Load-balancing and synchronisation for parallel agent-based road traffic simulation

Xu, Yadong

2017

Xu, Y. (2017). Load-balancing and synchronisation for parallel agent-based road traffic simulation. Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/72141>

<https://doi.org/10.32657/10356/72141>

**LOAD-BALANCING AND SYNCHRONISATION FOR
PARALLEL AGENT-BASED ROAD TRAFFIC
SIMULATION**

XU YADONG

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the Degree of
Doctor of Philosophy

2017

Acknowledgements

I would like to express my sincere gratitude to my supervisor Professor Cai Wentong who has given me so many invaluable and critical advices on the research work. He has so much patience in guiding me to grow as a researcher, from developing research ideas to polishing up my writing skills. He has made my PhD study a far more rewarding experience than I could imagine.

I would like to give my thanks to my previous supervisor Dr. Michael Lees for his continuous guidance in the first two years of my PhD study. He has led me along the way to research and shown me a lot in the research field. In the two years, he has been always so helpful whenever guidance and advices were needed. He has been so encouraging, inspiring, and considerate when I encountered difficulties in research.

I would also like to give a lot of my thanks to Dr. Heiko Aydtt who has given me a lot of valuable advices and comments along my PhD study. He has also given me a lot of day-to-day help in many aspects. I could always benefit a lot from discussions with him.

I would like to thank Nanyang Technological University for its financial support in the NTU Research Scholarship. At the same time, I would like to thank Singapore National Research Foundation for its financial support for this research work under its Campus for Research Excellence And Technological Enterprise (CREATE) programme.

Contents

Acknowledgements	i
Abstract	vii
List of Tables	ix
List of Figures	xi
Symbols	xv
Acronyms	xviii
1 Introduction	1
1.1 Background and Motivation of the Research	1
1.2 Problem Statements	5
1.3 Contributions	7
1.4 Thesis Outline	8
2 Literature Review	10
2.1 Traffic Simulation and Models	10
2.1.1 Macroscopic and mesoscopic models	10
2.1.2 Microscopic models	12
2.1.3 Nanoscopic models	13
2.2 Partitioning and Dynamic Load-balancing of Parallel Agent-based Traffic Simulation	14
2.2.1 Introduction to partitioning and dynamic load-balancing	14
2.2.2 Partitioning of agent-based traffic simulation	16
2.2.3 Dynamic load-balancing in parallel agent-based traffic simulation .	19

2.3	Time Synchronisation in Parallel Agent-based Traffic Simulation	20
2.3.1	General synchronisation protocols	20
2.3.2	Time synchronisation of parallel agent-based traffic simulations . .	22
2.4	Relaxation of Synchronisation	23
2.4.1	Uncertainties in simulation	24
2.4.2	Relaxation of synchronisation utilising uncertainty	25
2.5	Reducing Inter-process Communication with Redundant Computation . .	26
2.6	Summary	28
3	Parallel Agent-based Traffic Simulation: SEMSim Traffic	30
3.1	Overview	30
3.2	Simulation Models and Time Advancement	31
3.2.1	Simulation space	31
3.2.2	Agents	31
3.3	Parallelisation in a Distributed-memory Environment	33
3.3.1	Decomposition of the simulation	33
3.3.2	Synchronisation and load imbalance issues	36
3.4	Partitioning and Dynamic Load-balancing	37
3.4.1	Partitioning	38
3.4.2	Dynamic load-balancing	41
3.5	Synchronisation of LPs	44
3.6	Experiments and Results	46
3.6.1	Implementation and hardware	46
3.6.2	Scenario set-up	47
3.6.3	Comparison of static partitioning and dynamic load-balancing using global barrier synchronisation	49
3.6.4	Comparison of global barrier synchronisation and MA protocol . .	54
3.7	Summary	57
4	A Graph-based Partitioning Heuristic	59
4.1	Overview	59

