:

$$\theta(TT/\Delta t3 * \mathrm{NLi} * \mathrm{NLa} * (C_6 + C_7 * \bar{f})) \tag{3.2}$$

$$\rightarrow \theta(TT/\Delta t3 * \mathrm{NLi} * \mathrm{NLa} * \bar{f}) \tag{3.3}$$

$$\rightarrow \theta(TT/\Delta t3 * m) \tag{3.4}$$

where, $\Delta t3$ is the simulation time step, NLi is the number of links, NLa is the average number of lanes of each link, $\bar{f}$ is the average number of vehicles passing a lane, C_6 is a constant amortized cost to update the speed, t_p and empty space of the lane, C_7 is a constant amortized cost of passing a vehicle on the lane. $\mathrm{NLi} * \mathrm{NLa} * \bar{f}$ is actually the aggregate number of vehicles passing adjacent links in the road network. If we use m to denote this number, the time complexity becomes $\theta(TT/\Delta t3 * m)$.

Compared with the time complexity of the current mesoscopic simulation framework (Formula 2.11 in Section 2.3.2), the cost of ETSF is not related to the number of vehicles on lanes ($\bar{n}$), but is sensitive to the average number of vehicles passing adjacent lanes ($\bar{f}$). The most important benefit is that the computational complexity of ETSF is not sensitive to the level of congestion in the traffic scenario. Once a vehicle chooses a route, the number of passing adjacent lanes for the vehicle is a fix number. It means that simulating a more congested traffic scenario does not necessarily indicate a longer execution time in ETSF. Moreover, $\bar{f}$ is much smaller than $\bar{n}$ in most cases. For example, given a 1000 meter lane whose output capacity is 1800 vehicle/hour, a time step is 2 seconds and the average occupancy length of a vehicle is 6 meters. Then, the maximum value of $\bar{n}$ is 166.7 while the maximum value of $\bar{f}$ is 1. As shown

in Figure 14, in uncongested traffic scenarios, when the level of demand is much lower than the road capacity, the computational cost of both the ETSF and the current mesoscopic simulation framework are proportional to the level of demand. *Second*, in congested traffic scenarios, the number of vehicles on lanes increases fast because of higher road densities, thus, the time cost of the current mesoscopic simulation framework grows fast. However, the number of vehicles passing a lane is decreased in congested scenarios, thus, the time cost of ETSF is reduced. Thus, in congested traffic scenarios, ETSF tends to be more efficient and more importantly more computational stable than the current mesoscopic simulation framework.

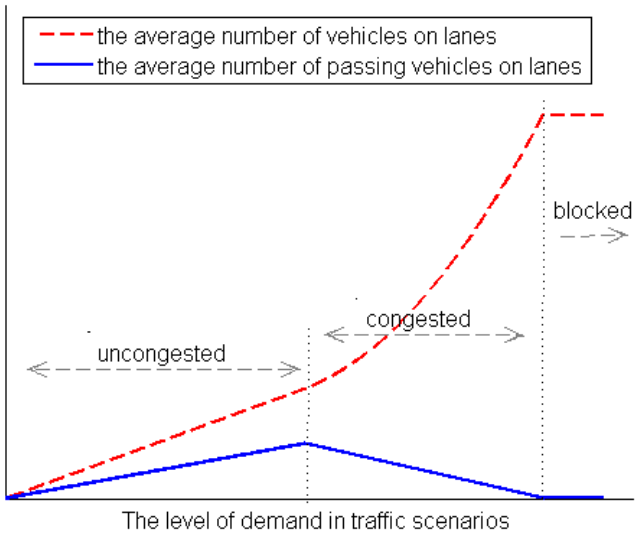


Figure 14: Comparison of *n* and *f* in different levels of traffic demand

3.3 Tradeoff in ETSF

Compared with current time-stepped mesoscopic simulation framework, ETSF makes the computational cost of simulating a traffic scenario less sensitive to the total number of vehicles in the traffic scenario and is proven to significantly reduce the time cost of simulating a congested traffic scenario (in Section 3.4 and Section 3.5). Moreover, there is no loss of accuracy in ETSF because the speed-density model and the queue model are fully supported in ETSF. Compared with event-driven supply framework (Dynameq, Mezzo and MATSIM), ETSF might be inefficient because it has to update the system status time step by time step, although it has

decreased the computing cost in each time step. However, it has better accuracy in computing lane speed and queue length in that it updates them every time step instead of assuming that vehicles have exactly the same speeds after they enter the lane.

The trade-off of ETSF is that positions of individual vehicles are not explicitly calculated in ETSF. Because in many city-scale simulation applications, the aggregate simulated results (e.g. the speed on each link) are much more important than disaggregate simulated results (e.g. the speed of a particular vehicle). However, there are some simulation applications, which require the simulated locations of individual vehicles. In ETSF, a vehicle's position can be calculated as follows. If a vehicle's entry time is later than (or larger than) the tq of the lane, the vehicle is in the moving part of the lane. Then, the vehicle's location is its accumulated movement distance, which is calculated using the Formula 3.1. If a vehicle's entry time is earlier than the tq of the lane, the vehicle is in the queue part of the lane. Then, the vehicle's location depends on its location in the queue. The requirement to calculate individual vehicle's location increases the time cost of ETSF. In an extreme case, if locations of all vehicles at each time step are required, ETSF has no benefits compared to the current mesoscopic simulation framework.

3.4 Synthetic Tests

The effectiveness and sensitivity of ETSF is investigated in a synthetic test, before it is evaluated in a city-scale traffic simulation.

3.4.1 Experimental Design

The experiments were based on a state-of-the-art mesoscopic traffic simulation software, DynaMIT (Dynamic Network Assignment for the Management of Information to Travelers), which employed an experimental road network based on the Singapore expressway road network and some typical traffic scenarios. Its source code was revised to implement ETSF and used to carry out the following four groups of experiments. In this section, the prerequisite 'vehicles on the same lane are moving using the same speed at a time step' is applied in both the current framework and the proposed ETSF framework. So the simulation results (e.g. road-based speed and density) are exactly the same. Thus, this section focuses mainly on the computational

efficiency. The experiments were executed on a computer with a 2.83GHz Intel Core 2 Quad processor, 4GB memory and Ubuntu Linux 12.10. To provide trustworthy results, each result in the following graphs represents the average of 10 runs.

Figure 15 shows the computer representation of the expressway system for Singapore road network. This expressway system consists of expressway links and ramps connecting local roads with the expressway. The network has been modeled using a detailed representation of the length, geometry and lanes of each link. Tailored from the above expressway network (the red thick part), a prototype topology of the road network is illustrated in Figure 15 with bold red links. The purpose is to control the road length and the congestion level in the experiments. The prototype network has 6 nodes and 5 links. Each link has one lane. Each lane has the same length (1000 meters), the same input capacity (1800 vehicles/hour) and the same output capacity (1800 vehicles/hour). As shown in Table 7, there are 4 Origin Destination (OD) pairs and each OD pair has only one path. Vehicles are moving from {node 1 and node 2} to {node 5 and node 6}. Vehicles from node 1 and node 2 are firstly merged at node 3 and then are split to flow to node 5 and node 6. The period of simulation scenario is 10 hours. There are three configurable parameters: the length of a link, the demand level of each OD pair and the simulation time step. The default length of a link is 1000 meters. Different lengths of a link {100, 200, 500, 2000, 5000, 10000 meters} are tested. The default demand level of each OD pair is 300 v/h. It means 300 vehicles are evenly loaded into the simulation for each OD pair during each hour. In this case, the accumulated flow in lane 3 is around 1200 v/h. Different demand level {0, 100, 200, 400, 500 v/h} are tested. The default simulation time step is 2 seconds. Different simulation time steps {4, 6, 8, 10, 12, 14, 16, 18, 20 seconds} are tested.

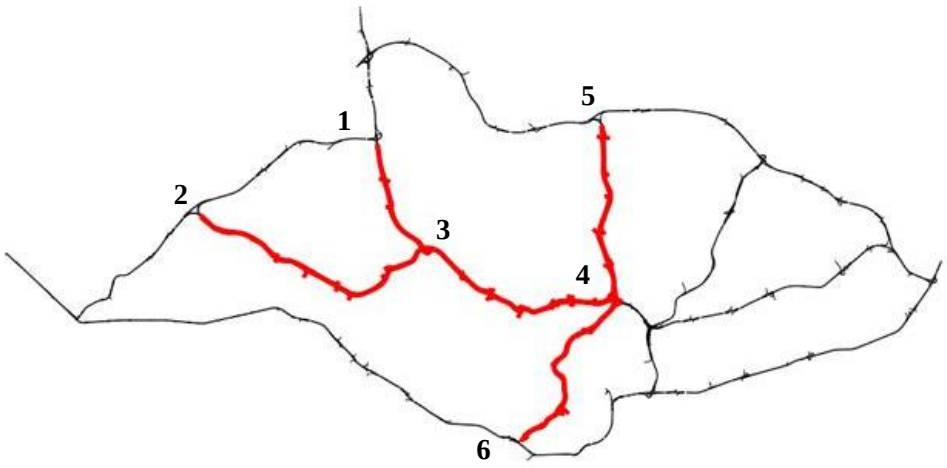


Figure 15: The Singapore expressway and a prototype network (node: 1-6)

Table 7: OD Pairs and Paths in the prototype network

ID	Origin	Destination	Paths
1	1	5	1->3->4->5
2	1	6	1->3->4->6
3	2	5	2->3->4->5
4	2	6	2->3->4->6

3.4.2 Length of Links

This experiment is designed to compare the execution time of the ETSF and current mesoscopic simulation framework with different settings of link lengths. The results are shown in Figure 16. The x-axis corresponds to the length of a link and the y-axis to the execution time of simulating the traffic scenario. The blue solid line represents the ETSF and the red dashed line represents the current supply framework. The time cost of the current mesoscopic simulation framework is sensitive to the length of a link. In fact, as the length of a link increases, the number of vehicles on the link increases and the simulation time linearly increases. On the other hand, as shown in Formula 3.2, the cost of simulating vehicle movement in the ETSF is only related to the flow of the link. Thus, ETSF is not sensitive to the length of a link. When the length of a link is 100 meters, the average number of vehicles on a link in this scenario is around 1.0. The execution times of both frameworks are similar. However, when the length of a link is 5000 meters, the average number of vehicles on a link is around 60. The execution time of the ETSF is only 25% of the execution time of the current framework.

3.4.3 Demand Level

This experiment is designed to compare the execution time of the ETSF and current mesoscopic simulation framework with different settings of demand levels. The results are shown in Figure 17. The x-axis corresponds to the demand level of each OD pair and the y-axis to the execution time of simulating the traffic scenario. First, both supply frameworks are sensitive to the number of vehicles loaded into the scenario. However, the execution time of the ETSF increases much slower than the current supply framework. Second, when the demand level is 0, which means there is no vehicles loaded into the scenario. The execution time of ETSF is slightly worse than the current supply framework, because ETSF needs to update t_p and t_q. Third, when the demand level of each OD pair is 500 v/h, the execution time of the current mesoscopic simulation framework increases faster, while the execution time of ETSF is more stable. It is because when the demand level is 500 v/h, the accumulated demand to enter link {3->4} (2000 v/h) is higher than the input capacity of the link, so congestion happen in link {1->3} and link {2->3}. It shows that the execution time of the current mesoscopic simulation framework is sensitive to network congestion while the execution time of ETSF is insensitive to network congestion.

3.4.4 Simulation Time Step

Both the current mesoscopic simulation framework and the ETSF are sensitive to the setting of the simulation time step. When the simulation time step is small (e.g. 0.5 seconds), $\overline{f}$ will be low and the computational complexity of the ETSF is proportional to the inverse of the simulation time step. Thus, reducing the simulation time step will increase the complexity of the ETSF. When the simulation time step is large, the computational complexity of ETSF cannot be lower than a certain value. The certain value is the accumulated time cost to move vehicles between links, which are not related to the size of the simulation time step.

This experiment is designed to compare the execution time of the ETSF and the current mesoscopic simulation framework with different simulation time steps. The results are shown in Figure 18. The x-axis corresponds to the simulation time step and the y-axis to the execution time of simulating the traffic scenario. First, when the simulation time step increases, the execution times of both supply frameworks decrease because the required frequency to update the simulation system status is reduced. Second, as the simulation time step increases, the benefit

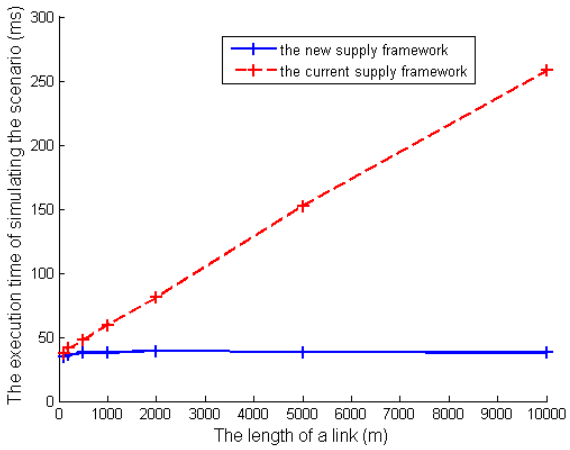


Figure 16: The execution time of ETSF and the length of a link

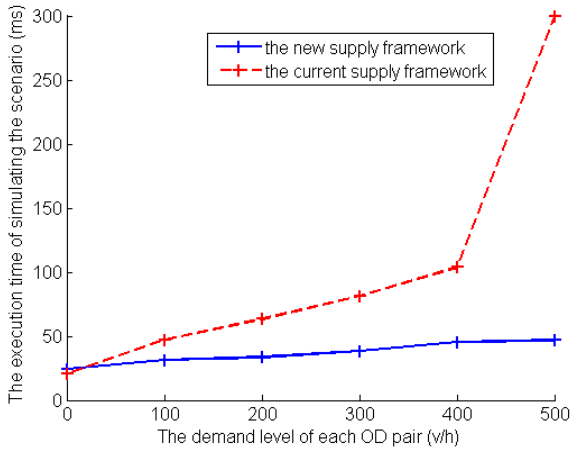


Figure 17: The execution time of ETSF and the demand level

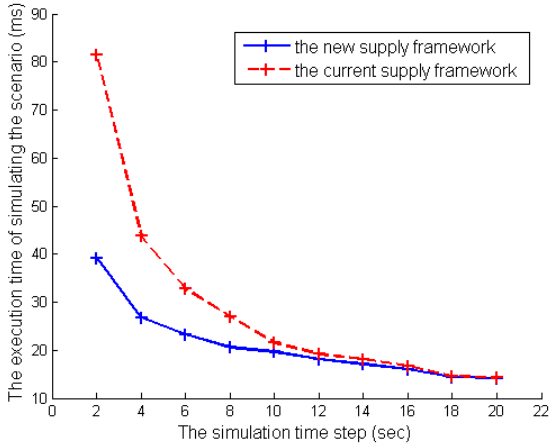


Figure 18: The execution time of ETSF and the simulation time step

of the ETSF becomes smaller. This is because as the simulation time step increases, the number of vehicles passing a link (f) increases. In an extreme case, if all vehicles in a link can pass the link at a time step, ETSF has no advantage over the current supply framework. However, in real-world mesoscopic traffic simulations, the simulation time step cannot be too large. For example, in DynaMIT, the maximum movement distance of a vehicle during a time step should be smaller than the minimum length of all links, so that vehicles cannot cross more than 1 link during a time step. In DynaMIT, the simulation time step is, in most cases, smaller than 5 seconds.

3.4.5 Integrated Scenarios

This experiment is designed to compare the execution time of the ETSF and current mesoscopic simulation framework in special scenarios. Three different lengths of a link are evaluated: long (10000 m), medium (2000 m) and short (100 m). Three different demand levels are evaluated: high (500 v/h), medium (300 v/h) and low (100 v/h). The results are shown in Figure 19. For each case, both frameworks are executed. First, in the special case with short links and low level of demands, the execution times of both supply frameworks are similar. Second, when the length of a link increases and the level of demand increases, ETSF is more time efficient than the current framework. Third, in the special case with medium length of links and medium level of demands, the execution time of ETSF is around a half of the execution time of the current supply framework. Finally, the special case with long links and high level of demands, ETSF reduces 95% of the execution time of the current supply framework. This experiment shows that ETSF outperforms the current mesoscopic simulation framework in congested traffic scenarios.

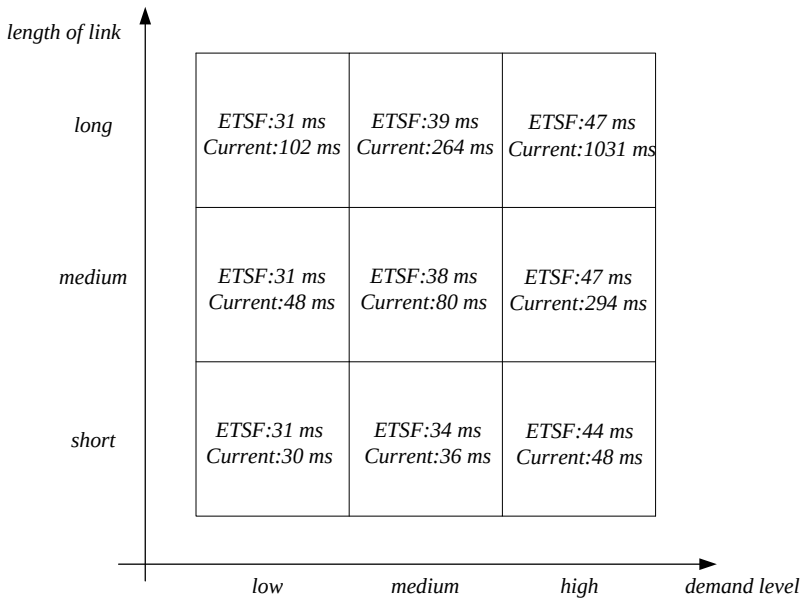


Figure 19: Comparisons of ETSF and the current mesoscopic simulation framework

3.5 Case Study

3.5.1 Experiment Setting

The Singapore expressway system is studied in this section. As in Figure 20, the expressway system consists of expressway segments and ramps connecting local roads with the expressway. The network has been modeled using a detailed representation of the length, geometry and lanes of each segment. The expressway system is made up of 831 nodes connected by 1040 links. Each link has one or multiple segments based on the road geometry and there are 3388 segments. The distribution of the length of the segments is shown in Figure 21. Most segments' lengths are in the range (300, 600). There are some short segments, which are mostly on-ramps or off-ramps, and there are also a few long segments. The simulation time step is 1 second. The demand is modeled as trips from 4106 OD pairs. Each origin is an on-ramp, where vehicles enter the expressway system from local roads, and each destination is an off-ramp, where vehicles depart the network. The configuration and calibration of the OD Matrix are explained in Section 6.5, and the calibrated OD matrices in two periods: Non-Peak-Hour (4:00AM to 5:00AM) and Peak-Hour (7:00AM to 8:00AM) are used in this section. In particular, there are in total 32,004 vehicles loaded into the non-peak traffic scenario and 106,386 vehicles loaded into the peak

traffic scenario. The routes of vehicles are pre-calculated using Path Size Logit model (Ben-Akiva et al., 1999) and routes are not changed during the traffic scenarios. The experiments were executed on a computer with a 2.83GHz Intel Core 2 Quad processor, 4GB memory and Ubuntu Linux 12.10.

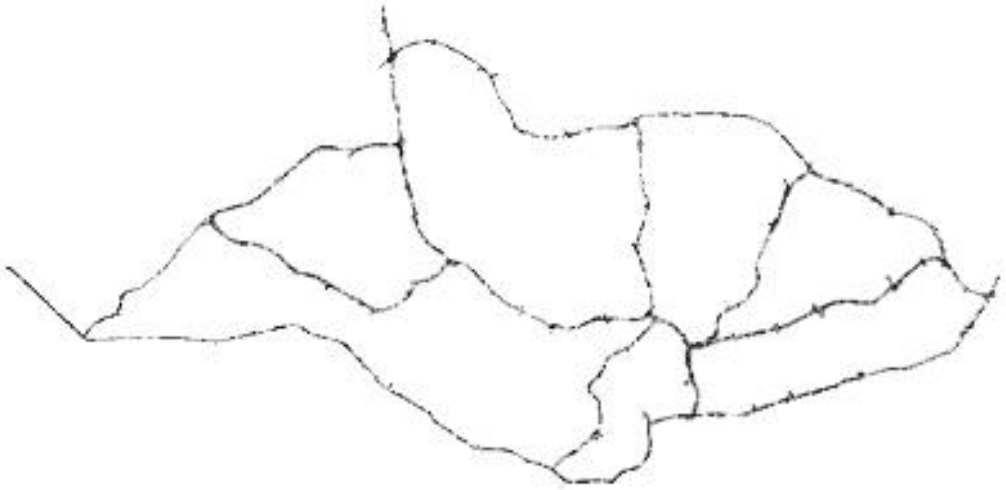


Figure 20: The Singapore expressway road network

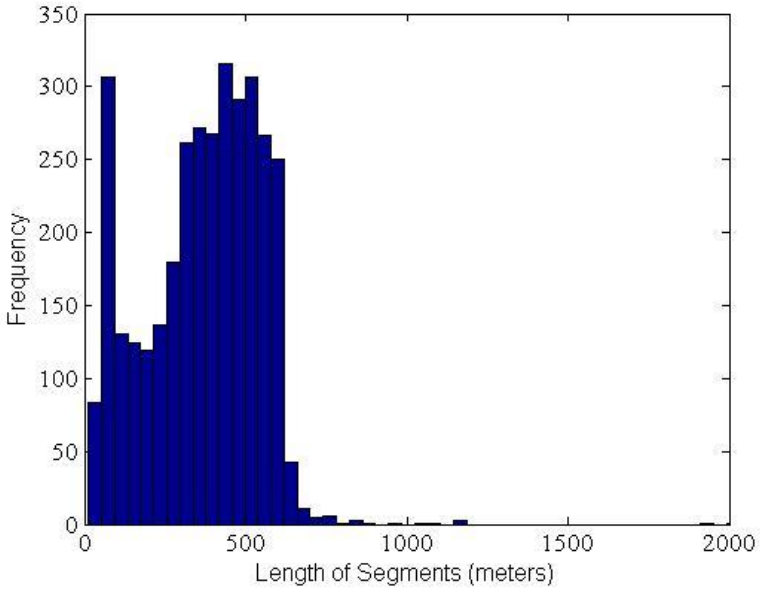


Figure 21: The distribution of the length of the segments in Singapore expressway

3.5.2 Results

The efficiency of ETSF is demonstrated in a real world city-scale traffic simulation. The results are shown in Table 8. In the first non-peak traffic scenario, ETSF performs a bit worse than the current traffic framework. This is expected, since in this scenario, the average travel time is smaller than 20 minutes. It means there are around 10,000 vehicles moving on the road network and there are only a few (less than 2) vehicles on each lane in average. The additional time cost comes from the calculation of t_p, t_q on each lane. In the second peak traffic scenario, ETSF is significantly better than the current traffic framework. As the number of vehicles loaded in a traffic scenario increases from 32,004 to 106,386 (3.32 times larger), the simulation time of the current traffic framework increase from 1863.9ms to 6690.2ms (3.59 times longer). The time cost of the current traffic framework is almost linear to the number of loaded vehicles. Note that there is no traffic congestion (e.g. spill-back) in the peak traffic scenario and the average travel speed is still high. The simulation time of ETSF only increase from 1951.6ms to 3448.0ms (1.77 times longer). The additional time cost of ETSF comes from more vehicles crossing between segments. In this traffic scenario, ETSF performs almost twice better than the current traffic framework, which shows the efficiency of ETSF.

Table 8: Evaluation of the ETSF on Singapore expressway network

ID	Time period	Total no. of vehicles	Simulation Time (ms)	
			Current	ETSF
1	Non-Peak-Hour (4:00AM to 5:00AM)	32,004	1863.9	1951.6
2	Peak-Hour (7:00AM to 8:00AM)	106,386	6690.2	3448.0

3.6 Summary

To tackle the problem of high computational cost when simulating congested traffic scenarios, this chapter proposed a novel Entry Time based Supply Framework (ETSF). The key idea is to model vehicles as a traffic fluid, when vehicles stay in the lane, and as individual vehicles, only when vehicles need to change lanes. ETSF has a lower computational complexity than the current

mesoscopic supply framework and more importantly, the computational complexity of ETSF is not sensitive to the number of vehicles in the simulated traffic scenario. Experiment results showed that ETSF outperforms the current supply framework, by reducing the execution time by 50% - 95% in large road networks and congested traffic scenarios. In the global research framework of mesoscopic traffic simulations, ETSF offers an innovative and time-efficient view to design the traffic dynamics on links. Combined with the parallel/distributed simulation technologies, ETSF is a candidate scalable supply framework for simulating city-scale traffic simulations.

4. Sim-Tree: A Two-Dimensional Spatial Index for City-scale Traffic Simulation

The spatial index was introduced in Section 2.3.3. This chapter firstly explains the shortcoming of using state-of-the-art spatial indexes in city-scale traffic simulations, and then introduces the motivations and the Sim-Tree. Finally, the efficiency of the Sim-Tree is demonstrated in a real-world parallel city-scale traffic simulation (Xu et al., 2014).

4.1 The Problem

The tree-based spatial index (Guttman, 1984; Beckmann et al., 1990) is a popular candidate solution to index moving objects in two-dimensional road networks. Figure 22(A) shows an example R*-tree based spatial index (the fanout is 3). Each node in the tree can have child nodes. Each leaf node is mapped to a two-dimensional area in the simulated road network and contains a list of objects located in this area. Each parent node has a mapping area which contains all child nodes' mapping areas and has a pointer to the list of child nodes. With such a tree structure, to do a region query, we do not need to check all objects in the simulated road network. Instead, we only need to scan the tree structure to get leaf nodes whose mapping areas overlap the target region and then check only objects in these leaf nodes. In order to make region queries efficient, the tree structure should be balanced, which means that objects in the simulated road network are evenly distributed under leaf nodes of the tree structure.

A drawback of tree-based spatial index is that it may adjust (or rebalance) its tree structure when vehicles update locations. As shown in Figure 22, the example R*-tree is balanced in (A). But when a vehicle v2 moves from R1 to R2, the tree is not balanced any more in (B). A split operation is executed and half of the drivers are re-inserted. The mapping areas of R1 and R2 also change. The rebalanced tree structure is shown in Figure 22(C). When a vehicle v4 moves

from R2 to R1, another rebalance operation is executed. The rebalanced tree structure is shown in Figure 22(D). Rebalancing the tree structure aims to guarantee that future region queries are efficient. However, in city-scale microscopic traffic simulation, a large number of vehicles are updating their locations at each simulation time step, making the time cost to update vehicles' locations and to adjust the tree structure expensive. The root of the inefficiency in R-Tree family is that R-Tree family is designed for scenarios with a large amount of queries and a limited number of updates, which is not true in city-scale traffic simulations. In our case studies to simulate the whole Singapore network, the time cost to manage an R*-Tree takes 10%-30% of the total simulation time, depending on the number of simulated vehicles. So, the question is "is there a way to build an efficient two-dimensional tree-based spatial index without incurring expensive update cost in a city-scale microscopic traffic simulation?"

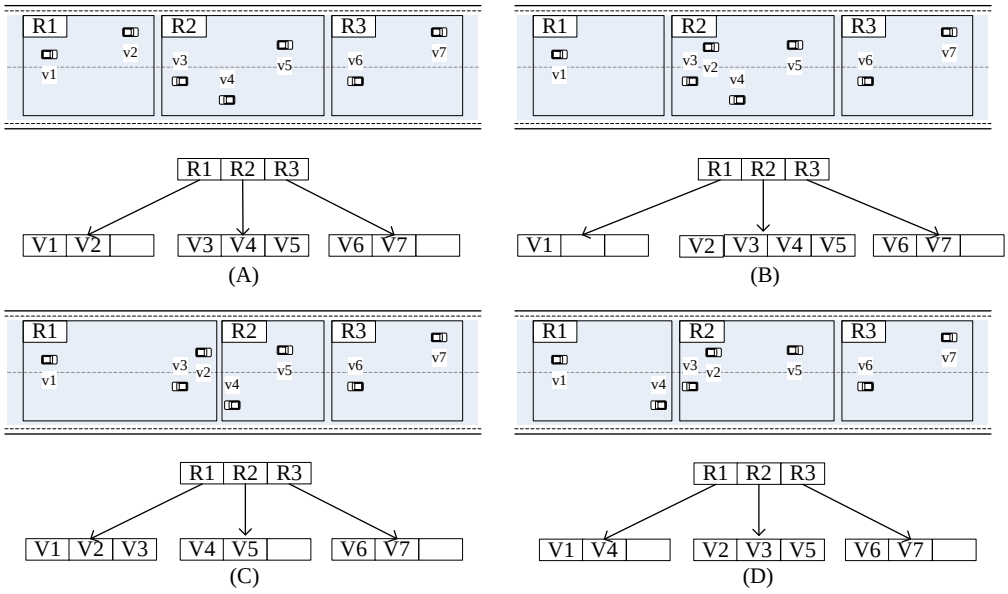


Figure 22: An example two-dimensional R*-tree based spatial index

There is another reason that makes this question important. Parallel microscopic traffic simulation is becoming popular for supporting the simulation of city-scale traffic scenarios. The main idea of parallel microscopic traffic simulation (Nagel at al., 2001, 2002) (Cameron et al., 1996) is to assign vehicles to cores (or CPUs), and then vehicles can be simulated in parallel. However, it is not trivial to parallelize the location update function, because parallel location updates may conflict with each other when adjusting the tree structure. Thus, the expensive serial

time cost of the location update function in two-dimensional tree-based spatial index also reduces the scalability of parallel microscopic traffic simulation.

4.2 The Key Ideas

This section introduces two observations in real-world traffic systems and traffic simulations, which are also the motivation to design a new spatial index.

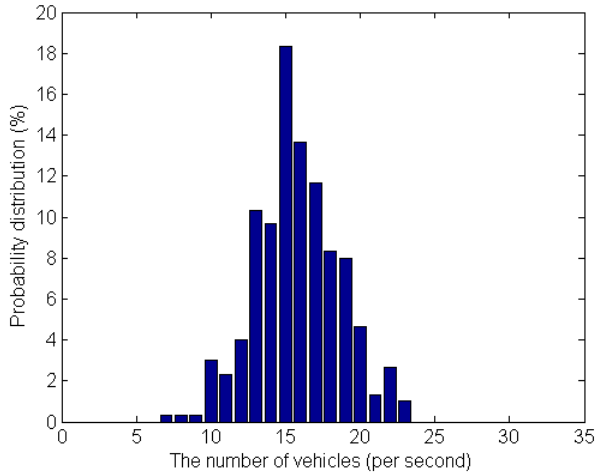
Observation 1: During a short period (e.g. 5 minutes) and a reasonably large area (e.g. a 100 meter road), the number of vehicles in the area is stable.

For example, Figure 23(A) shows the distribution of the number of vehicles (per second) in 5 minutes on a 100 meter section of a real-world 4-lane road on the Ayer Rajah Expressway in Singapore. The average number of vehicles in the 5 minutes is 15.8 and the standard deviation is 2.85. 94% of the numbers of vehicles in the 5 minutes are in the range [10, 20]. Figure 23(B) shows the trend of the number of vehicles in the 5 minutes on the section. It can be seen that whenever the number of vehicles departs from the average number of vehicles during the period, the number of vehicles will inadvertently return to the average number of vehicles.

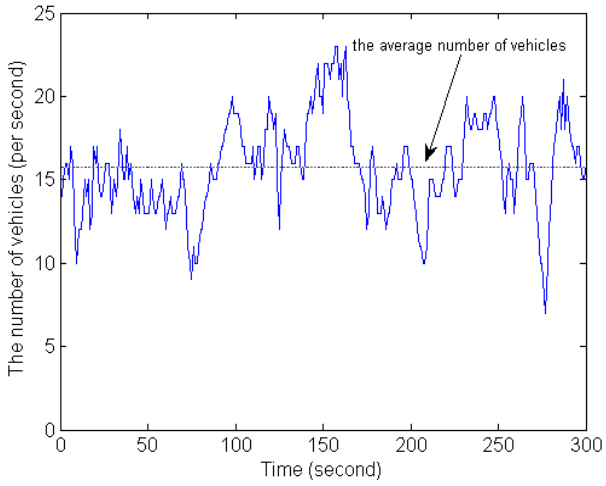
The traffic flow theory gives another explanation. The number of drivers in an area is determined by the drivers' demand on the area and the supply capacity of the area. During peak hours, the drivers' demand on the area is high, so the expected number of vehicles in the area is high. If an incident happens in the area, the supply capacity of the area is reduced, so traffic congestion might happen and the expected number of vehicles in the area is also high. If both the drivers' demand on the area and the supply capacity of the area are not changed, during a short period and a reasonably large area, the expected number of vehicles in the area is stable.

This observation means that if a balanced tree structure of a spatial index is built based on the average road density in a road network during a period, the tree structure tends to be balanced at each simulation time step within the period. More importantly, since the road density in a road network is not sensitive to an individual vehicle's location, there is no need to check or rebalance the tree when individual vehicles update their locations. Building a balanced (and stable) two-

dimensional spatial index based on the average road density in a road network during a period is the main idea behind the Sim-Tree.



(A) The distribution of the number of vehicles (per second) in 5 minutes on the section



(B) The trend of the number of vehicles (per second) in the 5 minutes on the section

Figure 23: The observed road density on a Singapore expressway section

Observation 2: The region query function and the location update function are used separately in a simulation time step in microscopic traffic simulations.

As shown in Figure 24, there are two phases in a simulation time step in microscopic traffic simulations. In the query phase, vehicles use the region query function to get nearby objects,

apply various travelers' behavior models to determine speeds, acceleration and locations. When all vehicles complete the travelers' behavior models, the traffic simulation goes to the update phase. In the update phase, vehicles use the location update function to update their new locations in the spatial index for the next simulation time step.

This observation implies two things. First, there are no location updates in the query phase. Thus, if a tree structure is balanced in the beginning of the query phase, all region queries in the query phase are efficient. Second, there are no region queries in the update phase. Thus, an imbalanced tree structure is acceptable in the update phase. In summary, we only need to check and rebalance (if required) the tree structure once at the beginning of each simulation time step.

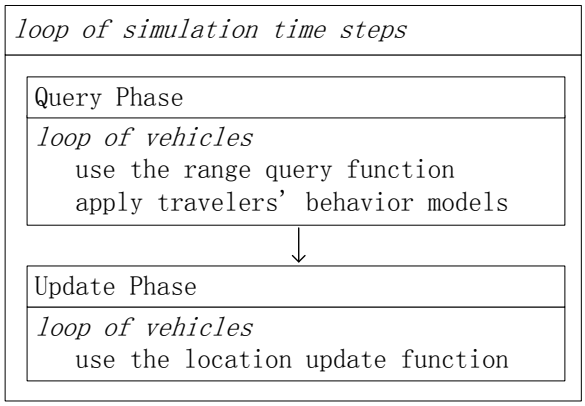


Figure 24: The region query function and the location update function

4.3 Sim-Tree

Motivated by the observations listed in Section 4.2, this section proposes Sim-Tree, a novel two-dimensional tree-based spatial index for city-scale parallel microscopic traffic simulations.

4.3.1 Data Structure

The data structure of Sim-Tree is shown in Figure 25. Each node in the tree structure has pointers to its child nodes and a pointer to its parent node. Each node has a two-dimensional mapping area (e.g. a rectangle) in a road network. The root node is mapped to the whole road network. The inner nodes are mapped to smaller areas. Each leaf node has an object buffer, which contains a list of objects which are located inside the leaf node's mapping area. Locations of objects (e.g.

drivers and pedestrians) are modeled as points. Leaf nodes' mapping areas have no overlap, thus, each object is located inside only one leaf node.

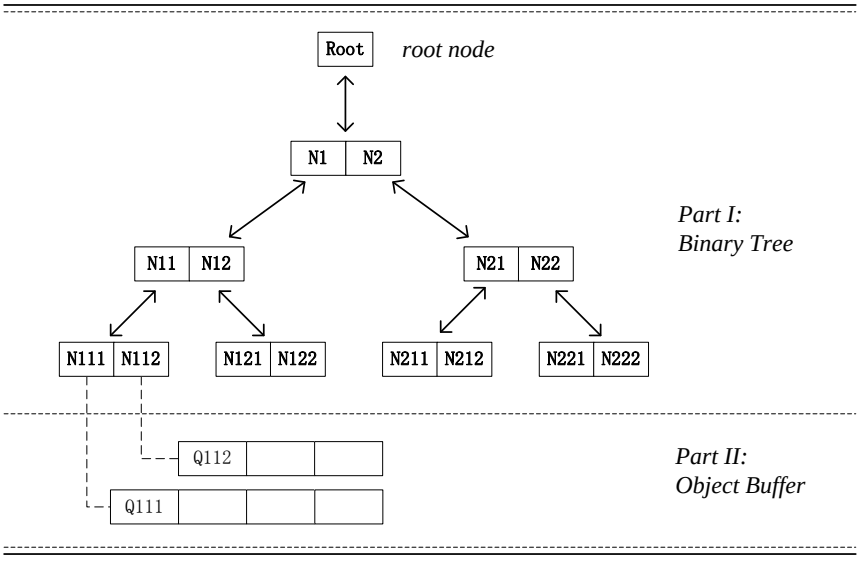


Figure 25: The Sim-Tree data structure

The innovation of Sim-Tree is not on its shape, but on its fit to the requirements of city-scale parallel microscopic traffic simulation. First, the binary tree structure in Sim-Tree is constructed based on the average road density in a road network during a period (observation 1). Second, the tree structure is checked only once in the beginning of each simulation time step (observation 2). Third, Sim-Tree is designed to minimize the time cost of the location update function (with the cost of non-optimal range query cost) to improve the scalability of parallel microscopic traffic simulations.

This paragraph explains why a binary tree is used in Sim-Tree. Assuming that objects are evenly distributed in leaf nodes and a query region is inside the mapping area of a leaf node, let w be the average number of objects in a leaf node, k be the number of child nodes in each node and N be the total number of leaf nodes, the average cost of a region query operation can be estimated using a formule:

$$\text{average query cost} = k * \log_k^N + w, \ (\mathrm{k} \geq 2, \mathrm{w} \geq 0) \tag{4.1}$$

where, $\log_k^N$ is the depth of the tree structure, $k * \log_k^N$ is the cost of finding the target leaf node, and w is the cost of check objects in the target leaf node. The cost unit is a rectangle operation,

e.g. to check whether a rectangle overlaps with another rectangle. To minimize the region query cost, the best value of *k* is *e* (the calculation is in Appendix II). In this chapter, k is 2.

There are two types of road densities used in Sim-Tree: pre-simulation road density and in-simulation road density. Pre-simulation road density indicates that the density on each road at each time step for a similar traffic scenario is available before simulating a traffic scenario. Pre-simulation road density is firstly divided into multiple periods, in order that the density pattern within each period is not significantly different. For example, the road density in a normal day can be divided into six periods: morning {pre-peak, peak, post-peak} and afternoon {pre-peak, peak and post-peak}. Then, the average road density during each period is used to build the initial balanced binary tree for that period. In-simulation road density indicates that the average road density during a recent period (e.g. 5 minutes) is available when simulating a traffic scenario. It is true for most microscopic traffic simulations (Yang, 1999; Cameron et al., 1996), because the density of each road is an important output of traffic simulations.

This paragraph explains how to build a balanced binary tree in Sim-Tree, given the average density on each road during a period. Firstly, the road network is evenly divided into a number of cells (e.g. 1 m^2). Then, each road is associated to multiple overlapping cells. Assuming vehicles are evenly distributed on a road, the average densities on roads are transformed into the average density in cells, according to the portion of road length in each cell. The relationship between roads and cells is pre-calculated before simulating a traffic scenario and is not changed during the simulation. Given the average density in cells at a time step (e.g. the simulation time is 0) during the simulation, Sim-Tree starts with only a root node, which is mapped to the whole road network (all cells) and contains all objects in the road network. Then, the node is divided into two child nodes vertically or horizontally. Each child node is mapped to a smaller area and contains half of the objects in the road network. These two child nodes' mapping areas do not overlap with each other. The division process is similar to the orthogonal recursive bisection algorithm (see Section 2.3.4). After that, each child node is divided into two nodes using the same method. The node division process is repeated until:

1. The average number of objects in one node is smaller than 5.
2. The size of one node's mapping area is smaller than a pre-defined threshold.

Rule 1 means that if there are no more than 4 objects in a node, there is no benefit to divide the node into 2 child nodes. If the node is divided into 2 child nodes, the depth of the branch is increased by 1 and the cost of finding the correct leaf node is increased by two rectangle operations. At the same time, the benefit comes from checking less number of objects. In the best case, if objects are evenly distributed in the two child nodes, the benefit is also two rectangle operations. Thus, if the number of objects in a node is not more than 4, there is no benefit to divide the node. In traffic simulations, drivers are configured to use the same-sized rectangles in the region query function. Rule 2 means that if the mapping area of a node is smaller than a pre-defined threshold (e.g. ¼ of the rectangle's size), a driver's region query tends to contain the mapping area of the node. In this case, all objects contained in the node can be returned immediately without additional checking. Thus, there is no benefit to divide the node. If the time cost of range queries from pedestrians is significant, the rectangle size of pedestrians' query region should be used instead.

4.3.2 Functional Design

As shown in Table 9, Sim-Tree has five public functions and two private functions. The first function is InitializeTreeStructure. It is used before a simulation run. This function builds a balanced tree structure based on the historical road density in a road network. The second public function is Insert. It is used to insert a new object in Sim-Tree. After inserting an object in Sim-Tree, other objects can query (or can see) this object using the region query function. Besides, an object is automatically removed from Sim-Tree when the object is removed from the simulation. The third public function is RegionQuery. Given a target rectangle, this function scans the tree structure from the root node and finds all leaf nodes whose mapping areas overlap the target rectangle. Then, this function checks objects in these leaf nodes to determine which objects are in the target rectangle. The next is BottomUpRegionQuery. This function has the same purpose as the function RegionQuery, but it is implemented in a different way. The bottom-up region query function is explained in Section 4.3.4. The last public function is UpdateAll. This function is used once in each simulation time step to update objects' new locations in the Sim-Tree. If a vehicle's new location is still inside its current leaf node, there is no need to update anything. Otherwise, the vehicle will be moved to the appropriate leaf node. The two private functions are used by Sim-Tree to check and rebalance its tree structure. The rebalance function is explained in

Section 4.3.3. Note that the function InitializeTreeStructure is not a strictly mandatory function, because Sim-Tree can rebalance its tree structure. However, starting from a reasonable tree structure and reducing (even disabling) rebalancing is always better than starting from an empty tree structure.

Table 9: The interface of the Sim-Tree

public functions:	
1	InitializeTreeStructure ()
2	Insert (OneObject)
3	RegionQuery (A Rectangle)
4	BottomUpRegionQuery (OneObject, A Rectangle)
5	UpdateAll ()
private functions:	
1	MeasureBalance ()
2	Rebalance ()

4.3.3 The Rebalance Function

Although the target of Sim-Tree is to avoid rebalancing the tree structure, there are two conditions where Sim-Tree has to adjust its tree structure. First, if there is no prior information about the average road density in a road network before simulating a traffic scenario, the Sim-Tree has to learn the road density while the simulation is ongoing and adjust its tree structure. Second, if road density in the road network changes significantly during a traffic scenario (e.g. an incident happens), Sim-Tree has to learn new road densities and adjust its tree structure. To be exact, a rebalance operation happens if any of the follow rules is true.

1. The average number of objects in all leaf nodes is smaller than 2.
2. The average number of agents in all leaf nodes is larger than 32 and the average rectangle size of all leaf nodes is larger than a pre-defined threshold.
3. The imbalance ratio of the tree structure is larger than 0.3.

Rule 1 means that if the depth of a Sim-Tree is too high and many leaf nodes have only 1 object or even have no object, the Sim-Tree needs to be rebalanced. Rule 2 means that if the depth of a Sim-Tree is too low (there are a large number of agents in most of the leaf nodes) and the average rectangle size of leaf nodes is larger than a pre-defined threshold (e.g. half of the size of a driver's query rectangle), the Sim-Tree needs to be rebalanced. If the average number of agents in all leaf nodes is larger than 32, but the average rectangle size of the leaf nodes is

smaller than the threshold, it means there is congestion in the road network. Since a typical driver's region query tends to contain leaf nodes' mapping area and objects in leaf nodes can be directly returned without checking individually, the region query function is still efficient and there is no need to rebalance the tree structure. Rule 3 means that if the depth of Sim-Tree is acceptable, but objects are not evenly distributed in leaf nodes, the Sim-Tree needs to be rebalanced. Note that parameters in the 3 rules are configurable. The imbalance ratio of a tree structure is measured by a formula:

$$\textit{The imbalance ratio} = \frac{\sum_{n \in N} \left| Objects(n) - E(N) \right|}{\sum_{n \in N} Objects(n)} \tag{4.2}$$

where, N is the list of leaf nodes, Objects (n) means the number of objects in node(n)'s mapping area, E(N) is the average number of objects in all leaf nodes. An imbalance ratio is a variable in [0, 1]. A smaller imbalance ratio means that the tree structure is more balanced. For example, if the imbalance ratio of a tree structure is 0, the tree structure is perfectly balanced.

There are three variables affecting a rebalance operation. The first variable is the checking frequency (f), which controls the frequency to check the three rules. The second variable is the imbalance ratio threshold (u). If the imbalance ratio of a tree structure is larger than u, the tree structure is judged as imbalanced (the default value is 0.3). The last variable is the confirmation threshold (c). Only if a tree structure is continuously judged as imbalanced more than c times, a rebalance operation is started. A rebalance operation is to throw away the previous imbalanced Sim-Tree and to re-build a new Sim-Tree using the average density on road during a recent period and the pre-calculated relationship between roads and cells, which was explained in Section 4.3.1. The sensitivity of these three variables to the performance of a rebalance operation is studied in Section 4.3.4.