4.2	Neighbour-restricting Partitioning Heuristic	60
4.2.1	Problem definition	60
4.2.2	Graph-growing partitioning	60
4.2.3	Refinement with local search	65
4.3	Experiments and Results	68
4.3.1	Set-up	68
4.3.2	Results	68
4.4	Summary	73
5	Heuristics of Relaxing Synchronisation	74
5.1	Overview	74
5.2	Synchronisation-skipping and Dead-reckoning	75
5.2.1	Relaxation Heuristic	75
5.2.2	Lookahead Determination	78
5.3	Reducing the Temporal Resolution of Agent-based Models	80
5.3.1	Relaxation heuristic	80
5.3.2	Lookahead determination	82
5.4	Experiments and Results	85
5.4.1	Set-up	85
5.4.2	Results	85
5.5	Summary	93
6	Reducing Synchronisation Messages with Computation Replication	94
6.1	Overview	94
6.2	Adaptive Computation Replication	95
6.2.1	Extended layers	95
6.2.2	Sizes of extended layers	98
6.2.3	Adaptive extended layers	104
6.3	Experiments and Results	110
6.3.1	Set-up	110
6.3.2	Results	110

6.4 Summary	116
7 Conclusions and Future Work	118
7.1 Conclusions	118
7.2 Future Work	121
Bibliography	123
A Threshold Estimation for Initiating Load-balancing Operations	138
B Lookahead Determination of MA synchronisation Protocol	141
B.1 Lookahead based on Traffic Conditions	141
B.2 Lookahead of Relaxed Synchronisation By Increasing Update Intervals .	143
C Driver Behaviour Models	144
C.1 Acceleration Model	144
C.2 Lane-changing Model	146
C.3 Gap-acceptance Model	148
D Searching Algorithms for Extended Layers	149
D.1 Function <i>searchDownstream</i>	149
D.2 Function <i>searchUpstream</i>	152
E Publications	155

Abstract

Workload balance and synchronisation of logical processes (LPs) are two critical factors that influence the execution time and the maximum speed-up of a parallel agent-based road traffic simulation. The original contributions of this thesis are the new methods that make use of the characteristics of agent models to efficiently conduct load-balancing and synchronisation in parallel agent-based road traffic simulations.

We firstly evaluated the existing works of partitioning and load-balancing, as well as synchronisation protocols for parallel agent-based traffic simulations. We identified that the existing partitioning algorithms do not fully minimise the communication overhead between LPs. As for simulation protocols, most of the existing traffic simulators apply global barriers to synchronise LPs. Some existing works use conservative protocols to allow LPs to progress asynchronously. However, the performance may not be better than that of using global barrier, since in general agent models in traffic simulation only allow small lookahead.

We then developed a graph partitioning algorithm, aiming to reduce the synchronisation overhead of the simulation while maintaining workload balance. In addition to balancing workload and minimizing communication volume, it also aims to minimise the number of neighbouring partitions of each partition. Minimising the number of neighbouring partitions can reduce synchronisation messages among LPs during the simulation. Less synchronisation messages means less communication overhead to the simulation.

To further reduce the synchronisation overhead, two methods of improving the lookahead were developed. The first method takes advantage of the intrinsic uncertainties

in traffic simulation to relax synchronisation. Relaxation of synchronisation is accomplished either by simply skipping synchronisation operations, or by reducing the resolution of agent models at the boundaries of partitions. However, it is important to ensure that simulation results are statistically unaltered by the relaxation. To maximise relaxation, the behaviour of agents needs to be analysed. The second method reduces synchronisation operations by conducting computation replication to reduce data dependencies between LPs. Some data previously sent via synchronisation messages are replaced by redundant computation. Extended layers of partitions are defined according to the behaviour of agents in order to determine the amount of redundant computation required for a certain lookahead. However, there is a trade-off between the benefit of reduced synchronisation operations and the overhead of computation replication. Depending on the distribution of agents on the road network, the optimal lookahead for achieving the best speed-up of the simulation may change during the simulation. Hence, the suitable lookahead is dynamically adjusted during the simulation to ensure that the benefit of reduced synchronisation operations is greater than the overhead of computation replication.

All the proposed methods were evaluated in an agent-based discrete-event traffic simulator in a distributed-memory environment using real-world road network and trip data. Results have shown that firstly the proposed partitioning algorithm produces less synchronisation overhead than stripe spatial partitioning and the popular METIS partitioning algorithm. Secondly, the two methods of improving lookahead of LPs can effectively reduce synchronisation overhead and improve the overall speed-up of the parallel simulation. For relaxing synchronisation, applying a suitable amount of relaxation does not alter simulation results statistically.