4.3.4 The Bottom-Up Region Query Function

A key step in a region query operation is to find a node whose mapping area is just large enough to contain the target region. This node is named as the root proxy node of the region query operation. Starting a region query operation from the root proxy node will achieve the same result as starting the region query operation from the root node. As shown in Figure 26, a vehicle

v2 queries nearby objects in a rectangle and N1 is the lowest node containing the rectangle. Thus, N1 is the root proxy node of this region query. In case of a region query in city-scale traffic simulation, the depth of the Sim-Tree is high and the distance from leaf nodes to the root proxy node is much smaller than the distance from the root node to the root proxy node. Motivated by this, we design a bottom-up region query function. The idea of a bottom-up region query is to find the root proxy node of a vehicle's region query operation from the leaf node of the vehicle in a bottom-up direction, instead of from the root node of the tree structure in a top-down direction.

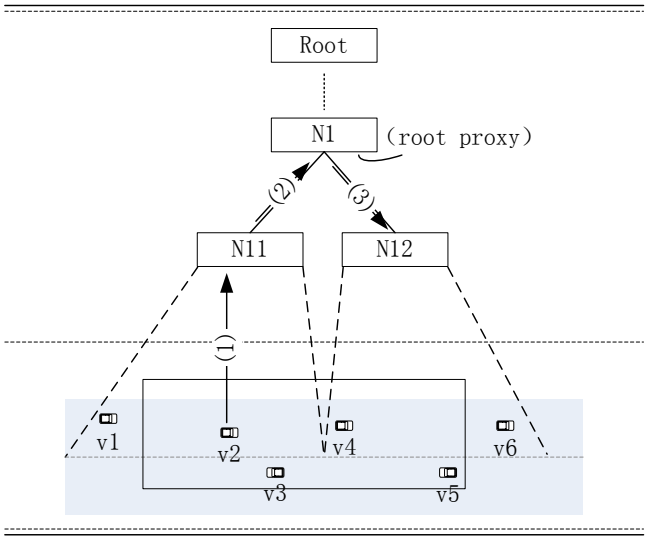


Figure 26: An example bottom-up region query by a vehicle (v2)

The bottom-up region query function consists of three steps. The first step is to check whether the vehicle's leaf node's rectangle overlaps the vehicle's query region. If not, a normal region query function starting from the root node is executed. If yes, the next step is to find the root proxy node of the region query in a bottom-up direction. The last step is to initiate a normal region query starting from the root proxy node. As shown in Figure 26, a vehicle v2 queries nearby objects in a rectangle. Firstly, the query rectangle overlaps with the vehicle's leaf node (N11), so a bottom-up region query is suitable. Second, an upward step is required to find the root proxy node N1 from the leaf node N11. Finally, a normal region query is initiated from N1.

The bottom-up region query function is more efficient than a normal region query in city-scale traffic simulation. There are two reasons. First, the query region of a vehicle is a rectangle

surrounding the location of the vehicle. So, starting from the leaf node of the vehicle is a reasonable choice. Second, as explained in Section 4.3.2, the mapping area of leaf nodes cannot be much smaller than a vehicle' query region. Thus, in most cases, only a small number of upward steps are required to find the root proxy node in a bottom-up direction. In our experiments to simulate 69,567 vehicles in the whole Singapore network, the average number of upward steps is smaller than 3.

4.4 Qualitative Analysis

As shown in Table 10, the main difference between the Sim-Tree and the state-of-the-art techniquesis the criteria to rebalance its tree when the tree becomes imbalanced and inefficient. First, a R*-Tree needs a balance check when any vehicle changes its location, a LRU-Tree needs a balance check when a vehicle changes its location and the new location is outside of its corresponding leaf nodes' mapping area, a Sim-Tree needs a balance check when all simulated vehicles change their locations at the end of a simulation time step. Second, a R*-Tree and a LRU-Tree need a rebalance if vehicles are not evenly distributed in the network, even in the middle of a time step; a Sim-Tree needs a rebalance only if links' mean densities change significantly after a number of time steps. Given an example of simulating 1000 vehicles moving on a road network in one time step, the R*-Tree needs to check its balance (and then possible rebalance) 1000 times; the LRU-Tree needs to check its balance (and then possible rebalance) 50 times, assuming 5% of vehicles move out of the corresponding leaf nodes' mapping area; the Sim-Tree needs to check its balance (and then possible rebalance) only once.

Table 10: Qualitative Comparison of the Sim-Tree and the State-of-the-Arts

	Criteria of balance check	Criteria of rebalancing
R*-Tree	Any vehicle changes its location	Vehicles are not evenly distributed in the network
LRU-Tree	A vehicle changes its location and the new location is outside of its corresponding leaf nodes' mapping area	Vehicles are not evenly distributed in the network
Sim-Tree	The end of a simulation time step	Links' mean densities change significantly

4.5 Case Study

4.5.1 Experimental Setting

This section introduces three major components of the testbed: a microscopic traffic simulator, the Singapore road network and a city-scale traffic scenario.

SimMobilityST is an microscopic traffic simulator, developed by the Future Urban Mobility in Singapore-MIT Alliance for Research and Technology (SMART). SimMobilityST is based on the concept of agent-based or micro-simulation. Representation of individuals as agents in the model is necessary to simulate how people will react in the future to new infrastructures, new technologies and innovations in system management and policy changes. It simulates two kinds of behaviors: (1) Decision-making behavior (like route choice) is taken when agents are at some decision point e.g., a bus stop, and (2) Agent movement behavior occurs during the movement (lateral or longitudinal) e.g., "Car Following" and "Lane Changing". SimMobilityST is also a parallel microscopic traffic simulator to for simulating city-scale traffic scenarios. Vehicles in a road network are evenly distributed on cores (or CPUs), so that vehicles are simulated in parallel. The default simulation time step is 0.1 seconds.

Singapore is a country that is well-known for its advanced transportation system. Figure 27 shows the Singapore road network. The Singapore road network consists of expressways, major arterial roads, collector roads and local roads. In this testbed, the Singapore road network is modeled as a map with 10,702 nodes and 20,918 segments.

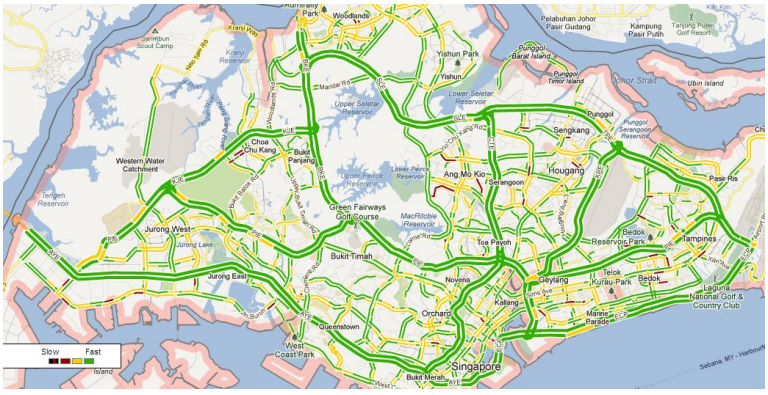


Figure 27: The Singapore road network

A city-scale traffic scenario is created. The time is from 8:00AM to 8:30AM on a weekday. 69,567 trips are generated during this period, based on Singapore's Household Interview Travel Survey (HITS). The Singapore government conducts the HITS Survey every four to five years to give transport planners and policy makers insights into residents' travelling patterns. Note that the purpose of this thesis is not to generate the exact trips on the Singapore road network, but to generate a reasonable city-scale traffic scenario. The first 10 minutes are the warm up period.

4.5.2 Efficiency

This experiment evaluates the time cost of the region query function and the location update function in Sim-Tree. The traffic scenario is to simulate 69,567 trips during a typical morning period in the Singapore road network. The microscopic traffic simulator SimMobilityST is used to simulate the traffic scenario using 1 worker thread. It means that the 69,567 trips are simulated using only 1 thread on 1 core. Four tree-based spatial indexes are evaluated in this experiment: the R*-tree, the LUR-tree, Sim-Tree and Sim-Tree-No-BU. The R*-tree and the LUR-tree were introduced in Section 2.3.3. The only difference between Sim-Tree and Sim-Tree-No-BU is that there is no bottom-up region query function (in Section 4.3.3) in Sim-Tree-No-BU.

The results are shown in Table 11. Four tree-based spatial indexes are evaluated and the measurements are the cost of the location update function, the cost of the region query function and the total cost of simulating the traffic scenario. First, and most importantly, the location update cost of the Sim-Tree is significantly lower than both the R*-Tree and the LUR-tree. This is expected, because the major target of Sim-Tree is to reduce the cost of the location update function (based on observations 1 and 2). Compared with Sim-Tree, both the R*-Tree and the LUR-tree spend time on operations (e.g. re-insert, split, merge, shrink and expand) in order to rebalance the tree structure. In addition, when using Sim-Tree in this case study, for more than 90% of location updates, the vehicles' new locations are still in the same leaf node, which generates very little additional overhead. Second, the region query cost of Sim-Tree is also better than the R*-Tree and the LUR-tree. This is an interesting finding. In the R*-Tree, one of the key parameters is the fanout of a node (the number of child nodes). If the fanout is large, a node is mapped to a large area in a road network, so the cost of the location update function is reduced (because vehicles' new locations tend to be in the same node). However, the cost of the region

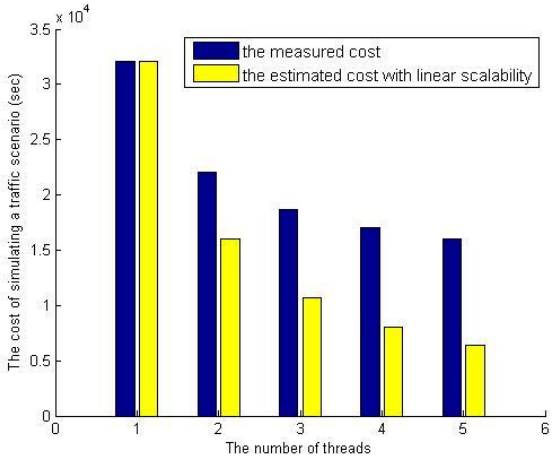
query function is also increased (as explained in Formula 4.1). Thus, the fanout of a node is a compromise between the cost of the location update function and the cost of the region query function. Based on the experiments, the fanout of a node in the R*-Tree in this case study is set to 50 to minimize the total simulation cost. However, in Sim-Tree, there is no need to worry about the fanout of a node, which is set to be 2, in order to optimize the region query function. Third, compared with Sim-Tree-No-BU, the bottom-up region query function (in Sim-Tree) reduces the cost of the region query function by 10.6%. Finally, Sim-Tree does not significantly reduce the total simulation cost. This is expected, since Sim-Tree is designed to reduce the time cost of the serial location update function and improve scalability.

Table 11: Performance comparison of four tree-based spatial indexes

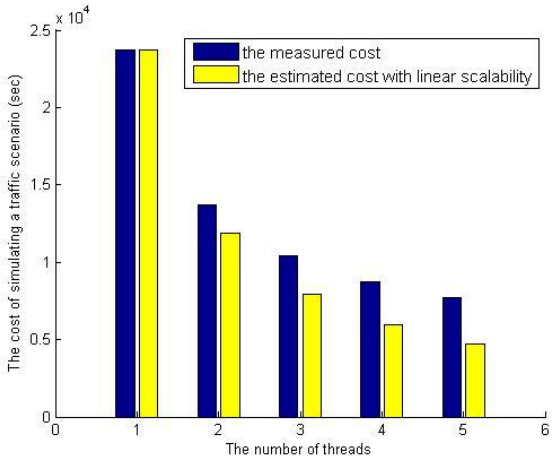
	The location update (sec)	The region query (sec)	The total simulation cost (sec)
The R*-Tree	9,643	2,660	32,047
The LUR-tree	1,266	2,633	23,741
Sim-Tree-No-BU	34	2,139	22,172
Sim-Tree	34	1,920	21,970

4.5.3 Scalability

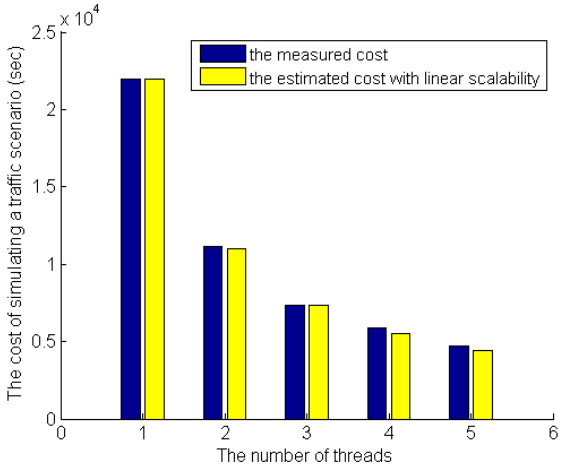
This section evaluates the scalability of Sim-Tree. The traffic scenario is the same as the traffic scenario in the first experiment. But the scenario is simulated using multiple threads (1-5 worker threads). Each worker thread is executed exclusively on a core and the master thread is executed on the sixth core. The results of simulating the traffic scenario in a parallel way using the R*-Tree, the LRU-Tree and the Sim-Tree are shown in Figure 28. The x-axis corresponds to the number of worker threads and the y-axis corresponds to the total time cost of simulating the traffic scenario. The blue bar is the measured time cost and the yellow bar is the time cost if a linear speedup (or a linear scalability) is achieved. If the blue bar is close to the yellow bar, it means a close-to-linear speedup. For the R*-Tree, there is a gap between the blue bar and the yellow bar. For example, the speedup of using 5 worker threads is only 2, because of the heavy serial cost in the location update function. For the LUR-Tree, the speedup of using 5 worker threads is improved from 2 to 3, due to reduced serial cost in the location update function. For the Sim-Tree, the blue bar is close to the yellow bar, which shows the Sim-Tree significantly improved the scalability of the parallel traffic simulation.



(A) The total time cost of simulating the traffic scenario using the R*-Tree



(B) The total time cost of simulating the traffic scenario using the LUR-Tree



(C) The total time cost of simulating the traffic scenario using the Sim-Tree

Figure 28: The total time cost of simulating the traffic scenario in a parallel way

4.5.4 The Rebalance Function

This experiment evaluates the rebalance function. The traffic scenario is the same as the traffic scenario in the first experiment. However, there is no prior information about the average road density in the road network. Thus, Sim-Tree needs to adjust its tree structure when the simulation is ongoing. This traffic scenario is simulated using 5 worker threads. As explained in Section 4.3.3, there are three variables controlling the rebalance function: the checking frequency (f), the imbalance ratio threshold (u) and the confirmation threshold (c). Based on our knowledge, the imbalance ratio threshold (u) is the most critical parameter. In this case study, the checking frequency (f) is 10, which means Sim-Tree checks its tree structure every 10 simulation time steps (or 1 second). The confirmation threshold (c) is 1, which means a rebalance operation is triggered only if Sim-Tree is imbalanced in two continuous checking. In this case study, different imbalance ratio thresholds (u) are evaluated. The cost of the rebalance function consists of three parts. The first part (Part 1) comes from checking whether a rebalance operation is required. The second part (Part 2) comes from aggregating the road density in the road network during a previous period. The third part (Part 3) comes from rebalancing a tree structure based on the aggregate road density in the road network in the previous period.

Table 12 shows the performance of the rebalance function using different imbalance ratio thresholds (u). The measurements are the number of the rebalance operations, the cost of the rebalance function and the cost of the region query function. First, the cost of rebalancing the tree structure (Part 3) dominates the cost of the rebalance function. Both the cost of checking whether a rebalance operation is required (Part 1) and the cost of aggregating the road density in the road network (Part 2) are small. Second, the cost of the rebalance operation (Part 3) is sensitive to the imbalance ratio threshold (u). As u increases from 0.15 to 0.35, the number of the rebalance operation is reduced from 632 to 2 and the cost of the rebalance operation is reduced from 849 seconds to 2 seconds. The cost of a rebalance operation depends on the number of vehicles in the road network. In this case study, the cost of a rebalance operation is around 1.0-1.5 seconds. Besides, the cost of the region query function is not sensitive to u when u is smaller than 0.3. Third, the rebalance operation is used twice (when u is 0.35), which is the same with the case when u is 0.3. However, the time cost of the region query function when u is 0.35 is higher. The reason is that the rebalance operations when u is 0.35 are triggered later, which

makes region queries before the rebalance operations inefficient. In summary, in this experiment, when there is no prior information about the road density in the road network, the rebalance function (when u is 0.3) enables the Sim-Tree to do region queries efficiently with small additional time cost.

Table 12: The performance of the rebalance function in the Sim-Tree

imbalance ratio threshold (u)	No. of rebalance	The rebalance function (sec)			The region query function (sec)
		Part 1	Part 2	Part 3	
0.15	632	143e-6	449e-6	849	399
0.20	373	209e-6	499e-6	498	401
0.25	45	138e-6	469e-6	61	404
0.30	2	129e-6	436e-6	2	408
0.35	2	121e-6	434e-6	2	446

4.6 Summary

Motivated by observations in a real-world microscopic city-scale traffic simulation, this chapter proposed a new two-dimensional spatial index: Sim-Tree. The main idea of Sim-Tree is to build a tree structure based on the average road density in a road network. The key feature of Sim-Tree is that there is no need to check or rebalance its tree structure when individual vehicles frequently update their locations. In experiments to simulate a city-scale traffic scenario, Sim-Tree performed significantly better than the state-of-the-art R*-tree family of spatial indexes. Sim-Tree reduced the time cost of a serial bottleneck in the location update operation by 97% (compared to the LUR-tree) and achieved a near linear speed-up.

There are three ways to enhance Sim-Tree in future. First, the current rebalance function is reactive, which means that a rebalance operation is triggered only if the tree structure is already imbalanced. The rebalance function can be changed to be proactive. Second, in some traffic conditions, there is no need to update a vehicle's surrounding traffic every simulation time step (e.g. 0.1 second). Reducing the frequency of updating a vehicle's surrounding traffic tends to reduce the total time cost of region query operations. Third, Sim-Tree, as a two-dimensional spatial index, can also be used as a supplement to a one-dimensional spatial index (e.g. a linear reference).

5. Scalable City-scale Traffic Simulation on the CPU/GPU Platform

5.1 The CPU/GPU Platform

Originally designed as a specialized hardware to accelerate the creation and the rendering of images for output to a display, modern graphics processing units (or GPUs) have evolved into a highly parallel, multi-core processor with tremendous computational power and very high memory bandwidth. The key to the popularity of GPU computing has been its massive performance when compared to the central processing units (or CPUs). Today, there is a performance gap of roughly seven times between the two when comparing theoretical peak bandwidth and gigaflops performance (Brodtkorb et al., 2013). This performance gap has its roots in physical restraints and architectural differences between the two processors. The CPU is in essence a serial von Neumann processor, and is highly optimized to execute a series of operations in order, while the GPU is specialized for compute-intensive, highly parallel computation - exactly what graphics rendering is about - and therefore designed such that more transistors are devoted to data processing rather than data caching and flow control. More specifically, the GPU is especially well-suited to address problems that can be expressed as data-parallel computations - the same program is executed on many data.

The computational performance of GPUs is growing due to their massive parallelism. However, increased parallelism will only increase the performance of parallel code sections, meaning that the serial part of the code soon becomes the bottleneck. Thus, most applications benefit from the combination of a massively parallel GPU and a fast multi-core CPU. In this thesis, this combination is named as the CPU/GPU platform. There are three major GPU vendors today, Intel being the largest. However, Intel is only dominant in the integrated and low-performance market. For high-performance graphics, AMD and NVIDIA are the sole two

suppliers. In academic and industrial environments, NVIDIA appears to be the clear predominant supplier, and we thus focus on GPUs from NVIDIA in this thesis, even though most of the concepts and techniques are also directly transferable to GPUs from AMD.

The key concepts in order to execute traffic simulations on the CPU/GPU platform include:

1) Heterogeneous programming
2) The thread hierarchy in the GPU
3) The memory hierarchy in the GPU

Heterogeneous programming means that the GPU (or CUDA) threads are assumed to be executed on a physically separate device from the CPU threads, which has its own processors and memory space. The GPU program is launched by the CPU program, and then the CPU program and the GPU program run simultaneously and communicate through the GPU/CUDA APIs. In the CPU/GPU platform, the CPU/GPU are also named as host/device. The memory spaces are named as host/device memory (or the CPU/GPU memory). The processors are named as host/device cores (or the CPU/GPU cores).

The thread hierarchy in the GPU consists of grids, blocks and warps. The execution of a kernel function on the GPU launches a grid of blocks. Each block consists of a number of wraps. Each warp contains a fix number of threads (the number is 32 in the current NVIDIA GPU architectures (Nvidia, 2013)). Threads within the same warp can synchronize and cooperate using fast shared memory. The aim of the massively threaded architecture of the GPU is to hide memory access latencies. A full utilization is achieved when there is always a warp that is ready to be executed at every clock cycle.

The memory hierarchy in the GPU consists of registers, shared memory, and global memory. Registers are the fastest memory units on a GPU, and each GPU has a limited register space. Registers are private for each thread. The second fastest memory type is shared memory, and shared memory can be just as fast as registers if accessed properly. Shared memory is accessible to all threads within one block, thus enabling cooperation. However, its size is limited and it can be a limitation to the number of launched threads per block. The third, and slowest type of memory is the global memory, which is also the largest memory in the GPU. Even though it has an impressive bandwidth, it has a high latency. Finally, different from the CPU programming, GPU programming has to specify which type of memory space to load data into.

5.2 ETSF on the CPU/GPU Platform

In this section, the proposed Entry Time-based Supply Framework (ETSF), which was discussed in Chapter 3, is re-designed to be executed on the CPU/GPU platform. To the best of our knowledge, this is a pioneer work on running a real-world city-scale traffic simulation on the CPU/GPU platform (Xu et at., 2014).

5.2.1 The Framework

The major motivation to design a new simulation framework is to make full use of two types of computational resources: the CPU and the GPU. In this framework, the GPU is responsible for the supply part of a mesoscopic traffic simulation. A key feature of supply simulation is that the simulation of a segment in a road network is only related to its surrounding segments, which fits the GPU's data parallel requirement. The CPU is responsible for the demand part and the I/O part of a mesoscopic traffic simulation. A key feature of demand simulation is that vehicles are making decisions based on the information on the global road network, which fits the CPU's random memory access feature. The supply and the demand of a mesoscopic traffic simulation were introduced in Section 2.1 and Section 2.2. Figure 29 shows the framework to execute a mesoscopic traffic simulation on the CPU/GPU platform. Compared to a traditional mesoscopic traffic simulation framework on the CPU, there are three key differences:

1) Asynchronous time management
2) Division of simulation components on the CPU and the GPU.
3) Data structure in the GPU memory

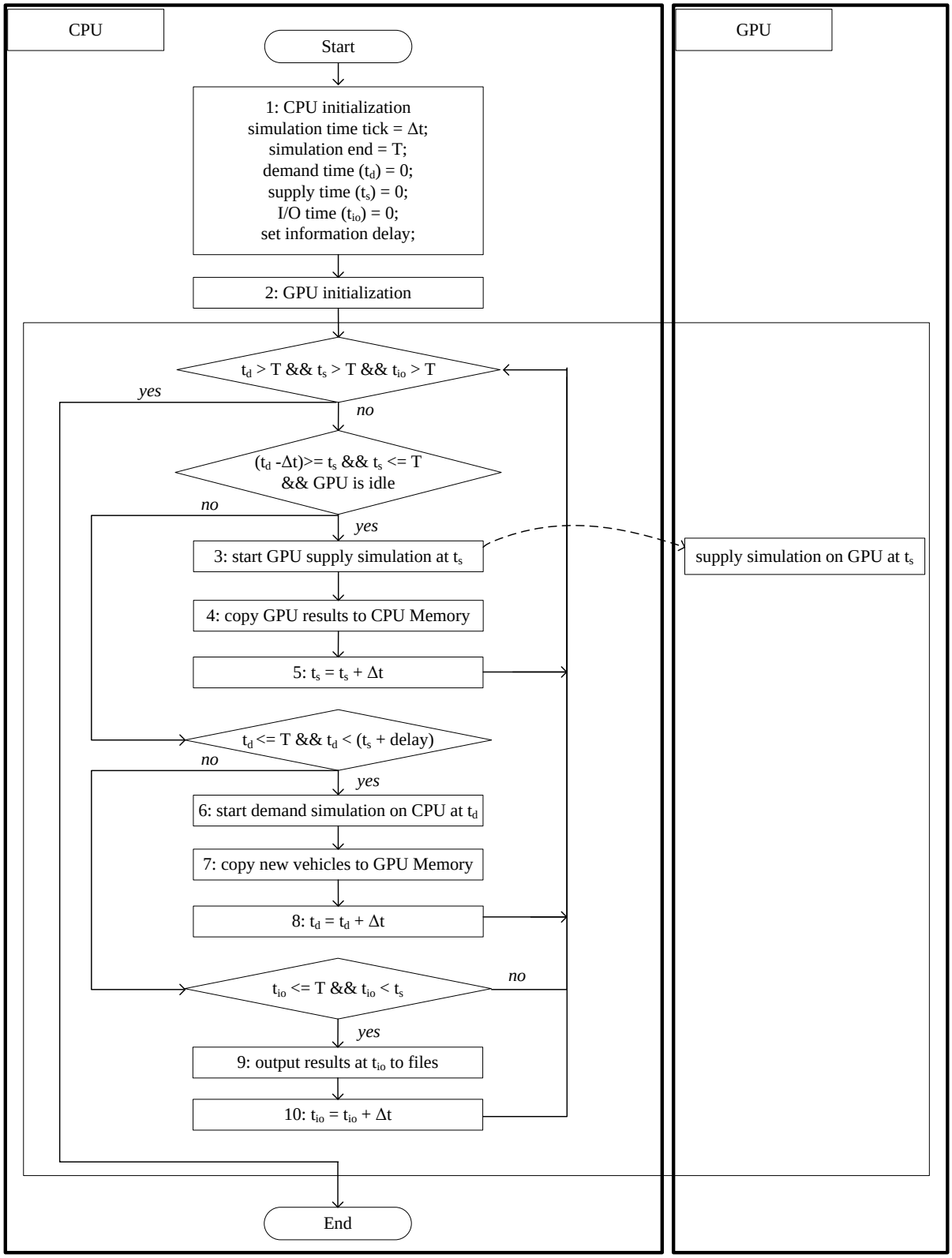


Figure 29: Mesoscopic traffic simulation framework on the CPU/GPU platform

The traditional simulation time, which controls the turnover of the system status, is divided into three components: a demand time step (t_d), a supply time step (t_s) and an I/O time step (t_{io}). A traffic simulation is completed only if t_d, t_s and t_{io} all reach the simulation end. In this framework, these three time steps advance in an asynchronous way. At any instantaneous time, these three time steps can be different. The time management in this framework is controlled by three basic rules:

1) $t_d >= t_s$
2) $t_s >= t_{io}$
3) $t_d <= t_s$ + information delay

First, t_d is always not smaller than t_s, because only if vehicles entering the simulation at time t are generated, the supply simulation at t can start. Second, t_s is always not smaller than t_{io}, because only if the supply simulation at t is completed, the simulation results at t can be outputted to files. The third rule involves a concept in traffic simulation: information delay. Vehicles generated at time *t* requires simulated traffic conditions with a delay (e.g. for route choices). The minimum value of information delay is 1, which means vehicles have real-time instantaneous information about the global traffic status in last time step (e.g. 1 second). However, information delay tends to be larger in real-world traffic systems (e.g. 15 minutes).

In the simulation procedure in Figure 29, step 1 and 2 initialize the required data structures on the CPU and the GPU, including the road network, traffic scenario configurations and other parameters. After initialization, the CPU controls time management. Besides, without breaking the three rules in time management, the following tasks are executed in parallel:

1) The supply simulation at time t_s on the GPU (step 5).
2) The demand simulation at time t_d on the CPU (step 6-8).
3) Write simulation results at time t_{io} to files (step 9-10).

The CPU firstly checks whether the GPU has finished the supply simulation at time t_s. If yes, the simulation results on the GPU (e.g. segment-based speed and density) are copied to the CPU and the supply time t_s is advanced. Then, the supply simulation at next time step is started on the GPU. Note that the CPU will not wait for the GPU supply simulation to finish. If the supply simulation on the GPU is ongoing, the CPU checks whether the demand simulation can be started. If the simulation results required for demand simulation are available, the demand

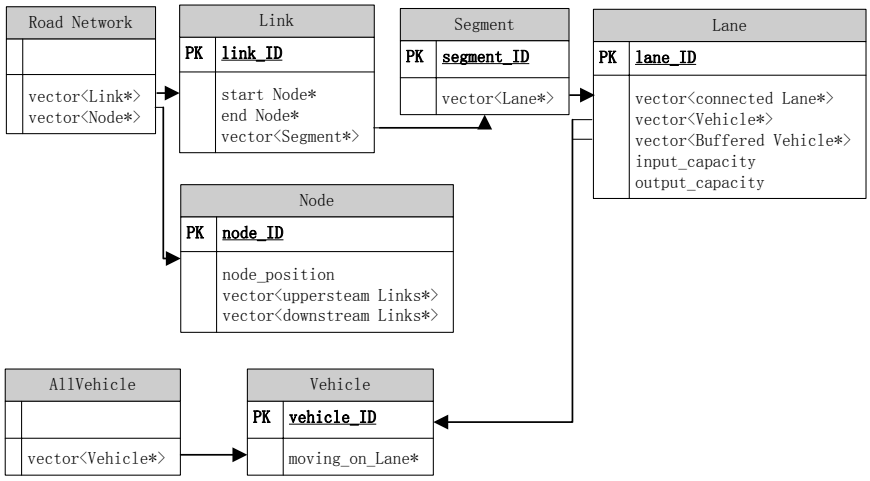
simulation is started on the CPU (e.g. multithreading in the CPU). Otherwise, the CPU checks whether there are available simulation results that need to be written into files. The CPU will continue the loop until the three times t_s, t_d and t_{io} all reach the simulation end. The demand simulation on the CPU was explained in Section 2.1, and the supply simulation on the GPU is explained below in Section 5.2.3.

5.2.2 Road Network and Vehicle Modeling on the GPU

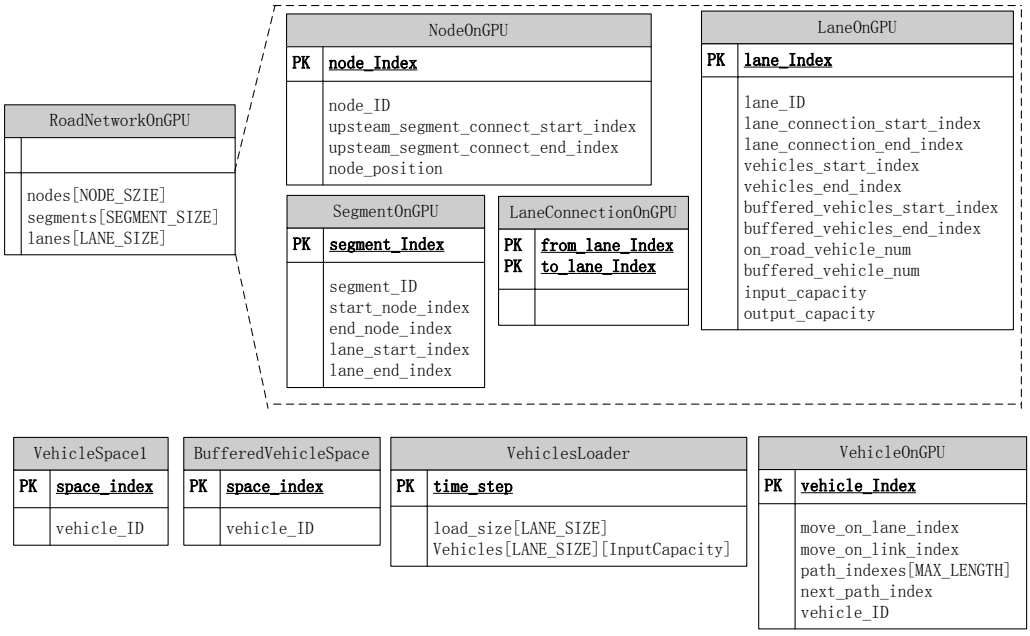
For high performance, the data structure in the GPU memory is completely different from the CPU, due to the thread hierarchy and the memory hierarchy in the GPU. Figure 30(A) shows the road network and vehicle on the CPU memory. A road network is composed of a list of links and a list of nodes. Each link consists of a number of segments and each node consists of a list of upstream links and downstream links. Each segment consists of multiple lanes and each lane contains a number of lane connections. Each lane also has access to vehicles, which are moving on the lane. Figure 30(B) shows a similar road network and vehicle modeling on the GPU memory. The aim of Figure 30 is to show the difference between network modeling and vehicle modeling on the CPU memory and the GPU memory.

There are two key differences between the data structures in the CPU memory and the GPU memory. First, on the CPU memory, the large number of road elements and vehicles are stored in random separated memory spaces and the objects are connecting with each other using pointers. While on the GPU memory, these elements are kept in arrays in a continuous memory space and different elements are connecting each other using the index inside the array. The reasons of doing this on the GPU memory are to make it easy to copy the entire road network from the CPU memory to the GPU memory and more importantly to allow efficient coalesced memory access, which means a group of GPU threads in a warp tend to access continuous memory space. Second, on the CPU memory, dynamic memory allocation (e.g. std::vector, which applies for a memory space when it is immediately required) is widely used in the data structure of a road network and vehicles, because of its flexibility and efficiency. However, on the GPU memory, dynamic memory allocation has to be replaced by static memory allocation, which applies a sufficient large amount of memory space in the beginning. It is a limitation of GPU programming, because it is not efficient to do random memory access. In this framework, it

uses "start & end indexes" to store the containing relationships. For example, as shown in Figure 30(B), each segment has two attributes *lanes_start_index* and *lanes_end_index*, pointing to the indexes of the first inner lane and the last inner lane; each lane has two attributes *vehicles_start_index* and *vehicles_end_index*, pointing to indexes of the first vehicle space and the last vehicle space in the lane. Given an index of a lane and an index of a vehicle, the object can be found in the lane pool and vehicle pool on the GPU memory.



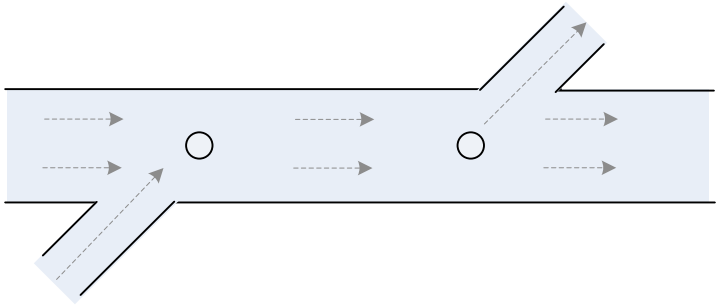
(A) The road network and vehicles modeling on the CPU memory



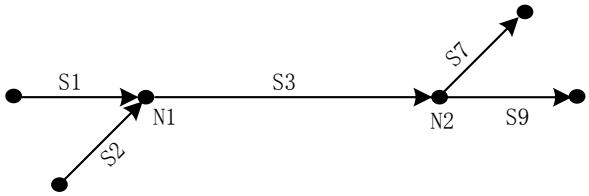
(B) The road network and vehicles modeling on the GPU memory

Figure 30: Road network and vehicles modeling on the CPU and the GPU

An example road network is illustrated to further explain the data structure in the GPU memory. Figure 31(A) represents a real-world road network. The road network consists of a main road with one on-ramp and one off-ramp. The road network is then modeled as 6 nodes and 5 segments in Figure 31(B). The two nodes of interest are node 1 and node 2 (as N1 and N2 in Figure 31(B)). Node 1 has two upstream segments and one downstream segment; node 2 has one upstream segment and two downstream segments. The main road has 2 lanes; the on-ramp and off-ramp roads have 1 lane. The length of segment 3 (S3) is 200 meters; the length of all other segments are 100 meters. The length of a vehicle is 5 meters. Then, the data structure of the road topology in the GPU Memory is shown in Figure 31(C). First, continuous memory spaces (e.g. tables) are used to keep nodes, segments, lanes, etc. Second, each element has a special ID (or index), which indicates its location in the table. For example, the segment S1 is the first element in the segment table. Third, a pair of start/end indexes are used to store the relationship between nodes and segments, segments and lanes, lanes and vehicles. Fourth, complex lane connection rules are directly modeled in this memory model. For example, as shown in the lane-connection table, the on-ramp segment 2 contains only one lane (lane3), and lane3 is only connected to one lane of segment 3 (lane5). It means, in this example, vehicles on the segment 2 cannot pass to the other lane of segment 3. Finally, each lane in segment 3 has a space for 40 vehicles, while the other lanes has a space for 20 vehicles. Thus, each lane in segment 3 reserves 40 vehicle ID space. An attribute "on_road_vehicle_num" is used to determine how many vehicles are on the lane and thus which vehicle IDs are valid.



(A) An example road topology



(B) The simulated road topology in traffic simulation

NodeOnGPU		
ID	up_start_index	up_end_index
N1	S1	S2
N2	S3	S3

SegmentOnGPU				
ID	start_node	end_node	lane_start_index	lane_end_index
S1	N*	N1	L1	L2
S2	N*	N1	L3	L3
S3	N1	N2	L4	L5

LaneConnectionOnGPU		
ID	from_Lane_ID	to_Lane_ID
LC1	L1	L4
LC2	L1	L5
LC3	L2	L4
LC4	L2	L5
LC5	L3	L5
LC6	L4	L*
LC7	L4	L*
LC8	L4	L*

LaneOnGPU				
ID	conn_start_index	conn_end_index	vehicle_start_index	vehicle_end_index
L1	LC1	LC2	V1	V20
L2	LC3	LC4	V21	V40
L3	LC5	LC5	V41	V60
L4	LC6	LC8	V61	V100
L5	LC9	LC10	V101	V140

(C) The data structure in the GPU Memory (* means the ID does not exist in the figure)

Figure 31: An example road network and the data structure in the GPU memory

5.2.3 Supply Simulation on the GPU

The supply simulation on the GPU consists of four kernel functions:

1) cpu_update()
2) pre_vehicle_passing()
3) vehicle_passing()
4) copy_simulation_results_to_cpu()

The first kernel function allows the CPU to change the status of the traffic simulation, before starting the supply simulation at the next time step. For example, if an incident happens, the road capacity is reduced. The CPU updates the new capacity to the road network on the GPU memory before simulating the next time step. Another example is en-trip route choices. En-trip route changing behavior is simulated on the CPU and then the new routes are copied to the GPU.

The second kernel function updates the status of each lane (e.g. density, speed and t_p), before passing vehicles to the downstream lanes. The update unit of this kernel function is a lane, which means each lane is simulated on a GPU thread. This kernel function firstly loads vehicles, which passed to this road at the previous time. After that, it loads new generated vehicles into the lane. Then, the kernel function calculates the speed of the lane based on the speed density relationship (as in Formula 2.2). After that, the kernel function calculates t_p. The method to calculate t_p was explained in Section 3.2.

The third kernel function scans vehicles on the lane and passes a number of downstream vehicles to the next lane. The update unit of this kernel function is a node. Each node and its upstream lanes are simulated on a GPU thread. It is because vehicles from upstream lanes might conflict with each other when crossing the node to the same next lane. One example is shown in Figure 32. In this small road network, node 1 has two upstream lanes: lane 1 and lane 4. Thus, vehicles on lane 1 and lane 4 are processed on the same GPU thread, to remove the potential conflicts. On the other hand, since each lane has only one upstream node, from where vehicles might pass through, there is no conflict when updating nodes in parallel. There are four rules to determine whether a vehicle can pass a lane to the next lane, which were explained in Section 3.2. If a vehicle crosses from the current lane to the next lane, the corresponding output capacity

of the lane, the input capacity and empty space of the next lane are updated. Finally, the vehicle ID should be removed from the current lane and inserted to the next lane.

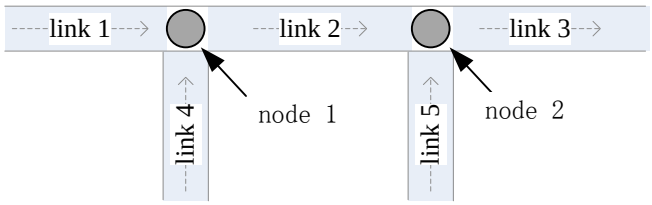


Figure 32: A node and its upstream links are updated on the same GPU thread

The last kernel function copies the simulated results, which include the speed, density, flow, queue length and empty space of each road, from the GPU memory to the CPU memory. As shown in Figure 30(B), these data are stored in a contiguous GPU memory space, in order to reduce the time cost of data transfer from the GPU memory to the CPU memory.

5.2.4 Double-Buffer Data Channel on the GPU

The process of copying simulation results from the GPU memory to the CPU memory at each time step (step 3 in Figure 29) is time costly in the CPU/GPU platform. As shown in Figure 33, a double-buffer data channel is designed on the GPU to minimize the data communication cost. There are two optimizations in the double-buffer data channel. Firstly, the frequency to copy simulation results from the GPU memory to the CPU memory is reduced. For example, in this case, each buffer keeps simulation results of up to 8 time steps (known as "buffer size"). When the buffer is full, the supply simulation on the GPU starts to write simulation results to the other buffer space. At the same time, the simulation results in this buffer are copied to the CPU memory. Secondly, the data transfer from the GPU to the CPU is done asynchronously. It means the supply simulation on the GPU does not wait for the data transfer to finish. However, the double-buffer data channel brings two additional rules in time management.

1) buffer size <= information delay
2) $t_s < t_{io} + 2 * \text{buffer size}$

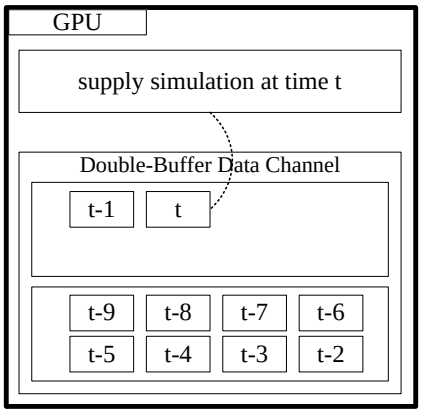


Figure 33: An example double-buffer data channel on the GPU

If the buffer size is larger than the information delay, there will be a deadlock in time management. The buffer waits for additional supply simulation results before transferring its data to the CPU. At the same time, the demand simulation on the CPU is waiting for the simulation results from the GPU to generate future vehicles, and the supply simulation on the GPU is blocked because of the lack of new vehicles. The mutual waiting between the three components is a deadlock. The suggested buffer size should be much smaller than the information delay. In this thesis, a typical information delay is 15 minutes and a typical buffer size is 1 minute (or 60 ticks, if the simulation time step is 1 second). Besides, the supply simulation on the GPU cannot write simulation results to a buffer, if the data in the buffer has not been transferred to the CPU. It means a slow data transfer or I/O will finally force the supply simulation on the GPU to stop and wait.

5.3 Simulation Results on the GPU

5.3.1 Problem Definition

A challenging problem that arise when executing a mesoscopic traffic simulation in a massive parallel way on the CPU/GPU platform is that a mesoscopic traffic simulation in a road network cannot be naturally spatially divided into a large number of independent traffic simulations in sub-networks. The reason is the "upstream-downstream dependence". As shown in Figure 13, when a vehicle crosses from one lane to the next lane, there are four conditions to check: the output capacity of the lane, the input capacity and empty space of the next lane, and the

entry_time of the vehicle. The requirement to know the input capacity and empty space of downstream lanes when simulating traffic movement in a lane is defined as "upstream-downstream dependence".

When a mesoscopic traffic simulation is executed in a sequential way, the order of lanes is sorted in a order that downstream lanes are processed before upstream lanes. Thus, the input capacity and empty space of downstream lanes are known when processing a upstream lane. However, when a mesoscopic traffic simulation is executed in a parallel way, lanes are processed simultaneously without any order. It becomes a challenge to comply with upstream downstream dependencies. As explained in the third kernel function in Section 5.2.3, each node and its upstream lanes are simulated on a GPU thread. When processing a vehicle passing from a lane to the next, the output capacity of the lane is known. Besides, the downstream lane is processed by only this GPU thread. It means there is no data race to access input capacity of the downstream lane and thus the input capacity of the downstream lane is known. However, the empty space of the downstream lane depends on the traffic movement of the downstream lane at the same time step, which is unknown. Without a perfect knowledge of empty spaces in downstream lanes, the upstream lane has to move vehicles based on its best estimation. This could lead to two types of unrealistic vehicle movements (Wen, 2009):

Pessimistic biased movement: If vehicles' movement in a upstream lane is based on overly conservative estimate of the downstream empty space, when the empty space of the downstream lane at time *t* is estimated to be the empty space of that link at the previous time *t-1*, vehicles might move slower than the sequential simulation.

Optimistic biased movement: If vehicles' movement in a upstream lane is based on overly optimistic estimate of the downstream empty space, when the empty space of the downstream lane is assumed to be always large enough to allow new vehicles, the simulator may fail to capture exactly the same queuing and spill-backs from the sequential traffic simulation.

5.3.2 Boundary Processing Method

The concept of boundary processing has been introduced in Section 2.3.4. The boundary area means a portion of a road network which connects the traffic flow from different partitions. In most cases, traffic movement in a boundary area requires a different procedure compared with

traffic movement on a normal road. In this thesis, when running traffic supply simulation on the GPU and each node of a road network is simulated on a separate GPU thread, the boundary area is in fact the entire road network.

The empty space of a lane is calculated using Formula 5.1. The empty space at a time step *t* depends on three variables: the empty space of the lane at the previous time step *t*-1, the speed of the lane at time *t* and the queue length of the lane at *t*.

$$empty\_space(t) = \min\{empty\_space(t-1) + v(t)*\Delta t \, , \, road\_length - q(t)\} \quad (5.1)$$

where, Δt is the simulation time step, $(empty\_space(t-1) + v(t)*\Delta t)$ reflects the traffic movement on the lane, and $(road\_length - q(t))$ reflects the queue feedback of the lane. The key idea of the proposed boundary processing method is to share the speed of each lane before calculating the empty spaces. First, given speeds of downstream lanes, it becomes easier for the upstream lane to estimate empty-spaces of downstream lanes and reduce the simulation error. Second, speed and empty-space are calculated in different kernel functions. Thus, there is no additional barrier in the proposed boundary processing method. The secondary idea of the proposed boundary processing method is to predict queue lengths of lanes in the current next step using Formula 5.2:

$$q(t) = \max\{q(t-1) + a*(q(t-1) - q(t-2)) \, , \, road\_length\} \quad (5.2)$$

where, $q(t-1) - q(t-2)$ reflects a short-term trend of the queue length in the last two time steps and *a* is a parameter, which indicates how strongly the predicted queue length depends on the short-term trend.