The proposed methods can help to conduct large-scale microscopic or nanoscopic road traffic simulations faster. This is particularly useful in the studies where a large number of simulation runs are required.

List of Tables

2.1	Four levels of traffic models and examples of simulation tools	11
3.1	Hardware configuration of each compute node of the cluster	47
3.2	Average number of neighbouring LPs of each LP with dynamic load-balancing using stripe partitioning and METIS partitioning	52
3.3	Dynamic load-balancing count and total overhead of the partitioning algorithms and re-distributing agents	54
4.1	Total overhead of dynamic load-balancing using the proposed partitioning algorithm	72
5.1	Statistical test of equivalence using different sensitivity factor values in SE, using METIS partitioning	87
5.2	Statistical test of equivalence using different traffic density threshold values in LR, using METIS partitioning	88
5.3	Statistical test of equivalence using different sensitivity factor values in SE, using GG partitioning	88
5.4	Statistical test of equivalence using different traffic density threshold values in LR, using GG partitioning	89
5.5	Time spent on determining relaxed update intervals	92
6.1	Symbols used in the analysis of overhead of computation replication	105
6.2	Average redundant computation per LP	113
6.3	Total overhead of calculating extended layers	114

6.4	Total overhead of evaluating the optimum lookahead	114
6.5	Comparison of average lookahead, simulation execution time, and over- head of evaluating optimum lookahead using different evaluation periods with 32 LPs, using GG partitioning	114
6.6	Comparison of average lookahead, simulation execution time, and over- head of evaluating optimum lookahead using different evaluation periods with 32 LPs, using METIS partitioning	115
D.1	Symbols used in searching algorithms for extended layers	149

List of Figures

2.1	Illustration of driver behaviour models and vehicle component models .	13
2.2	Examples of RCB and graph partitioning	18
3.1	Illustration of agents and their sensing ranges	33
3.2	LP Architecture of SEMSim Traffic parallelised in a distributed-memory environment	34
3.3	Partitioning of a road network, buffer regions, and proxy agents	35
3.4	Dynamic workload of LPs in parallel traffic simulation using static partitioning.	38
3.5	Using traffic flow and density information of the road network to generate a weighted graph	40
3.6	Illustration of MA synchronisation protocol	45
3.7	The number of agents on the road network throughout the simulation. .	48
3.8	Singapore road network partitioned with stripe partitioning and METIS partitioning	48
3.9	Workload imbalance of 16 LPs throughout the simulation using static partitioning.	49
3.10	Comparison of workload imbalance throughout the simulation between static partitioning and dynamic load-balancing, using 16 LPs with METIS partitioning.	50
3.11	Average load-imbalance using static partitioning and dynamic load-balancing with stripe partitioning and METIS partitioning.	50

3.12	Indicators of synchronisation overhead using dynamic load-balancing with stripe partitioning and METIS partitioning	51
3.13	Performance of the parallel simulation using dynamic load-balancing with stripe partitioning and METIS partitioning	53
3.14	Average lookahead in terms of update intervals using MA synchronisa- tion protocol	55
3.15	Total synchronisation messages using MA and barrier synchronisation .	56
3.16	Execution time using MA synchronisation and global barrier synchroni- sation	56
3.17	Speed-up using MA synchronisation and global barrier synchronisation	57
4.1	Comparison of the proposed partitioning algorithm with stripe partitioning	64
4.2	Illustration of the neighbouring-restricting rule of local-research refinement	67
4.3	Singapore road network in 16 partitions generated by the proposed par- titioning heuristic and stripe partitioning.	68
4.4	Average load imbalance of the simulation in terms of number of agents .	69
4.5	The average number of neighbouring partitions per partition using vari- ous partitioning methods	70
4.6	Indicators of synchronisation overhead with various partitioning methods	71
4.7	Execution times and speed-ups of the simulation using various partition- ing methods	72
5.1	Illustration of the effect of skipping a synchronisation operation	76
5.2	Illustration of stable, linearly unstable, and metastable density regimes within flow-density diagrams	78
5.3	Illustration of the effect of increasing the update interval of boundary agents	81
5.4	Execution time of the parallel simulation using different synchronisation methods	90