Simulation results of the proposed mesoscopic traffic simulations on the CPU/GPU platform using the proposed boundary processing method are different from simulation results of a serial traffic simulation on the CPU platform, only if:

1) The estimated empty-space of a lane using Formula 5.1 is wrong.
2) The empty-space becomes a key factor to determine whether a vehicle can pass lanes.

First, the estimated empty-space of a lane will be wrong, only if the empty-space is dominated by its queue length and the predicted queue length is different from the true queue length. Second, the four factors to determine whether a vehicle can pass lanes were introduced in Section 3.2. The empty-space of a downstream lane becomes a key factor, only if the empty-space is smaller than the required space from the upstream lanes. For example, if the simulation time step is 1

second, the input capacity of a lane is smaller than 1 vehicle per time step. So, the empty-space is a key factor only if the empty-space is smaller than the length of a vehicle (e.g. 5 meters).

5.3.3 The Rollback Method

If simulation results of the proposed mesoscopic traffic simulations on the CPU/GPU platform are different from simulation results of a serial traffic simulation on the CPU platform, a rollback method is triggered. The rollback method is an important component of the proposed traffic simulation framework on the CPU/GPU platform and it guarantees the correctness and therefore the validity of the proposed framework.

In the rollback method, the estimated empty space and the "true" empty space are compared at the end of each simulation time. If the empty-space of a downstream lane is used as a key factor to determine whether a vehicle can pass lanes, and the used empty space and the "true" empty space of the downstream lane are different (e.g. the "true" empty space allows one more vehicles to pass through), a rollback method is triggered, using the "true" empty spaces in replacement of the estimated empty-spaces. An example is shown in Figure 34. In this example, the estimated empty-space of lane 2 is wrong and then it forces the most downstream vehicle in lane 1 to stay in lane 1, even though the "true" empty space is enough to keep one more vehicle. The rollback method is triggered to correct the wrong behavior.

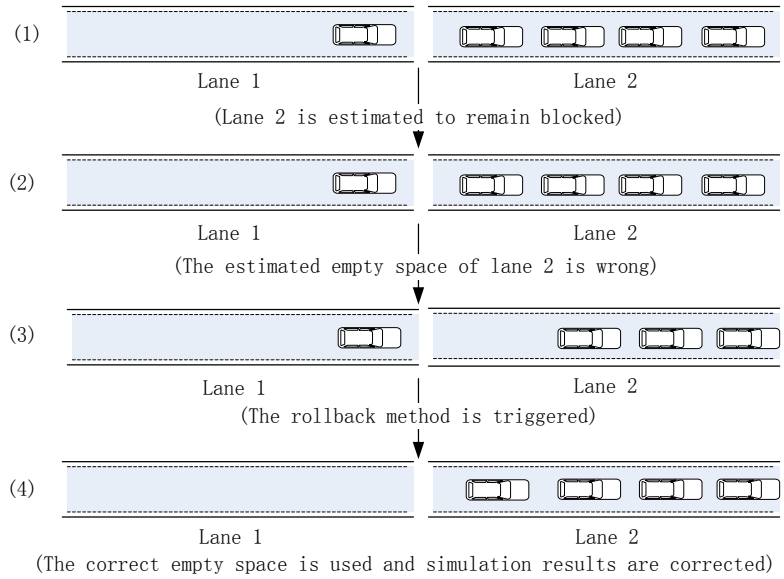


Figure 34: The Rollback Method

5.4 Synthetic Tests

The performance of the mesoscopic traffic simulation framework on the CPU/GPU platform is investigated in a synthetic test, before it is evaluated in a real-world city-scale traffic simulation.

5.4.1 Experimental Design

An artificial large grid road network is used as the testbed, with 10201 nodes and 20200 unidirectional links. Each node has an index from 0 to 10200, indicating the store location on the GPU memory. Each link also has an index from 0 to 20199. 100,000 vehicles are loaded into the road network during 1000 simulation time steps (each time step is 1 second). Vehicles are loaded into the road network from nodes in the top and the left, which are moving to the bottom and the right. Each vehicle randomly picks a route from the pre-calculated candidate routes. En-trip route choice is not simulated in this traffic scenario. The traffic scenario is simulated on two types of platforms: the CPU platform and the CPU/GPU platform. Only the total time cost of the supply simulation during the 1000 simulation ticks is measured in this experiment. The CPU platform includes an Intel E5-2620, 32 GB main memory and a 500GB SATA 7.2K RPM. The CPU/GPU platform includes the CPU and an additional GPU, which is a GeForce GT 650M. The GPU has 384 CUDA cores and 2 GB global memory. The supply simulation on the CPU and the GPU follow the same logic. The source codes are both implemented using C++ on Ubuntu 12.04 and compiled using g++_4.6.3 and CUDA 5.5. The release version executable file is used.

5.4.2 Speedup and Analysis

The results are shown in Table 13. Five configurations are investigated. Each configuration is executed 5 times and the average time cost is shown. The speedup is measured by comparing the time cost of supply simulation on a GPU to the time cost of supply simulation on a CPU core. In the first configuration, executing the supply simulation on a CPU core takes 4720.88 ms. In the second configuration, directly executing the proposed supply simulation on a GPU takes 704.72 ms, which means a speedup of 6.7. The performance is sensitive to the configuration of the kernel functions. When the number of threads in a block is 192, the maximum performance is achieved. In the third configuration, the double-buffer data channel on the GPU is enabled to allow asynchronous data transfer between the CPU and the GPU. We found that the double-

buffer data channel is efficient and the data transfer cost is almost hidden by the supply simulation on the GPU. The total time cost is reduced by 35% and the speedup is significantly improved from 6.7 to 10.3. After that, we found that the registers are not efficiently used in the third configuration. In the fourth configuration, internal variables, which will not be shared between kernel functions and not transferred to the CPU, are moved from the global memory to the registers. The speedup is slightly improved to 10.7. Finally, there are parameters which are never changed during the traffic simulation, such as the length of a simulation time step. In the fifth configuration, constant parameters are moved from the global memory to the constant memory for efficient memory access. The speedup is slightly improved to 11.2.

Table 13: The time cost of running ETSF on the CPU and the GPU

Case ID	Configuration Description	Time Cost (ms)	Speedup
1	Supply simulation on a CPU core	4720.88	1.0
2	Supply simulation on the GPU	704.72	6.7
3	Case 2 + Double-buffer data channel optimization on the GPU	457.29	10.3
4	Case 3 + Push internal variables to the registers	439.29	10.7
5	Case 4 + Push constant parameters to the constant memory	423.37	11.2

Table 14 shows the profiling of two kernel functions in the framework: *pre_vehicle_passing* and *vehicle_passing*. First, the simulation units in these two kernel functions are roads and nodes. In this traffic scenario, these two kernel functions launched 20200 GPU threads and 10201 GPU threads. The occupancies of these two kernel functions are high, which indicates the GPU cores are fully utilized. Besides, even after moving internal variables from the global memory to registers, registers are not a bottleneck. Second, as explained in Section 5.2.2, the road network and vehicles are stored into a contiguous memory space. Thus, threads in a warp tend to access a contiguous memory space, which is also known as coalesced memory access. The number of memory transaction per request (both load and store) for these two kernel functions are small (<2), which indicates the memory access is well coalesced. Third, the branch taken ratio (within threads in the same warp) for the first kernel function is 72%. Threads in the first kernel function do not take exactly the same branch, because roads have different number of

entering vehicles and different number of passed vehicles. The branch taken ratio for the second kernel function is much lower (41.7%). It is expected, because in the second kernel function the source code to find and pass vehicles to downstream roads varies in different types of road topologies. The number of upstream links and the number of vehicles on nodes are different. However, there is not much branch divergence in these two kernel functions. The instruction serialization ratio of these two kernel functions are 15.5% and 18.6%. It means that the branch taken ratio does not cause performance loss in this case. Fourth, the numbers of instruction per clock (IPC) for these two kernel functions are 0.9 and 1.0, which are far below the hardware's peak value (4.0). As shown below, the major reason for low instruction issue efficiency is the execution dependency. Based on our knowledge of the framework, it is because most data (e.g. the road network and vehicles) is stored in the global memory, which causes high memory access latency. The memory access latency is not completely hidden by thousands of threads. Finally, the achieved GLOPS for the two kernel functions are also lower than the hardware's peak, which is also related to high global memory latency.

Table 14: Profiling of major GPU/CUDA kernel functions

ID	Measurement	kernel function: *pre_vehicle_passing*	kernel function: *vehicle_passing*
1	Launched GPU threads	20200	10201
2	GPU occupancy	81%	90%
3	Registers (used / available)	3072 / 65536	1920 / 65536
4	Transaction per request (load/store)	1.73/1.49	1.67/1.83
5	Branch taken ratio (%)	72.1%	41.7%
6	Instruction serialization	15.5%	18.6%
7	Instruction per clock (IPC) (measurement/ maximum)	0.9 / 4.0	1.0 / 4.0
8	Warp issue efficiency (no eligible %)	49.7%	36.1%
9	Issue Stall Reasons (execution dependency)	92.3%	88.4%
10	CUDA achieved GFLOPS	14.8	6.0

5.5 Case Study

In this section, the proposed ETSF framework on the CPU/GPU Platform is evaluated on a real-world city-scale traffic road network.

5.5.1 Experimental Setting

As shown in Figure 20, the Singapore expressway road network is used in this section. The Singapore expressway system is modeled as a network with 831 nodes, 1040 links, and 3388 segments. The demand is modeled as trips in 4106 OD pairs. Origins are set to be the beginning of on-ramps where traffic gets onto the expressway system from local roads and destinations are set to be the end of off-ramps where traffic gets out of the expressway system. The simulation time step is 1 second. and the simulation period is 60 minutes. The calibration of the OD Matrix is explained in Section 6.5, and the calibrated OD Matrix from 7:00AM to 8:00AM is used in this scenario. In particular, there are in total 106,386 vehicles loaded into the traffic scenario, and departure times of these vehicles are uniformly distributed. After a vehicle is generated, the vehicle chooses a route using the Path Size Logit model (Ben-Akiva et al., 1999). En-trip route choice is not simulated in this traffic scenario. As with the synthetic test, the traffic scenario is simulated on two types of platforms: the CPU platform and the CPU/GPU platform.

5.5.2 Results

The supply simulation of this traffic scenario takes 3448.0ms on the CPU and 894ms on the GPU, which means a speedup of 3.86. Compared with the speedup (11.2) on the artificial large grid road network, the speedup on the Singapore expressway is much lower. There are three major reasons. First, in the artificial grid road network, roads have the same length and vehicles on roads are directly stored inside roads. However, in the Singapore expressway, roads have very different lengths from 50 meters to 2000 meters (see Figure 21). Instead of directly storing the vehicles, the "start & end indexes" was introduced in Section 5.2.2. The "start & end indexes" makes the usage of memory space more flexible. It is important for loading the whole city-scale traffic simulations into the GPU memory. However, accessing a vehicle on a road requires twice memory accesses. More memory accesses make the proposed framework less efficient. Second, the number of GPU threads (3388) launched on the Singapore expressway is smaller than the

grid road network (20200). It makes the GPU occupancy lower (65%). Third, the artificial grid road network is more structured. For example, most nodes have two downstream links and 2 upstream links. However, the Singapore expressway network topology is more complicated. It makes the memory access less coalesced.

5.5.3 Discussions

This section discusses additional thoughts about running mesoscopic traffic simulations on the CPU/GPU platform. First, it is beneficial to run the demand simulation on the CPU, the supply simulation on the GPU and the data communication between the CPU and the GPU in an asynchronous way. In the proposed framework, the supply simulation on GPU is the bottleneck and the time costs of the other two tasks are almost hidden. Second, this thesis demonstrates a supply simulation framework (ETSF) on the CPU/GPU platform. Running the ETSF supply framework on the GPU gets a speedup from 3.5 to 11.2, compared with running the same logic on the CPU. However, when generalizing the conclusion to other supply simulation frameworks, it should be noted that ETSF naturally guarantees a good load balance on each road. In ETSF, the workload of a road is not sensitive to the number of vehicles on the road and the length of the road. This feature may not exist in other supply simulation frameworks, in which case, load balancing should be considered. Third, the memory access latency is a bottleneck in the proposed mesoscopic traffic simulation framework. In mesoscopic simulation frameworks, the update of a road depends on majorly its own road status (e.g. road density and queue status) and requires a small number of parameters from its downstream roads. There is few shared data access among nearby roads and nodes, which limits the usage of the more efficient shared memory in the GPU. We think this is the bottleneck that prevents us from achieving a higher simulation speedup on the GPU.

5.6 Summary

The GPU is gaining popularity, because of its massive performance compared to the CPU. In this section, we investigated whether the GPU can be a potential high-performance platform for future mesoscopic traffic simulations. First, we proposed a comprehensive mesoscopic traffic

simulation framework on the CPU/GPU platform. After that, we designed a boundary processing method and a rollback method to guarantee the correctness of simulation results. Then, the performance of the proposed simulation framework was evaluated in an artificial grid road network (getting a speedup of 11.2) and also a real-world city-scale traffic scenario (getting a speedup of 3.86), compared with running the same logic on the CPU. Based on our view, the proposed mesoscopic traffic simulation framework on the CPU/GPU platform provides an innovative and potential solution for high-performance mesoscopic traffic simulations.

Further research on the topic of traffic simulation on the CPU/GPU platform includes four directions. First, the proposed framework needs to be improved to make better use of the shared memory in the GPU. Second, the performance cost of the rollback method in congested traffic scenarios needs to be further investigated. Third, the proposed framework needs to be expanded to support the multi-GPU platform. Finally, the proposed framework needs to be modified if the CPU and the GPU are using shared global memory in future.

6. An Enhanced Calibration Algorithm for City-scale Traffic Simulation

The state-of-the-art algorithm *simultaneous perturbation stochastic approximation* (SPSA) proposed a comprehensive gradient descent calibration framework to calibrate all types of variables simultaneously using multiple data sources. However, we found that the SPSA algorithm deteriorates when the scale of the problem becames larger, in terms of the network size and the length of the simulation period. This chapter begins with a problem formulation of calibration. Then, an in-depth analysis of the state-of-the-art SPSA algorithm is presented. Finally, an enhanced calibration algorithm[1] for city-scale traffic simulations is introduced, investigated and evaluated.

6.1 Problem Formulation

Let the time period of interest be divided into intervals $H = \{1, 2, ..., h\}$. The calibration problem is formulated using the following notations:

- x: calibration variables, e.g. OD flows, parameters in route choice models.
- x^a: Priori values of calibration variables.
- M^o: Observed aggregate time-dependent measurements
- M^s: Simulated aggregate time-dependent values
- G: Road network

[1] The enhanced calibration algorithm is a collaborated research work (Lu et al., 2014). The author of this thesis contributed in the idea of 0/1 weight matrix, data cleaning method and also the real-world case study.

The off-line calibration problem is formulated as the minimization of an objective function over the variable space:

$$z(\mathrm{x}) = \mathrm{z}_1(\mathrm{M}^o, \mathrm{M}^s) + \mathrm{z}_2(\mathrm{x}, \mathrm{x}^a); \tag{6.1}$$

subject to the following constraints:

$$\mathrm{M}^s = simulate(\mathrm{G}, \mathrm{x}); \tag{6.2}$$

$$l_x \le x \le u_x; \tag{6.3}$$

Equation 6.1 is the objective function, where $\mathrm{z}_1()$ is a function which measures the difference between observed measurements and simulated values, $\mathrm{z}_2()$ is a function which measures the difference between estimated values and priori values. The value of M^s requires one execution of traffic simulation. Besides, in many cases, the number of calibration variables is larger than the number of independent observed measurements. Thus, priori values and bounds (upper bounds and lower bounds) are introduced to constrain the calibration algorithm. Note that x, x^a, M^o, M^s are vectors, which contains values of different variables at different locations and different time intervals.

6.2 The SPSA Algorithm

Stochastic approximation (SA) methods are a family of iterative stochastic optimization algorithms used in error function minimization when the objective function has no known analytical form and can only be estimated with noisy observations. It iteratively perturbs a sequence of variables in different directions and estimates the gradients for these variables until converging to an acceptable objective function value. The general iterative form of SA is:

$$x_{k+1} = x_k - a_k g_k(x_k) \tag{6.4}$$

where x_k is estimated values of calibration variables in the k_{th} iteration, $g_k(x_k)$ is estimated gradient of calibration variables in the k_{th} iteration, a_k is the movement step size in the k_{th} iteration. The value of a_k gets smaller as k becomes larger. One example is shown below.

$$a_k = \frac{a}{(\mathrm{A}+\mathrm{k})^b} \tag{6.5}$$

where A and b are algorithm parameters.

The estimation of gradients is one critical step in calibration. Different approaches have been proposed to approximate gradients. In the finite-difference method (FDSA), the gradient is estimated by perturbing the variables in the decision vector one at a time, evaluating the object function values and computing the gradient as:

$$g_{ki}(x_k) = \frac{z(x_k + c_{ki}) - z(x_k - c_{ki})}{2c_{ki}} \tag{6.6}$$

where $g_{ki}(x_k)$ is the estimated gradient of the i_{th} variable, c_{ki} is the perturbation size of the i_{th} variable, z() is the target function (as shown in Formula 6.1). A simple example is given to understand the usage of $g_{ki}(x_k)$. Say: x_k =6, a_k =1, c_{ki} =1, z($x_k + c_{ki}$) = z(7) = 2 and z($x_k - c_{ki}$) = z(5) = 4. Then, the gradient $g_{ki}(x_k)$ is -1 and the calibrated value of x_{k+1} is 7. Note that a smaller z() value (or the target function value) means a better fit to observed measurements. Thus, the calibrated set of variables tends to reduce the target function value. This approach is capable of estimating high quality gradient vectors in non-analytical problems with noisy observations. It is, however, not efficient for city-scale traffic simulations. The number of objective function evaluations within one algorithm iteration is 2P, where P is the total number of variables in the decision vector. In simulation-based calibration algorithms, one objective function evaluation involves a run of the city-scale traffic simulation, which may take minutes or hours. Considering there are a large number of calibration variables (e.g. the OD Matrix) in city-scale traffic simulations, the method becomes infeasible.

Spall (1992, 1994, 1998, 1999) proposes an innovative solution to the calibration of city-scale traffic simulations: simultaneous perturbation stochastic approximation (SPSA). SPSA efficiently estimates the gradient by perturbing all the variables in the decision vector simultaneously and the approximation of gradient needs only two function evaluations regardless of the number of variables:

$$g_{ki}(x_k) = \frac{z(x_k + c_k) - z(x_k - c_k)}{2c_{ki}} \tag{6.7}$$

Formula 6.6 and 6.7 are similar. The only difference between FDSA and SPSA is c_{ki} and c_k. In FDSA, the estimation of the gradient of each calibration variable requires two runs of the traffic simulation, while in SPSA, the estimation of gradients of all calibration variables requires constant (only two) runs of the traffic simulation. SPSA provides a huge saving of computational time. A random perturbation vector (Spall, 1998) is also introduced into the SPSA algorithm to allow the perturbation of a portion of parameters. In terms of convergence performance, Spall (1998) argues that SPSA follows a path that is expected to deviate only slightly from that of FDSA. In other words, SPSA performs as good as FDSA. SPSA may have approximated gradients that differ from the true gradients, but they are almost unbiased.

SPSA and its variations have been applied extensively in calibration. Ma et al. (2007) compare the performance of SPSA against the genetic algorithm (GA) and the trial-and-error iterative adjustment algorithm (IA) for the calibration of a microscopic simulation model in a northern California network and conclude that SPSA can achieve the same level of goodness-of-fit as the other two do but it has a significantly shorter running time. Lee and Ozbay (2008) propose a Bayesian calibration methodology and applied a modified SPSA algorithm to solve the calibration problem of a cell transmission based macroscopic simulation model. Paz et al. (2012) calibrate all the parameters in CORSIM models simultaneously using SPSA and demonstrate its effectiveness.

These applications of SPSA, however, are all limited to networks with small scale in terms of size and number of time intervals. In a case study for the entire expressway system in Singapore, which is discussed in detail in Section 6.5, it was found that although SPSA kept its computational efficiency, its performance in terms of convergence rate and long run goodness-of-fit deteriorated significantly when the problem scale increased. The target function stopped to decrease at relatively high values. Different values of algorithm parameters and step sizes were implemented. However, no significant improvement was made. The next section analyzes the reasons for SPSA's performance deterioration.

6.3 Motivation and The W-SPSA Algorithm

The reason for the performance deterioration of SPSA in the calibration of city-scale traffic simulations was found to be the way SPSA estimated gradients. In each iteration of the SPSA algorithm, the gradient estimation process essentially tries to find a direction for each variable value to move. This is achieved by comparing the influences to the system caused by perturbing the variable values in two opposite directions. Given our formulation in equation 6.7, in SPSA the influences caused by perturbing the value of a specific variable are determined by the target function (or the sum of all the distances between model outputs and corresponding observed measurements). The key question is that the change of the target function may or may not be sensitive to all variables. This problem can be alleviated by perturbing a small portion of variables, however, it will significantly downgrade the calibration speed of SPSA.

This may not be a major issue in systems where each variable is highly correlated to most of the observed values. However, in the calibration of city-scale traffic simulations, correlations between model parameters and observed values are mostly sparse. Figure 35 shows a small network for the illustration of gradient estimation error in SPSA. In the network, there is 1 OD pair (A -> B) and 6 sensors. Assume the length of an interval is 15 minutes and the simulation period consists of 4 intervals (60 minutes). Assume there is no congestion and the longest travel time in this network is 15 minutes. Therefore in this small calibration problem, there are 4 OD flows to calibrate and there are 24 observed values. In SPSA, gradients of these 4 OD flows are calculated based on the aggregate difference of these 24 observed values and 24 simulated values. However, in this example, the value of each OD flow only affects the values of sensors 3 and 4 during the current and the next intervals. It means the gradient of each OD flow should only be related to 4 observed values, instead of 24 observed values. In this thesis, incorporating unrelated observed values into the gradient estimation of a calibration variable is called as the systematic error in SPSA. W-SPSA is proposed as an enhancement of SPSA, in order to remove the systematic error, without downgrading the calibration speed of SPSA.

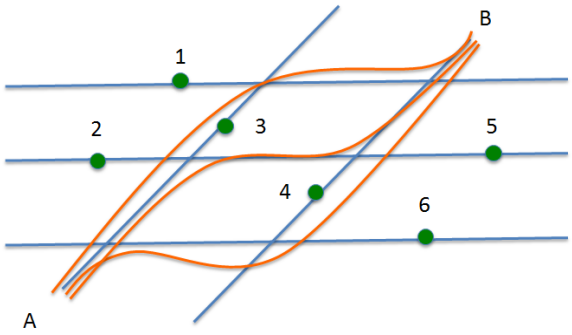


Figure 35: A small network for the illustration of gradient estimation error in SPSA

The idea of Weighted-SPSA (or W-SPSA) is to introduce a 2D weight matrix in the process of gradient estimation in SPSA. The 2D weight matrix represents the relationship between calibration variables and observed values. Continuing the example in Figure 35, the 2D weight matrix is shown in Table 15. The vertical dimension represents calibration variables while the horizontal dimension represents observed values. AB_{ti} means the OD flow from A to B in the *i*th time interval. S_j,t_i means the value of sensor *j* in the *i*th time interval. Values in the 2D weight matrix are either 1 or 0, which means the corresponding calibration variable and observed value are correlated or not. In this example, only 15% values in the 2D weight matrix are 1.

Table 15: An example 2D weight matrix in a small example network

	$S_{1,t1}$	$S_{2,t1}$	$S_{3,t1}$	$S_{4,t1}$	$S_{5,t1}$	$S_{6,t1}$	$S_{1,t2}$	$S_{2,t2}$	$S_{3,t2}$	$S_{4,t2}$	$S_{5,t2}$	$S_{6,t2}$
AB_{t1}	0	0	1	1	0	0	0	0	1	1	0	0
AB_{t2}	0	0	0	0	0	0	0	0	1	1	0	0
AB_{t3}	0	0	0	0	0	0	0	0	0	0	0	0
AB_{t4}	0	0	0	0	0	0	0	0	0	0	0	0
	$S_{1,t3}$	$S_{2,t3}$	$S_{3,t3}$	$S_{4,t3}$	$S_{5,t3}$	$S_{6,t3}$	$S_{1,t4}$	$S_{2,t4}$	$S_{3,t4}$	$S_{4,t4}$	$S_{5,t4}$	$S_{6,t4}$
AB_{t1}	0	0	0	0	0	0	0	0	0	0	0	0
AB_{t2}	0	0	1	1	0	0	0	0	0	0	0	0
AB_{t3}	0	0	1	1	0	0	0	0	1	1	0	0
AB_{t4}	0	0	0	0	0	0	0	0	1	1	0	0

After obtaining the weight matrix, the gradient estimation of variable *i* in the *k*th iteration in W-SPSA is done using the following formula:

$$z_i(\mathrm{x}) = \mathrm{z}_1(\mathrm{M}^o, \mathrm{M}^s, \mathrm{W}_i) + \mathrm{z}_2(\mathrm{x}, \mathrm{x}^a) \tag{6.8}$$

$$g_{ki}(x_k) = \frac{z_i(x_k + c_k) - z_i(x_k - c_k)}{2c_{ki}} \tag{6.9}$$

where $z_i(\mathrm{x})$ is the object function of variable *i*, W_i is the correlation between the *i*th variable and all observed values. Note that different calibration variables use different target functions when calculating gradients, while calculating these target functions requires executing the traffic simulation only twice. In the example network, it means the estimated gradient of AB_{t1} is only related to values of sensors {3 and 4} in intervals {1 and 2}, which removes the systematic error in SPSA.

The weight matrix is a qualitative (not quantitative) measurement of the correlation between calibration variables and observed values. There are two major benefits. First, a quantitative correlation between calibration variables and observed values is not easy to estimate. Second, a quantitative correlation between calibration variables and observed values is dynamic during different time intervals. It makes it even harder to estimate the weight matrix. Comparably, a qualitative (0 or 1) correlation is more easy-to-generate and more stable. However, a quantitative correlation, if available, is always beneficial in W-SPSA.

In this thesis, two approaches are proposed to generate the weight matrix: analytical approximation and numerical approximation. Analytical approximation is an approach to estimate the weights based on available network knowledge (e.g. network topology, paths choice set, equilibrium link travel time, route choice model) and simplified traffic assignment methods (e.g. assumed divergence ratios and free-flow speeds). Numerical approximation uses the traffic simulator to approximate the weight matrix. A number of simulation runs (with different calibration variables or different random seeds) are started. After running N independent experiments, the estimated weights are obtained by averaging the result (or correlation) from each experiment. If the averaged correlation is larger than a pre-defined threshold δ (e.g. 10%), the weight is 1; otherwise, the weight is 0. However, the best way to estimate weight matrices is problem-specific and requires a good understanding of the characteristics of the problem, in-depth analysis of variables and measurements, and some engineering judgments.

6.4 Synthetic Tests

The effectiveness and sensitivity of W-SPSA is investigated in a synthetic test, before it is evaluated in the calibration of a city-scale traffic simulation.

6.4.1 Experimental Design

In the synthetic test system, time-dependent OD flows are calibrated using time-dependent traffic counts. Instead of using a traffic simulator to evaluate OD flows and get its simulated counts, linear functions are used to map OD flows to sensor counts. The true values of the OD flows are randomly generated numbers with magnitudes comparable to real world cases. The initial OD flows (prior values) are generated by randomly perturbing the true OD flows. The observed sensor counts are computed using the true OD flows and randomly generated linear relationships:

$$M_j = \sum_{i=1}^{p} \beta_{ij} x_i \tag{6.10}$$

where x_i is the i_{th} time-dependent OD flows in one interval, M_j is the j_{th} observed sensor value, β_{ij} is a randomly generated coefficient according some rules. If β_{ij} is 0, it means M_j and x_i have no correlation; otherwise, M_j and x_i are correlated. The OD flow in one interval is assumed to be only related to sensor counts in the same interval. It means M_j and x_i in different intervals are not correlated. The dimension of the problem is 1000 OD pairs and 100 sensors, with, at every interval, one value for each OD flow and sensor count. The ratio between the number of OD pairs and sensors is comparable to the real world case studies (e.g., in Balakrishna (2006)).

There are two additional concepts: variable correlation and network correlation. Variable correlation quantifies the degree a variable correlates to all the observed values. The network correlation is the average over all variable correlations in the network. All variables have different variable correlations generated randomly with a mean value that equals the network correlation. The variable correlations are limited to a range centered at the network correlation. For example, if the network correlation is 10%, each OD flow is randomly correlated to 1-20

sensors and weights are randomly generated values between 0.1 and 1 (the weights for uncorrelated sensors are 0).

The root mean square normalized error (RMSN) is used to measure the goodness-of-fit of the calibration algorithm:

$$RMSN = \frac{\sqrt{S\sum_{i=1}^{S}(y_i - y_i)^2}}{\sum_{i=1}^{S} y_i} \tag{6.11}$$

where y_i is the ith observed value, y_i is the corresponding simulated value. S is the total number of observed values. The experiments are done in the Matlab (R2011b) environment. Part of the algorithm parameters in both SPSA and W-SPSA were determined based on Balakrishna (2006): b = 0:602, γ = 0:101, A = 50. For *a* and *c*, which are the perturbation and advance step size, linear search was performed to find their best values for each case in the experiments. The following dimensions are considered in these experiments:

- Algorithm: SPSA vs. W-SPSA;
- Number of time intervals: 1, 2, 4, or 8 intervals were considered;
- Weight matrix: five different types of weight matrices were considered: (i) a "perfect" matrix, which has a quantitative correlation between variables and measurements, (ii) a weight matrix in which all correlations were converted to 0 or 1, (iii) a weight matrix in which all correlations were converted to 0 or 1 and then 30% received the wrong side (from 0 to 1), (iv) same as (iii) but the 30% that received the wrong value were from 1 to 0, (v) all correlations equal to one (equivalent to SPSA);
- Network correlation: the degree of network-scale average correlation between ODs and measurements, taking the following values: 10%, 20%, 40%, 80%.

6.4.2 Effectiveness

This first experiment tests the scalability of SPSA and W-SPSA by comparing their performance. The network correlation is 10%. Random weights are assigned to the sensors between 0.1 to 1. In W-SPSA, accurate weight matrices are known and used. The results are shown in Figure 36, where the horizontal axis represents the number of algorithm iterations during the calibration process, and the vertical axis represents the RMSN after a specific number of iterations.

W-SPSA outperforms SPSA in terms of convergence rate and final achieved goodness-of-fit independently of the number of intervals. The convergence rate of SPSA and W-SPSA decrease with the increase of the number of intervals, because of increased problem scale. However, W-SPSA achieves similarly good long run goodness-of-fit with different number of intervals. The results show that while SPSA is scalable in term of computational time (two objective function evaluations regardless of the problem scale), it is not scalable enough in terms of convergence and goodness-of-fit. W-SPSA achieves significantly better scalability in both computational performance and goodness-of-fit. The additional calculation in the gradient approximation process of W-SPSA is negligible compared to the running time of the simulator.

6.4.3 Sensitivity

Accurate weight matrices are extremely hard to get in real-world traffic systems. In the second experiment, inaccurate weight matrices are investigated. Tests were again performed assuming 1000 OD pairs and 100 measurements per interval. Each OD pair was correlated with 1 to 20 measurements (the number was uniformly distributed). 4 intervals were considered in this test. Figure 37 shows the performance of W-SPSA using inaccurate weight matrices. The dark blue line represents SPSA. The light blue line corresponds to the performance of W-SPSA with the perfect weight matrix. The red line is the result when the magnitude information of the weights was not available and only 0 or 1 was assigned to the weight matrix based on the knowledge about correlated or uncorrelated observation. For the pink line, 30% of the uncorrelated pairs were mistakenly decided to be correlated and therefore assigned value 1 instead of 0 in the weight matrix and the green line represents the opposite situation, when 30% of correlated pairs were believed to be uncorrelated.

W-SPSA was thus shown to work well without any knowledge about the relative correlation magnitudes, when at least the decision between 0 and 1 was correct. The algorithm became slightly unstable with 1 to 0 mistakes when significant correlations were neglected and relatively better with 0 to 1 mistakes. This means whenever a decision about correlation is hard to made, setting it as 1 will be a safer way. In conclusion, even without any magnitude information and with errors when deciding correlations, W-SPSA still achieves better goodness-of-fit than SPSA.

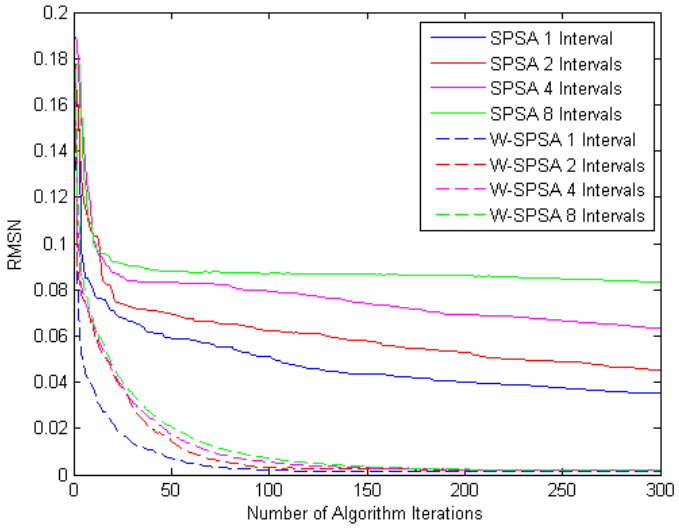


Figure 36: Performance comparison of SPSA and W-SPSA

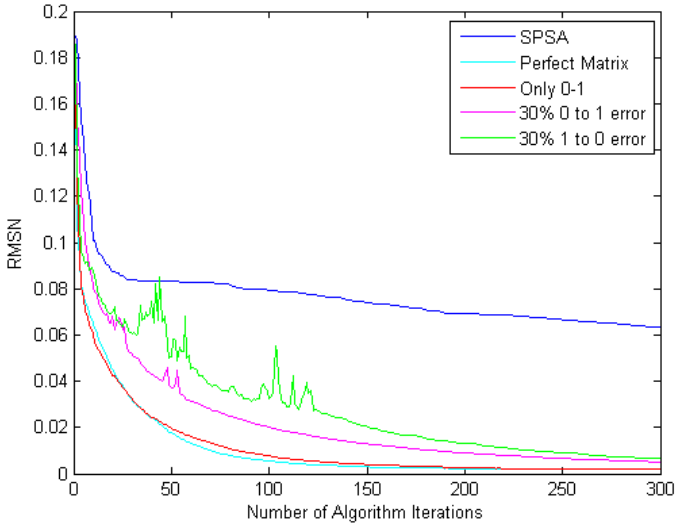
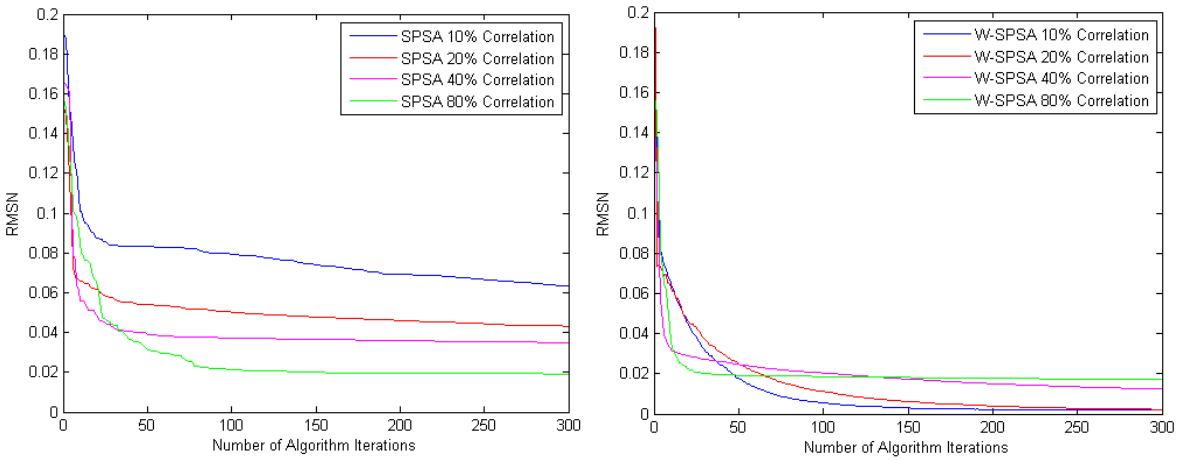


Figure 37: Performance of W-SPSA using inaccurate weight matrices

The degree of network correlation is also expected to play a significant role in the performance of W-SPSA. The third experiment is designed to investigate different network correlations. This test was done with 1000 OD pairs and 100 measurements per interval and 4 intervals in total. Figure 38 shows the relationship between network correlation and calibration performance. When the network correlation is increased, the performance of SPSA increases, while that of W-SPSA decreases. With an 80% network correlation, SPSA achieved long run goodness-of-fit similar to W-SPSA, although the convergence rate of W-SPSA was still much higher than SPSA, because W-SPSA still had an advantage in the temporal dimension. Theoretically, SPSA has the same performance as W-SPSA with a 100% network correlation and 1 simulation interval. This is easy to understand, because SPSA is essentially a limiting case of W-SPSA when all the weights in the weight matrix are 1, which means 100% correlation between all the variables and measurements. In other words, W-SPSA is a generalization of SPSA.



(A) Calibration performance of SPSA (B) Calibration performance of W-SPSA

Figure 38: The relationship between network correlation and calibration performance

6.5 Case Study

In this section a case study of the entire expressway network in Singapore is discussed to demonstrate the performance of W-SPSA in the calibration of a real-world city-scale traffic simulation.

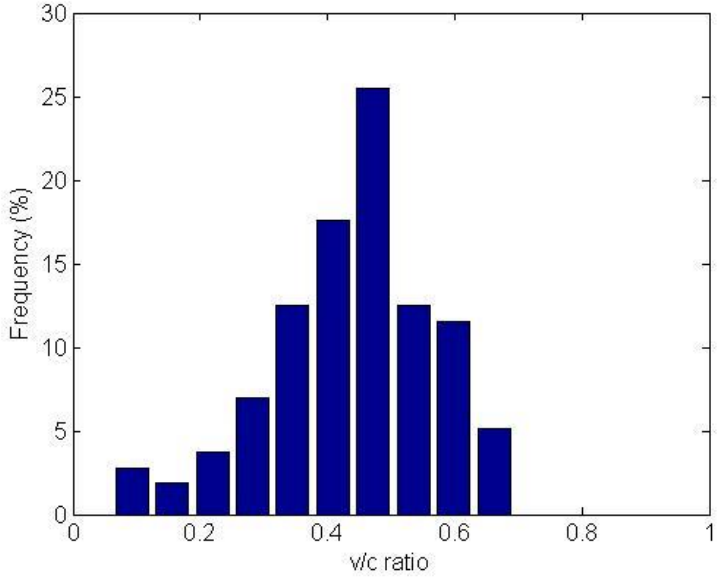
6.5.1 Experiment Setting

A state-of-the-art mesoscopic traffic simulator, DynaMIT, was used in the case study to model the expressway system in Singapore. DynaMIT (Ben-Akiva et al., 1997, 2001, 2010) is designed for traffic managers to evaluate surveillance and traffic management design, as well as plan future system and network modifications. Used in real-time, DynaMIT can provide consistent and unbiased estimation and prediction of traffic conditions to drivers and traffic control centers to assist them with route guidance and evaluations of different control strategies. DynaMIT has been successfully applied in a variety of cities, including Los Angeles, California (Wen et al., 2006), Lower Westchester County, New York (Rathi et al., 2008, Antoniou et al., 2011a), and Boston, Massachusetts (Balakrishna et al.; 2008). DynaMIT's ability to model highly congested urban networks was shown in a recent case study in the city of Beijing (Ben-Akiva et al., 2012).

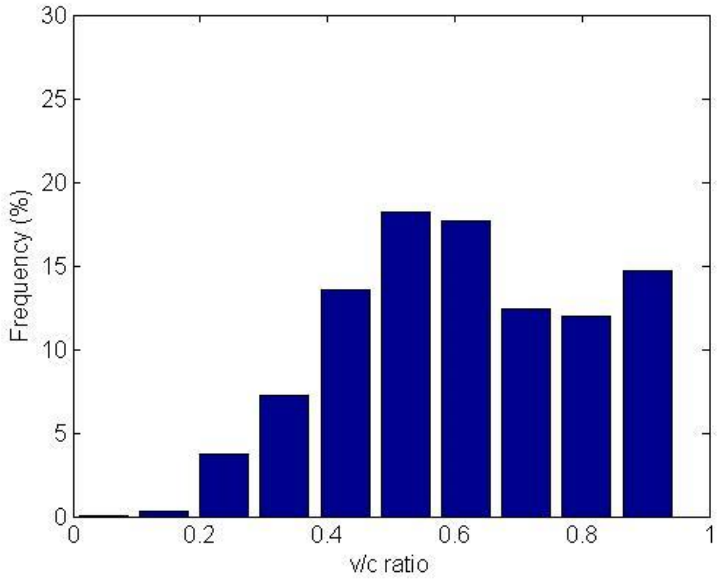
Figure 20 shows the Singapore expressway road network. This expressway system consists of expressway segments and ramps connecting local roads with the expressway. The network has been modeled as 831 nodes, 1040 links and 3388 segments. The isolated expressway system has clearly defined origins, which are the beginnings of on-ramps, and destinations, which are the ends of off-ramps. Thus, the classic "Zone - Centroid" method, which was explained in Section 2.1.2, is not used. In total, there are 831 on-ramps and 830 off-ramps. Heuristic rules are used to eliminate unreasonable short, unreasonable long and similar-located OD pairs. In the end, 4106 OD pairs are chosen among the 831 * 830 possible node combinations. The simulation period is set from 0:15 to 23:00, split into 91 intervals of 15 minutes each. Thus, there are in total 373,646 time-dependent OD flows.

Observed counts are available from 216 detection cameras located across the expressway system. The counts data are provided by the Land Transport Authority of Singapore for every 5-minute interval and are aggregated to 15-minute intervals to match the simulation interval length. In this case study, the data is averaged across the 31 days in August 2011 to reduce the influence of incidents and possible sensor failures. Figure 39 presents the distribution of the (estimated) congestion level (illustrated through the volume to capacity ratio, v/c) for the entire network: the top subfigure presents the congestion level of the segments with sensors, while the bottom subfigure presents the congestion level of all segments. It is noted that ramps (included in the

bottom subfigure of Figure 39) are more congested, but there are no sensor measurements on the ramps (as the data collection cameras are in the middle of Expressway segments).



(A) the congestion level of the segments with sensors



(B) the congestion level of all segments

Figure 39: The distribution of the road congestion level

While the presented approach and solution algorithm is aimed at the joint demand-supply calibration process, in this case study we have only used it to calibrate the OD flows using sensor counts. Balakrishna and Koutsopoulos (2008) show that a good seed OD can significantly improve the final goodness-of-fit in terms of fit to measurements. However, in this case study, no seed OD is available and the starting ODs are assigned with reasonable, but not necessarily highly reliable, values with the assistance of local traffic researchers from the Land Transport Authority of Singapore. Therefore, the distance between historical variable values and calibrated variable values is not included in the objective function and the calibration process.

6.5.2 Data Consistency Check

The traffic surveillance data is provided from the EMAS system (Expressway Monitoring and Advisory System). The flow data is obtained from detection cameras (see Figure 40), which are mounted on street lamps at distances of approximately 500 meters. The detection cameras are able to "count" vehicles passing on the road below. Every 5 minutes the cameras report the flow back to the central system. The central system then estimates the flow on the road segment using specific algorithms, even for those without sensors on them.

Figure 40: A detection camera that measures traffic flow on Singapore expressways

We investigated the flow data carefully due to the fact that they are fusion data generated by unknown algorithms and the exact location of some cameras are not available to us. The data consistency check is based on "daily accumulated sensor flows", which sums all flow values of each sensor during one day. In this case, each sensor (or camera) gets 288 values during a day. Three consistency tests were done to detect possible inconsistencies within the data. Figure 41 shows the three scenarios that were covered in our tests. The solid circles represent nodes. A line between two nodes is a link. Rectangular boxes represent sensors. The arrows show the direction of traffic flow.

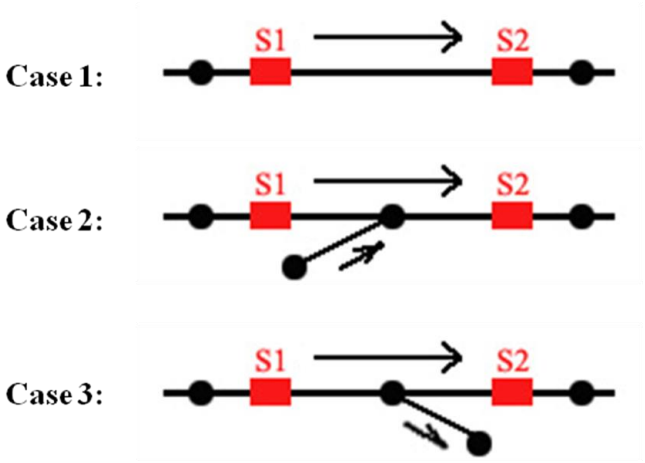


Figure 41: Three scenarios in the flow data consistency check

In case 1, there is no on-ramp or off-ramp between the two sensors, which means no vehicle is entering or exiting between these two sensors and these two sensors are supposed to give similar daily accumulated sensor flows. If there is a big discrepancy between the measured flow at sensor 1 (S1) and that at sensor 2 (S2), these two sensors are determined to be inconsistent. In case 2, there is an on-ramp between sensor 1 and sensor 2. It means the daily accumulated sensor flows at sensor 1 should be no bigger than sensor 2. Otherwise, these two sensors are determined to be inconsistent. In case 3, there is an off-ramp between sensor 1 and sensor 2. It means the daily accumulated sensor flows at sensor 1 should be no smaller than sensor 2. Otherwise, these two sensors are determined to be inconsistent. Once a pair of inconsistent sensors are detected, manual check is done to remove one of them.

6.5.3 Calculation of the Weight Matrix

In this case study, time-dependent traffic counts on 216 segments were used to calibrate time-depend OD flows for 4106 OD pairs in 91 intervals. The weight matrix had 4106 * 91 rows and 216 * 96 columns with each element reflecting the relative correlation between an OD flow and an observed count. These correlations were calculated using the network topology, historical travel time on each link and a route choice model. DynaMIT includes a path choice-set generation algorithm to generate a set of reasonable paths a driver could choose from for each OD pair. The probability to choose each path is decided by a route choice model (Frejinger, 2007, provides a detailed review for path choice set generation and route choice models). Reasonable experiential values of the parameters in the route choice model can be assigned before the calibration as the initial value and to generate weight matrix. Key inputs to the route choice model are historical travel times on each link. These historical travel times will also be used in a later step to calculate correlations in the weight matrix. If no such data is initially available, usually free-flow travel time on each link is used. It should be noted that the traffic conditions may vary significantly across the whole simulation period (peak vs. off-peak), so do the weights. Therefore, three different weight matrices are generated to be used in traffic conditions in AM-Peak, PM-Peak and Non-Peak.

This calculation is similar to the calculation of the assignment matrix in a network, potentially giving rise to the following question: why not directly use an assignment matrix to calculate OD flows with the traditional general least square (GLS) approach? Two reasons can be given for this. The first reason is that this approach has the potential to incorporate different types of data and calculate different types of variables simultaneously, while GLS estimation with assignment matrix can only use traffic flow data to estimation OD flows (incremental methodological extensions have been suggested, but they are limited and cumbersome; see e.g. Antoniou et al., 2006). The second reason is that GLS estimation is based on direction calculations using the assignment matrix. The accuracy of its results depends heavily on the quality of the assignment matrix. However, W-SPSA is a stochastic searching approach and is not as sensitive to the quality of the weight matrix (either 1 or 0).

6.5.4 Choice of Algorithmic Parameter Values

Part of the parameters in W-SPSA was determined based on Balakrishna (2006): b= 0.602, γ= 0.101, A = 50. These parameters control the rate of perturbation and advance step size reduction and can be determined based on standard guidelines and adjusted during the calibration practice. By outputting the input parameter values and RMSN after each run of the model (three times in an iteration: after the two perturbations and after obtaining the updated variable values), it can be decided if the step size reduction is too fast or too slow. In this case study, it was found that using the values from standard guidelines was good enough.

Compared with b, γ, and A, it was found in this case study that the algorithm was more sensitive to a and c, which are the initial advance and perturbation step size, respectively. For c, the first step is to determine an initial value based on the specific problem. In this case study, the OD flows are the numbers of cars that leave from an origin to a destination in a 15-minute interval. Changing this value by 1 or 2 will hardly make any meaningful change due to stochasticity from the simulator and the algorithm. However, changing it by 100 vehicles will have too much influence and will introduce serious congestion across the whole network. Therefore, the number of 10 was used as the starting value of c. The starting value of a was decided by running the calibration for a few iterations, inspecting the change of variable values after each iteration to make sure that it is in a reasonable range. For example, in an iteration, after perturbing the OD flow from 50 to 60 and to 40, if the updated OD flow is 100, it means that a is too big. On the contrary, if the updated OD flow is 50.5, it means that a is too small. The values of a and c were tested and further adjusted with preliminary runs by inspecting the performance of RMSN reduction. In these runs, only a few intervals were calibrated to save time, before running the calibration for the whole day.

6.5.5 Results

The performance of W-SPSA in a real world large network with a large number of simulation intervals was demonstrated by applying it to the simultaneous calibration of DynaMIT for the entire 91 intervals on the Singapore expressway network. As shown in Figure 42, W-SPSA has been shown to outperform SPSA in terms of convergence rate and long run achievable goodness-of-fit. The initial RMSN was 0.42 and after running W-SPSA for 30 iterations, it converged to a

RMSN of 0.16 on average. The RMSN is reduced to 0.17 after only 10 iterations in W-SPSA. It is noted that other measurements (e.g. reliable speed measurements) were not available for this road network, and therefore these measurements could not be used in the calibration process.

Figure 43 shows the fit to sensor counts in two different intervals: 7:45-8:00 (morning peak), 18:00-18:15 (evening peak). The x-axis corresponds to observed counts and the y-axis to the simulated counts after calibration. Blue points represent the simulated and observed flows at a specific sensor. The red 45-degree line indicates a perfect match between simulated counts and observed counts. Most points are very close to the 45-degree line, while the deviations are fairly regularly distributed around the line, suggesting no systematic over- or under-estimation of the counts.

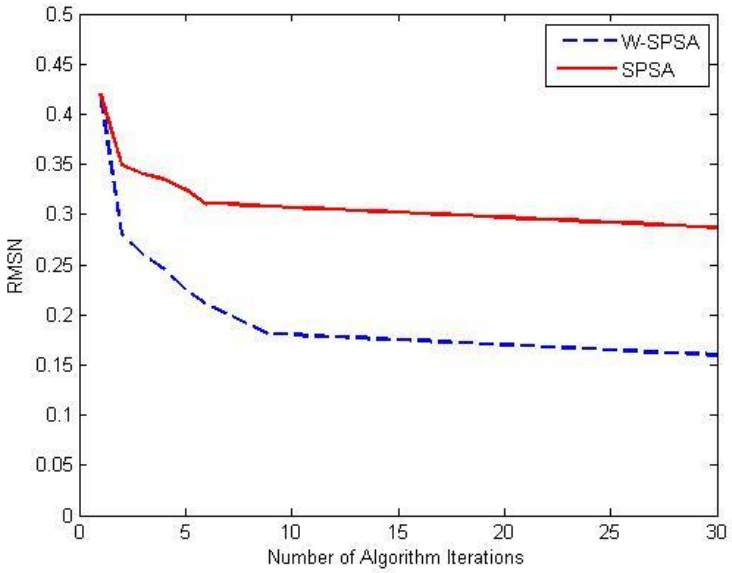
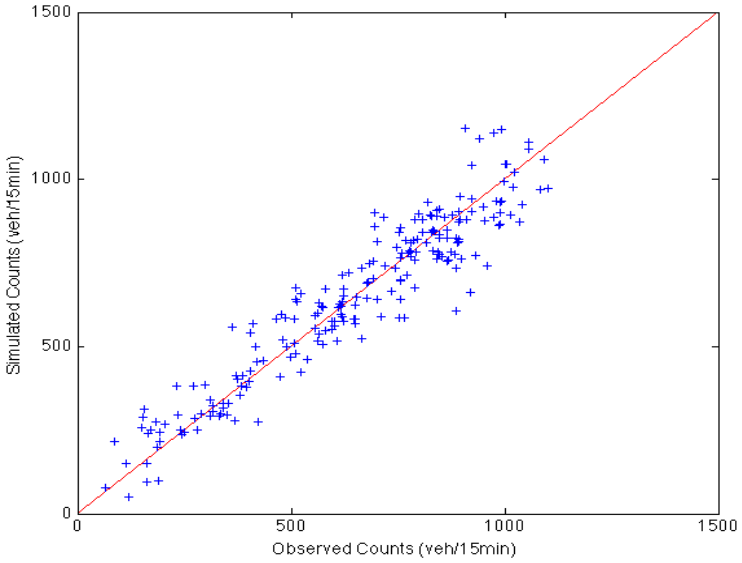
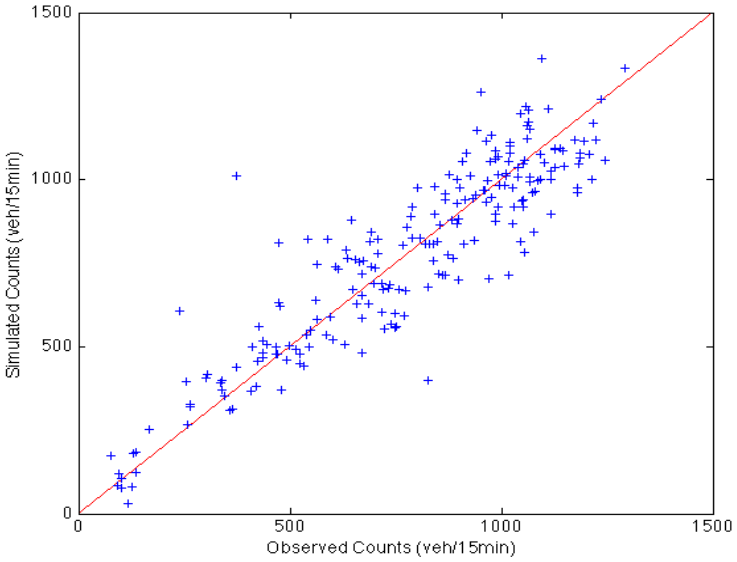


Figure 42: Comparison of W-SPSA and SPSA on a real-world traffic scenario



(A) AM Peak 7:45-8:00



(B) PM Peak 18:00-18:15

Figure 43: Fit to sensor counts in two different intervals

6.6 Summary

The topic of calibration of city-scale simulation models is an active research field. This chapter presented an enhancement to SPSA, a well-established solution algorithm of the state-of-the-art joint demand-supply calibration of traffic simulation models. The original SPSA algorithm was generalized by incorporating a 2D correlations weight matrix to reduce the gradient approximation error and make the algorithm work better. Synthetic tests showed the advantage of the enhanced SPSA, or W-SPSA, in terms of convergence rate, long run achievable goodness-of-fit and its robustness when accurate weight matrices are not accessible. Furthermore, a case study of the entire expressway network in Singapore showed the successful application of W-SPSA to the calibration of a large network with a large number of intervals.

Theoretically SPSA has a better best-case performance, because it considers all correlations, while W-SPSA ignores some correlations. However, the fact that SPSA considers all correlations implies that it is also susceptible to all the noise in the system. Increasing the number of non-zero weights considered by W-SPSA would theoretically further improve its performance, but it would result in a computational overhead (translated into a larger number of iterations to achieve convergence). Furthermore, W-SPSA still suffers from some of the limitations associated with SPSA, such as the fact that it can still get trapped in local optima rather than finding the global optimum. The importance of a good initial solution in reducing the likelihood of getting stuck in a local optimum, especially in highly congested networks, has been pointed out by Frederix et al. (2013). One way to overcome this limitation in practice is to re-initialize the algorithm with different initial solutions and observe whether the same solution is reached every time.

7. Conclusions and Future Research

7.1 Conclusions

Road congestions in a city-scale (or urban) traffic system are largely determined by the equilibrium between the demand (people's requirement for travel) and the supply (the capacity of the traffic system). In general, there are two types of solutions to manage road congestions in a city-scale traffic system: transport planning and traffic control schemes. Due to the high complexity in a city-scale traffic system and the limited modelling capability of mathematical models, city-scale traffic simulation becomes an appealing toolkit to evaluate the holistic impact of transport planning and traffic control schemes to the entire city-scale traffic system. However, a successful deployment and maintenance of a city-scale traffic simulation is not trivial, and this limits the usage of city-scale traffic simulations in the real-world traffic systems. The thesis investigated and then proposed solutions on two challenges: the performance of a city-scale traffic simulation and the calibration algorithm to estimate variables (e.g. model parameters and model inputs) in a city-scale traffic simulation. The thesis proposed frameworks, data structures and algorithms in computer science, to solve problems involved in performance optimization and calibration of a city-scale traffic simulator.

The research contributions of this thesis to the state-of-the-art performance optimization of city-scale traffic simulations are listed below:

- This thesis proposed a systematic three-step performance optimization methodology. These steps are: framework optimization, serial bottlenecks optimization and scalability optimization. The three-step performance optimization methodology was successfully demonstrated to reduce the execution time to simulate the Singapore expressway road network from 7:00AM to 8:00AM with in total 106,386 vehicles.

- This thesis proposed an Entry Time based Supply Framework (ETSF) to reduce the execution time to simulate congested traffic scenarios. The computational complexity of the proposed ETSF framework is less sensitive to the level of congestions. Experiment results showed that ETSF outperforms the current supply framework, by reducing the execution time by 50% - 95% in city-scale road networks and congested traffic scenarios.
- When the number of objects (e.g. vehicles) increases in a city-scale traffic scenario, the maintenance cost to rebalance a spatial index becomes a serial bottleneck. This thesis proposed Sim-Tree, which is a more stable spatial index, whose balance depends only on the average road density and thus is insensitive to individual vehicles' changing locations. The results of experiments simulating a city-scale traffic scenario on a 6-core machine showed that the Sim-Tree performs significantly better than the R*-tree family of spatial indexes.
- A new type of hardware, the GPU, which has hundreds of cores, is gaining popularity in high performance computing, because of its massive performance compared to the CPU. A research question is whether the GPU can be a potential high-performance platform for city-scale traffic simulations. This thesis proposed a comprehensive framework to run the proposed ETSF framework on the CPU/GPU platform. The proposed framework was demonstrated on a large-scale artificial grid road network and a real-world Singapore expressway road network.

The research contributions of this thesis to the state-of-the-art calibration of city-scale traffic simulations are listed below:

- As the traffic road network size grows, we found that the state-of-the-art calibration algorithm (SPSA) deteriorates. in the systematic error to incorporate uncorrelated measurements in the method of estimating gradients of calibration variables. Motivated by this finding, we propose W-SPSA ('W' means Weighted). The key idea of W-SPSA is to incorporate a 2-D weight matrix in the calibration algorithm, to assist the estimation of the gradient. The 2-D weight matrix represents correlations between variables and measurements. W-SPSA is successfully demonstrated to calibrate 373,646 time-dependent OD flows in one day in Project DynaMIT on Singapore Expressway Network.

However, there are several limitations in this research:

- Sim-Tree, as an efficient two-dimensional spatial index, cannot be directly used in traffic simulators that use one-dimensional coordinate systems, like DynaMIT.
- The evaluation of the proposed method to run traffic simulations on the CPU/GPU platform depends on the ETSF framework. Thus, the qualitative performance gain might be different when the proposed method is used in a different traffic framework.
- The W-SPSA algorithm requires a 2-D weight matrix, which represents the correlations between variables and measurements. However, the methodology to generate the 2-D weight matrix depends on the type of variables and the type of measurements. It requires users to determine how to generate the 2-D weight matrix in a different case study. Besides, the lack of high quality traffic speed data in the real-world case study reduces the practical significance of the case study.

7.2 Future Research

Future researches in performance optimization of city-scale traffic simulations are suggested:

- Applying the three-step performance optimization methodology

  The three-step performance optimization methodology was proposed based on the fundamentals in parallel computing technologies (e.g. Amdahl's law) and then was demonstrated using academic traffic simulators (e.g. DynaMIT and SimMobility) in the Singapore expressway network. It will be interesting to investigate the three-step performance optimization methodology to improve the performance of existing commercial parallel traffic simulators.

- Specialized traffic simulation frameworks

  The idea of the Entry Time based Supply Framework (ETSF) comes from identifying the most important components in city-scale traffic simulations, and then optimizing its performance by avoiding calculating individual vehicle's locations. For other problems, the idea of rebalancing functional capability and computational complexity can generate

more efficient frameworks, than directly reusing one of the traditional traffic simulation frameworks (microscopic, macroscopic, mesoscopic and nanoscopic).

- City-scale traffic simulations on the GPUs

  This thesis proposed a simulation framework to execute a city-scale traffic simulation on the GPUs. A speedup of 3.5-11.2 was observed when simulating the supply component on a GPU than on a CPU. However, the high global memory access delay on the GPU and the inefficient usage of the shared memory are bottlenecks to make full use of the massive computational power in the GPUs.

Future research directions in the calibration of city-scale traffic simulation are suggested:

- Applying W-SPSA in simultaneous demand-supply joint calibration

  The idea in weighted gradient approximation is applicable for general simultaneous demand-supply joint calibration. However, this thesis focused on OD calibration using aggregate flow counts on selected roads. Using W-SPSA in simultaneous demand-supply joint calibration involves the estimation of a generalized weight matrix, a joint usage of different types of observed data (e.g. flow and GPS data), and a higher computational cost.

- Incorporating novel data types into the calibration

  Nowadays more and more emerging types of data are available to transportation researchers and agencies, which provide an opportunity to understand and estimate the influence of factors that are rarely considered in previous studies. For example, with the abundance of information on the Internet and text mining techniques, the information for special events (e.g., concerts, sports games), including their location, time, and potential popularity can be obtained and archived automatically. Using these data, the influence of such events to demand and supply can be modeled and calibrated.

- Parallel calibration

  The proposed algorithm W-SPSA can be parallelized in many ways. The simplest way is to execute traffic simulations with the two different perturbed variables in parallel on two threads/processors. The second way is to allow more sets (>2) of perturbed variables

in parallel and generate better estimated gradients. It will also be interesting to investigate multiple executions of W-SPSA with different starting values, to avoid getting trapped by the local minimal. Finally, the calculation of weight matrices might be massively parallelized.

Bibliography

[1] Ahmane M., Abbas-Turki A., Perronnet F., Wu J., Moudni A. E., Buisson J. and Zeo R., (2013), "Modeling and controlling an isolated urban intersection based on cooperative vehicles", Transportation Research Part C: Emerging Technologies 28: 44-62.

[2] Ahmed K., Ben-Akiva M., Koutsopoulos H. and Mishalani R., (1996), "Models of freeway lane changing and gap acceptance behavior", Transportation and traffic theory (13): 501-515.

[3] Aimsun Online. (2014). http://www.aimsun.com/wp/?page_id=33.

[4] Akman V., Franklin W. R., Kankanhalli M. and Narayanaswmi C., (1989), "Geometric computing and the uniform grid data technique", Computer Aided Design 21, no. 7: 410–420.

[5] Antoniou C., (2004), "On-line calibration for dynamic traffic assignment", PhD thesis, Massachusetts Institute of Technology.

[6] Antoniou C., Balakrishna R., Koutsopoulos H. N. and Ben-Akiva M., (2011), "Calibration methods for simulation-based dynamic traffic assignment system", International Journal of Modelling and Simulation 31, no. 3.

[7] Appiah, J. and Rilett, L. R., (2010), "Joint estimation of dynamic origin-destination matrices and calibration of micro-simulation models using aggregate intersection turncount data", Transportation Research Board 89th Annual Meeting.

[8] Aydt H., Xu Y., Lees M., Knoll A., (2013), "A Multi-threaded Execution Model for the Agent-Based SEMSim Traffic Simulation", International Conference on Systems Simulation (AsiaSim). Singapore. 1-12.

[9] Azevedo J., (1993), "An algorithm for the ranking of shortest paths", European Journal of Operational Research 69: 97-106.

[10] Balakrishna R., (2006), "Off-line calibration of dynamic traffic assignment models", PhD thesis, Massachusetts Institute of Technology.

[11] Balakrishna R., Morgan D., Yang Q. and Slavin H., (2012), "Comparison of Simulation-Based Dynamic Traffic Assignment Approaches for Planning and Operations Management", 4th TRB Conference on Innovations in Travel Modeling. Florida.

[12] Balakrishna, R., Antoniou, C., Ben-Akiva, M., Koutsopoulos, H. N. and Wen, Y., (2007), "Calibration of microscopic traffic simulation models: methods and application", Transportation Research Record: Journal of the Transportation Research Board 1999, no. 1: 198-207.

[13] Barcelo J., (2002), "Dynamic network simulation with AIMSUN", Proceedings of the International Symposium on Transport Simulation.

[14] Barcelo J., (2010), "Fundamentals of Traffic Simulation", Springer, New York.

[15] Barcelo, J., Ferrer J. L., García D., Florian M. and Saux E., (1998), "Parallelization of Microscopic Traffic Simulation for ATT Systems Analysis", Massachusetts: Kluwer Academic Publishers.

[16] Beckmann, N., Kriegel, H., Schneider, R. and Seeger, B., (1990), "The R*-tree: an efficient and robust access method for points and rectangles", Proceedings of the 1990 ACM SIGMOD international conference on Management of data. 322-331.

[17] Ben-Elia E. and Shiftan Y., (2010), "Which road do I take? A learning-based model of route-choice behavior with real-time information", Transportation Research Part A: Policy and Practice, 44 (4), 249-264

[18] Ben-Akiva M. and Bierlaire M., (1999), "Discrete choice methods and their application to short-term travel decisions", In Handbook of Transportation Science, 5-34.

[19] Ben-Akiva M., Bierlaire M., Burton D., Koutsopoulos H. and Mishalani R., (2001), "Network State Estimation and Prediction for Real-Time Traffic Management", Networks and Spatial Economics 1, no. 3-4: 293-318.

[20] Ben-Akiva M., Bierlaire M., Koutsopoulos H. and Mishalani R., (1998), "DynaMIT: a simulation-based system for traffic prediction", Proceedings of the DACCORD Short-Term forecasting workshop.

[21] Ben-Akiva M., Gao S., Wei Z. and Wen Y., (2012), "A dynamic traffic assignment model for highly congested urban networks", Transportation Research Part C: Emerging Technologies 24: 62–82.

[22] Ben-Akiva M.,Bierlaire M.,Koutsopoulos H. and Mishalani B., (2002), "Real Time Simulation of Traffic Demand-Supply Interactions within DynaMIT", Transportation and network analysis: current trends.

[23] Ben-Akiva, M. and Lerman S. R., (1985), "Discrete Choice Analysis", MIT Press.

[24] Ben-Akiva, M., Bergman, M. J., Daly, A. J. and Ramaswamy, R., (1984), "Modeling intermodeling inter urban route choice behaviour", The Ninth International Symposium on Transportation and Traffic Theory.

[25] Bhat C. R. and Koppelman F. S., (1999), "Activity-Based Modeling of Travel Demand", in, Hall R. W. (Ed.), "Handbook of Transportation Science", Springer, US.

[26] Bierlaire M., (2004), "An efficient algorithm for real-time estimation and prediction of dynamic od tables", Operations Research 52: 116-127.

[27] Bovy P. H. L. and Stella F. C., (2006), "Stochastic route choice set generation: Stochastic route choice set generation: behavioral and probabilistic foundations", Proceedings of the 11th International Conference on Travel Behaviour Research.

[28] Box, M. J., (1965), "A new method of constrained optimization and a comparison with other methods", Computer Journal 8, no. 1: 42-52.

[29] Boyles, S. D., Tang S. and Unnikrishnan A., (2014), "Parking search equilibrium on a network", Transportation Research Part B: Methodological.

[30] Brockfeld E., (2005), "Calibration and validation of microscopic traffic flow models", The 84th Annual Meeting of the Transportation Research Board.

[31] Brodtkorb A. R., Hagen T. R. and Sætra M. L., (2013), "Graphics processing unit (GPU) programming strategies and trends in GPU computing", Journal of Parallel and Distributed Computing, 4-13.

[32] Burghout, W., Koutsopoulos, H. and Andreasson, I., (2006), "A Discrete-Event Mesoscopic Traffic Simulation Model for Hybrid Traffic simulation", Intelligent Transportation Systems Conference.

[33] Cameron G. D. B. and Duncan C. I. D., (1995), "PARAMICS wide area micro-simulation of ATT and traffic management", Proceedings of 28th International symposium on Automative Technology and automation (ISATA). 475-484.

[34] Cameron G. D. B. and Duncan G. I. D., (1996), "Paramics - Parallel Microscopic Simulation of Road Traffic", The Journal of Supercomputing, 25-53.

[35] Cantelmo, G., Cipriani, E., Gemma, A. and Nigro, M., (2014), "An adaptive bi-level gradient procedure for the estimation of dynamic traffic demand", IEEE Transactions in Intelligent Transportation Systems 15, no. 3: 1348-1361.

[36] Cascetta E., Inaudi D. and Marquis G., (1993), "Dynamic estimators of origin-destination matrices using traffic counts", Transportation Science 27, no. 4: 363-373.

[37] Cascetta, E., (1984), "Estimation of trip matrices from traffic counts and survey data: a generalized least squares estimator", Transportation Research Part B: Methodological 18: 289-299.

[38] Cascetta, E., Nuzzolo A., Russo F. and Vitetta A., (1996), "A modified logit route choice model overcoming path overlapping problems: Specification and some calibration results for interurban networks", Proceedings of the 13th International Symposium on Transportation and Traffic Theory. Lyon, France

[39] Cascetta E, Papola A., Marzano V., Simonelli F., Vitiello I., (2013), "Quasi-dynamic estimation of o-d flows from traffic counts: formulation, statistical validation and performance analysis on real data", Transportation research Part B, vol. 55: 171-187

[40] Cetin N., (2005), "Large-scale parallel graph-based simulations", PhD Thesis, Switzerland, ETH Zurich.

[41] Cheong C. C. and Toh R., (2008), "Household Interview Surveys from 1997 to 2008 – A Decade of Changing Travel Behaviours", Singapore.

[42] Cipriani E., Florian M., Mahut M. and Nigro M., (2011), "A gradient approximation approach for adjusting temporal origin-destination matrices", Transportation Research Part C: Emerging Technologies 19: 270–282.

[43] Commuter, (2014), http://project-commuter.info/gallery/

[44] Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C., (2009), "Introduction to Algorithms (3rd ed.)", MIT Press and McGraw-Hill.

[45] Daganzo C. F., (1994), "The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory", Transportation Research Part B: Methodological 28, no. 4: 269-287.

[46] Darda D., (2002), "Joint calibration of a microscopic traffic simulator and estimation of origin-destination flow", Master thesis, Massachusetts Institute of Technology.

[47] Dittrich J., Blunschi L. and Salles M. A. V., (2009), "Indexing Moving Objects Using Short-Lived Throwaway Indexes", In Proceedings of the 11th International Symposium on Advances in Spatial and Temporal Databases.

[48] Djukic T., Lint J. W. C. and Hoogendoorn S. P., (2012), "Application of principal component analysis to predict dynamic origin-destination matrices", Transportation Research Record 2283: 81-89.

[49] Duncan G., Matthew N. and Twomey S., (2014), "People Simulation for BIM and Infrastructure", White Paper, A2K Technologies

[50] Elmasri R. and Navathe S. B., (2010), "Fundamentals of database systems", Upper Saddle River, N.J.: Pearson Education.

[51] Fellendorf M., Vortisch P., (2014), "Microscopic Traffic Flow Simulator VISSIM", in, Barcelo J. (Ed.), "Fundamentals of Traffic Simulation". Springer, New York, 2010.

[52] Frederix R., Viti F. and Tampere C. M., (2013), "Dynamic origin–destination estimation in congested networks: theoretical findings and implications in practice", Transportmetrica A: Transport Science 9, no. 6.

[53] Frejinger E., (2007), "Route choice analysis: Data, models, algorithms and applications", PhD thesis, Ecole Polytechnique Federale de Lausanne.

[54] Gazis D. C., Herman R., and Potts R. B., (1961), "Car-Following Theory of Steady-State Traffic Flow", Operations Research 7(4) , 499-505

[55] Gladkov D., Tapia J. and Roshan M. D., (2012), "Graphics processing unit based direct simulation Monte Carlo", Simulation: Transactions of the Society for Modeling and Simulation International, 680-693.

[56] Gipps, P. G., (1981), "A behavioural car-following model for computer simulation", Transportation Research Board Part B, 15, 105-111

[57] Gipps, P.G., (1986), "A Model for the Structure of Lane-Changing Decisions", Transportation Research Part 20B, 403-414.

[58] Gurian, P. L., Villalobos, J., Chiu, Y. and Heyman, J., (2005), "Development of a multi-resolution large-scale vehicular traffic simulation and assignment model to assess impact of port-of-entry on regional infrastructure", Proceedings of the 2005 ASCE International Conference on Computing in Civil Engineering.

[59] Guttman, A., (1984), "R-Trees: A Dynamic Index Structure for Spatial Searching", Proceedings of the 1984 ACM SIGMOD international conference on Management of data, 47-57.

[60] Hazelton M., (2000), "Estimation of origin-destination matrices from link flows on uncongested networks", Transportation Research Part B: Methodological 34, no. 7: 549-566.

[61] Hobbs, F. D., (1974), "Traffic planning and engineering", Oxford, New York Pergamon Press.

[62] Hourdos, J. and Michalopoulos, P., (2008), "Access to Destinations: Twin Cities Metro-wide Traffic Micro-simulation Feasibility Investigation", Minnesota Department of Transportation.

[63] Huyer W. and Neumaier A., (2008), "Snobfit-stable noisy optimization by branch and fit", ACM Transactions on Mathematical Software (TOMS) 35, no. 2.

[64] Ilarri S., Mena E. and Illarramendi A., (2010), "Location-dependent query processing: Where we are and where we are heading", ACM Computing Surveys.

[65] Jayakrishnan R., Mahmassani H. and Hu T., (1994), "An evaluation tool for advanced traffic information and management systems in urban networks", Transportation Research Part C: Emerging Technologies 2, no. 3: 129-147.

[66] Jensen C. S., Lin D. and Ooi B. C., (2004), "Query and update efficient B+-tree based indexing of moving objects", Proceedings of the 30th international conference on very large data bases. 768–779.

[67] Jin H., (2001), "High Performance Mass Storage and Parallel I/O: Technologies and Applications (1st ed.)", Rajkumar Buyya and Toni Cortes (Eds.), John Wiley & Sons, Inc., New York, NY, USA.

[68] Jones M. H., (1999), "Stochastic optimization, stochastic approximation and simulated annealing", Wiley Encyclopedia of Electrical and Electronics Engineering.

[69] Kalidas A., (1996), "Estimation and prediction of time-dependent origin-destination flows", PhD thesis, Massachusetts Institute of Technology.

[70] Karypis G. and Kumar V., (1999), "A fast and highly quality multilevel scheme for partitioning irregular graphs", SIAM Journal on Scientific Computing, 359-392.

[71] Karypis G. and Kumar V., (1999), "Multilevel k-way Hypergraph Partitioning", 36th Design Automation Conference. 343-348.

[72] Kattan L. and Abdulhai B., (2006), "Non-iterative approach to dynamic traffic origin-destination estimation using parallel evolutionary algorithms", Journal of Transportation Research 1964: 201-210.

[73] Kim S. J., (2006), "Simultaneous Calibration of a Microscopic Traffic Simulation Model and OD Calibration", PhD thesis, Texas A and M University.

[74] Kim K. and Rilett L. R., (2004), "A genetic algorithm based approach to traffic micro-simulation calibration using its data", The 83rd Annual Meeting of the Transportation Research Board.

[75] Kindratenko V. V., Enos J. J., Guochun S., Showerman M. T., Arnold G. W., Stone J. E., Phillips J. C. and Wen-Mei H., (2009), "GPU clusters for high-performance computing", Cluster Computing and Workshops.

[76] Koh W. L. and Zhou S., (2011), "Modeling and simulation of pedestrian behaviors in crowded places", ACM Transactions on Modeling and Computer Simulation 21, no. 3.

[77] Kumaret V., Grama A., Gupta A. and Karpis G., (1994), "Introduction to parallel computing: design and analysis of algorithms", Redwood City, CA.

[78] Kwon D., Lee S. and Lee S., (2002), "Indexing the current positions of moving objects using the lazy update R-tree", Proceedings of the Third International Conference on Mobile Data Management.

[79] Lasalle D. and Karypis G., (2013), "Multi-Threaded Graph Partitioning", 27th IEEE International Parallel & Distributed Processing Symposium.

[80] Laval, J. A., (2004), "Hybrid models of traffic flow: impacts of bounded vehicle accelerations", PhD thesis, Department of Civil Engineering, University of California, Berkeley, USA.

[81] Lee J. B. and Ozbay K., (2008), "Calibration of a macroscopic traffic simulation model using enhanced simultaneous perturbation stochastic approximation methodology", Transportation Research Board 87th Annual Meeting.

[82] Lee M., Hsu W., Jensen C. S., Cui B., and Teo K. L., (2003), "Supporting frequent updates in R-trees: a bottom-up approach", In Proceedings of the 29th international conference on very large data bases.

[83] Lee D. H. and Chandrasekar P., (2002), "A framework for parallel traffic simulation using multiple instancing of a simulation program", Intelligent Transportation Systems Journal.

[84] Li B., (2005), "Bayesian inference for origin-destination matrices of transport networks using the em algorithm", Technometrics 47, no. 4: 399-408.

[85] Lin C. and Snyder L., (2008), "Principles of Parallel Programming", Addison-Wesley.

[86] Liu H., Ma W., Jayakrishnan R. and Recker W., (2004), "Large-Scale Traffic Simulation Through Distributed Computing of Paramics", California PATH Research Report.

[87] Lu L., Xu Y., Antoniou C. and Ben-Akiva M., (2014), "W-SPSA: An Enhanced SPSA Algorithm for the Calibration of Dynamic Traffic Assignment Models", Journal: Transportation Research Part C: Emerging Technologies.

[88] Ma J., Dong H. and Zhang H. M., (2007), "Calibration of microsimulation with heuristic optimization methods", Transportation Research Record: Journal of the Transportation Research Board 1999, no. 1: 208-217.

[89] Mahmassani H., Hu T. and Jayakrishnan R., (1992), "Dynamic traffic assignment and simulation for advanced network informatics (DYNASMART)", Proceedings of the 2nd International Capri Seminar on Urban Traffic Networks.

[90] May J. M., (2001), "Parallel I/O for high performance computing", Academic Press.

[91] May A. D. and Keller, H. E. M., (1967), "Non-integer Car Following Models", Transportation Research Board, 19-32.

[92] Messmer A. and Papageorgiou M., (1990), "METANET: a macroscopic simulation program for motorway networks", Traffic Engineering and Control 31: 466-470.

[93] Michalakes J. and Vachharajani M., (2008), "GPU acceleration of numerical weather prediction", IEEE International Symposium on Parallel and Distributed Processing.

[94] Munoz L., Sun X., Sun D., Gomes G. and Horowitz R., (2004), "Methodological methodological calibration of the cell transmission model", Inproceedings of Annual Control Conference. Boston, 798-803.

[95] Nagel K. and Rickert M., (2001), "Parallel implementation of the TRANSIMS micro-simulation", Parallel Computing, 1611-1639.

[96] Nie X. and Zhang H. M., (2005), "A comparative study of some macroscopic link models used in dynamic traffic assignment", Networks and Spatial Economics 5, no. 1: 89-115.

[97] Nie Y. and Zhang H. M., (2008), "A variational inequality approach for inferring dynamic origin-destination travel demands", Transportation Research Part B: Methodological 42, no. 7: 635-662.

[98] Nokel K., (2002), "Parallel DYNEMO: Meso-Scopic Traffic Flow Simulation on Large Networks", Networks and Spatial Economics, 387–403.

[99] Noronha V. and Church R., (2002), "Linear Referencing and Alternate Expressions of Location for Transportation", California Department of Transportation.

[100] Nvidia, (2013), "CUDA C Programming Guide".

[101] Ortuzar J. D. and Willumsen L. G., (2011), "Modelling Transport (4nd ed.)", Wiley.

[102] Othman N. B., Luo L., Cai W., and Lees M., (2013), "Spatial indexing in agent-based crowd simulation", Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques.

[103] Panwai S. and Dia H., (2005), "Comparative evaluation of microscopic car-following behavior ", IEEE Transactions on Intelligent Transportation Systems, 6(4), 314-325

[104] Park H. and Fishwick P. A., (2011), "An analysis of queuing network simulation using GPU-based hardware acceleration", ACM Transactions on Modeling and Computer Simulation.

[105] Passerat-Palmbach, J., Mazel, C. and Hill, D. R. C., (2011), "Pseudo-Random Number Generation on GP-GPU", International Workshop on Principles of Advanced and Distributed Simulation.

[106] Payne, H. J., (1979), "FREEFLO: A Macroscopic Simulation Model of Freeway Traffic", Transportation Research Board 772: 68-75.

[107] Paz, A., Molano, V. and Gaviria, C., (2012), "Calibration of corsim models considering all model parameters simultaneously", 15th International IEEE Conference on Intelligent Transportation Systems (ITSC). 1417-1422.

[108] Perumalla, K. S., Aaby B. G., Yoginath S. B. and Seal S. K., "GPU-based real-time execution of vehicular mobility models in large-scale road network scenarios", International Workshop on Principles of Advanced and Distributed Simulation. 2009.

[109] Pipes, L. A., (1953), "An Operational Analysis of Traffic Dynamics", Journal of Applied Physics 24: 274-281.

[110] Prato C. G., (2004), "Latent Factors and Route Choice Behavior", PhD thesis, Politecnico Politecnico di Torio

[111] Prato C. G., (2009), "Route choice modeling: past, present and future research directions", Journal of Choice Modelling, 2 (1), 65-100

[112] Yang Q., (1999), "A Simulation Laboratory for Evaluation of Dynamic Traffic Management Systems", Ph.D. Thesis, Massachusetts Institute of Technology.

[113] Yang Q., Koutsopoulos H. and Ben-Akiva M., (2000), "A simulation laboratory for evaluating dynamic traffic management systems", Transportation Research Record: Journal of the Transportation Research Board 1710: 122-130.

[114] Qian Z. and Michael H., (2011), "Computing individual path marginal cost in networks with queue spillbacks", Transportation Research Record.

[115] Ramakrishnan R. and Gehrke J., (2003), "Database Management Systems", McGraw-Hill.

[116] Rathi V., Antoniou C., Wen Y., Ben-Akiva M. E. and Cusack M. M., (2008), "Assessment of the impact of dynamic prediction-based route guidance using simulation-based, closed-loop framework", Transportation Research Board 87th Annual Meeting.

[117] Sidlauskas D., Ross K. A., Jensen C. S. and Saltenis S., (2011), "Thread-level parallel indexing of update intensive moving-object workloads", Proceedings of the 12th international conference on Advances in spatial and temporal databases. 186–204.

[118] Sidlauskas D., Saltenis S. and Jensen C. S., (2012), "Parallel main-memory indexing for moving-object query and update workloads", Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data.

[119] Sidlauskas D., Saltenis S., Christiansen C. W., Johansen J. M. and Saulys D., (2009), "Trees or grids? Indexing moving objects in main memory", Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. 236–245.

[120] SMART, (2014), http://smart.mit.edu/

[121] Smith M., Duncan G. and Druitt S., (1995), "PARAMICS: Microscopic traffic simulation for congestion management", Colloquium on Dynamic Control of Strategic Inter-Urban Road Networks.

[122] Spall J. C., (1992), "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation", IEEE Transactions on Automatic Control 37, no. 3: 332-341.

[123] Spall J. C., (1998), "An overview of the simultaneous perturbation method for efficient optimization", Johns Hopkins APL Technical Digest. 482-492.

[124] Spall J. C., (1994), "Developments in stochastic optimization algorithms with gradient approximations based on function measurements", Proceedings of the 26th conference on Winter simulation. 207-214.

[125] Stathopoulos A. and Tsekeris T., (2004), "Hybrid meta-heuristic algorithm for the simultaneous optimization of the o-d trip matrix estimation", Computer-Aided Civil and Infrastructure Engineering 19, no. 6: 421-435.

[126] Strippgen D. and Nagel K., (2009), "Multi-agent traffic simulation with CUDA", Proceedings of International Conference on High Performance Computing & Simulation.

[127] Taylor N. B., (2003), "The CONTRAM Dynamic Traffic Assignment Model", Networks and Spatial Economics 3: 297-322.

[128] Thomas H. C., Charles E. L., Ronald L. R. and Clifford S., (1990), "Introduction to Algorithms (3th ed.)", The MIT Press.

[129] Toledo T., Koutsopoulos H. N., Davol A., Ben-Akiva M., Burghout W., Andreasson I., Johansson T. and Lundin C., (2003), "Calibration and validation of microscopic traffic simulation tools: Stockholm case study", Transportation Research Record 1831: 65-75.

[130] Tomer T., Koutsopoulos H. and Ben-Akiva M., (2007), "Integrated driving behavior modeling", Transportation Research Part C: Emerging Technologies 15, no. 2: 96-112.

[131] TransModeler, (2014). http://www.caliper.com/transmodeler/.

[132] Van A. M. and Rakha, H., (1995), "TravTek evaluation modeling study", Technical report, Federal Highway Administration, US DOT.

[133] Wang Y., Messmer A. and Papageorgiou M., (2001), "Freeway network simulation and dynamic traffic assignment with METANET tools", Transportation Research Record: Journal of the Transportation Research Board 1776: 178-188.

[134] Wen Y., (2009), "Scalability of Dynamic Traffic Assignment", PhD Thesis, Massachusetts Institute of Technology.

[135] Wen Y., Balakrishna R., Ben-Akiva M. and Smith S., (2006), "Online deployment of dynamic traffic assignment: architecture and run-time management", IEEE Proceedings in Intelligent Transport Systems.

[136] Burghout W., Koutsopoulos H. N. and Andreasson I., (2006), "A Discrete-Event Mesoscopic Traffic Simulation Model for Hybrid Traffic simulation", Intelligent Transportation Systems Conference.

[137] Xu Y. and Tan G., (2012a), “An Offline Road Network Partitioning Solution in Distributed Transportation Simulation”, IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT).

[138] Xu Y. and Tan G., (2012b), “hMETIS-based Offline Road Network Partitioning", Asia Simulation Conference,.

[139] Xu Y., Song X., Weng Z. and Tan G., (2014), “An Entry Time based Supply Framework (ETSF) for Mesoscopic Traffic Simulations”, Journal: Simulation Modelling Practice and Theory.

[140] Xu Y. and Tan G., (2014), “Sim-Tree: Indexing Moving Objects in Large-Scale Parallel Microscopic Traffic Simulation”, ACM SIGSIM Conference on Principles of Advanced Discrete Simulation.

[141] Xu Y., Tan G., Li X. and Song X., (2014), “Mesoscopic Traffic Simulation on CPU/GPU”, ACM SIGSIM Conference on Principles of Advanced Discrete Simulation.

[142] Yeo H., Skabardonis A., Halkias J., Colyar J. and Alexiadis V., (2009), "Oversaturated Freeway Flow Algorithm for Use in Next Generation Simulation", Transportation Research Record: Journal of the Transportation Research Board, 68-79.

Appendix I: Terminology Definition

[1] **2-D**: Two-dimensional coordinate system. Similarly, 1-D means one-dimensional.

[2] **calibration**: the calibration of a traffic simulator means to estimate variables (e.g. model parameters and model inputs), in order to match model outputs with real-world traffic surveillance measurements..

[3] **city-scale:** In this thesis, the scale of a traffic scenario is measured by the number of roads (or segments) and the number of cars moving on roads. For example, a typical city-scale traffic scenario contains more than 3000 segments and up to 1,000,000 cars.

[4] **centroid node:** Zones in a road network are represented in the computer models as if all their attributes and properties were concentrated in a single point called centroid nodes. A typical centroid node is located as the geometry center of a zone.

[5] **congestion:** it is a condition on a road network, which is characterized by slower speeds, higher densities, and thus more vehicles on the road network.

[6] **demand:** Modeling from the travelers' point of view, the 'Demand' is to understand how travel decisions are made, such as origin and destination choice, mode choice, departure time choice, route choice and response to traffic information.

[7] **event-driven traffic simulation:** The status of the road network and individual vehicles in the simulation are updated only when some events (e.g. a vehicle enters a road) occur.

[8] **entry-time**: Each vehicle has an entry-time, which is the time when the vehicle enters the current lane. Note that an entry-time might have decimals.

[9] **flow:** The number of vehicles passing a point during a time interval, and then expressed as an equivalent hourly rate. The unit is vehicle/hour (v/h).

[10] **GPU (Graphics Processing Unit):** It is originally designed as a specialized hardware to accelerate the creation and the rendering of images for output to a display, however, modern GPUs have evolved into a highly parallel, general-purpose, multi-core processor with tremendous computational power and very high memory bandwidth.

[11] **jam density:** it refers to an extreme traffic density associated with a completely stopped traffic flow, usually in the range of 185–250 vehicles per mile per lane.

[12] **lane length:** if a 100-meter segment has 3 lanes, the segment's lane length is 300 meters.

[13] **performance**: Performance of a traffic simulator, in this thesis, means the execution time of the traffic simulator to simulation a traffic scenario. High performance means shorter execution time; performance optimization means reducing the execution time.

[14] **OD Matrix**: an OD matrix is a 2-D table, whose rows are origin zones and columns are destination zones. Each number in the matrix represents the number of trips going from an origin zone to a destination zone during a period.

[15] **scalability**: Scalability, in this thesis, means the efficiency of reducing the execution time to simulate a traffic scenario by using multiple processing units (cores and machines). It is the same as speed-up.

[16] **spatial index**: it is a data structure in traffic simulations that manages locations of objects (e.g. a vehicle, a pedestrian, etc.) in a road network (e.g. to find the left vehicle).

[17] **speed:** the moving distance per unit time. The default unit is meter/second (m/s).

[18] **supply:** modeling from the road network's point of view, the 'Supply' is to understand the capacity of a road network and also traffic control policies, incident response policies and event management methods.

[19] **time-stepped traffic simulation:** the status of the road network and individual vehicles in the simulation are updated at an appropriately chosen unit time. The chosen unit time (e.g. 0.1 seconds or 5 seconds) is named the time step of a traffic simulation.

[20] **time speedup:** time speedup is the executing time of a traffic simulation on 1 processing unit over the executing time of a traffic simulation on N processing units. In this thesis, speedup is the same as time speedup.

[21] **transport engineering:** an overall concept of engineering in any mode of transportation, including road traffic engineering, railway engineering, airport engineering, etc.

[22] **transport planning:** it refers to the 4-step transport planning program: trip generation, trip distribution, mode choice and trip/route assignment.

[23] **traffic control:** it refers to a variety of technologies and policies to control the demand and the supply, e.g. signal control, ramp metering, traffic information control, incident management, road pricing, car sharing, etc.

[24] **variable and parameter:** variables refer to a general concept whose value is unknown, consisting of model parameters (e.g. parameters in route choice models) and model inputs (e.g. the OD matrix as an input of route choice models).

Appendix II: Calculation of k in Sim-Tree

The average cost of a region query operation in Sim-Tree can be estimated using the formula:

$$Average\,Query\,Cost = k * log_k^N + w,\ (k \geq 2, w \geq 0)$$

where,

k is the number of children in a node in Sim-Tree,

w is the average number of objects in a leaf node,

N is the total number of leaf nodes,

log_k^N is the depth of the tree structure,

$k * log_k^N$ is the cost of finding the target leaf node

To find the turning point (which is also the optimal solution of k),

$$\frac{d(k * log_k^N + w)}{d(k)} = 0$$

$$-> \ln N * \frac{d(\frac{k}{\ln k})}{d(k)} = 0$$

$$-> \ln N * \frac{\ln k - 1}{(\ln k)^2} = 0$$

$$-> \ln k - 1 = 0$$

$$-> k = e$$