5.5	Total number of synchronisation messages sent using different synchronisation methods	91
5.6	Average lookahead of relaxation methods using different synchronisation methods	91
6.1	Extended layers and new buffer regions	96
6.2	Different sizes of link segments in extended layers	99
6.3	An example of LP_i progressing through time using two extended layers .	100
6.4	Four basic cases of extended layer segments on connecting links.	101
6.5	Average lookahead and total number of synchronisation messages in MA and MA-CR synchronisation	111
6.6	Execution time and speed-up of the parallel simulation using barrier, MA, and MA-CR synchronisation.	112
6.7	Average lookahead of all LPs throughout the simulation with and without adaptive range.	115
B.1	Scenarios of lookahead calculation using MA synchronisation protocol .	142
C.1	Flow diagram of the lane-changing decision of agents.	147
C.2	Lead gap and lag gap in the target lane	148
D.1	Cases in downstream search on link l in determining extended layers . .	152
D.2	Case 3 in the upstream search on link l in determining extended layers .	153

Symbols

A^t	set of agents in the whole simulation at simulation time t	42
A_i^t	set of agents in the i th partition at simulation time t	42
$A_{i,j}^k$	set of agents in the k th extended layer of partition G_i inside G_j	104
$B_{i,j}$	buffer region of G_i inside G_j	35
CL_v	set of links connecting to node v in a road network	38
G	road network, represented as a graph of links and nodes	34
G_i	i th partition of the road network	34
$\hat{G}$	weighted graph for partitioning, generated from the road network G	38
$\hat{G}_i$	i th partition of weighted graph $\hat{G}$	40
I	total number of partitions	34
Imb^t	load imbalance at time t before the load-balancing operation	41
Imb_{bal}^t	load imbalance at time t after a load-balancing operation	138
$K_{i,j}$	number of available extended layers between LP_i and LP_j	104
$K_{i,j}^m$	adaptive range for evaluating extended layers between LP_i and LP_j in evaluation period τ_m	104
$M_{i,j}^t$	set of agents migrating from LP_i to LP_j at time t	44
N_{dlb}	period of making a load-balancing decision in terms of update intervals	138
$O_{i,j}^t$	set of outgoing boundary links from G_i to G_j at time t	79
Q_l^t	traffic flow of link l at time t	78
$Q_{v,u}$	traffic flow of the link from node v to node u	39
$\mathcal{S}$	similarity matrix of the partitions before and after a load-balancing operation	42
$S_{i,j}^t$	set of shared states sent by LP_i to LP_j at time t	44
$SizeA$	data size of a complete agent	107
$SizeS$	data size of the share state of an agent	107
$SynLP_i^t$	set of LPs having synchronisation appointments with LP_i at time t	44

T_a	computation time of an agent in one update interval	104
V_i	set of vertices in the i th partition of weighted graph $\hat{G}$	40
$X_{i,j}^k$	k th extended layer of partition G_i inside G_j	95
$\Delta t_{i,j}^t$	mutually agreed lookahead between LP_i and LP_j at time t	44
$\Delta t_{exist_{i,j}}$	minimum time for any agent in G_i to travel to buffer region $B_{j,i}$	142
$\Delta t_{new_{i,j}}$	minimum time for any new agent in G_i to travel to buffer region $B_{j,i}$	141
$\Gamma_{i,j}(k)$	redundant computation in LP_i between two consecutive synchronisations between LP_i and LP_j , incurred by replicating k extended layers from LP_j	104
α	sensitivity factor in the relaxed mutual appointment synchronisation	78
α_d	weight of traffic density in the communication index	39
α_f	weight of traffic flow in the communication index	39
δ	time-step, a.k.a, update interval	32
$\delta_{i,j}^t$	relaxed update interval of agents at the boundary of G_i and G_j at time t	82
γ_b	back sensing range of agents	32
γ_f	front sensing range of agents	32
ρ_l	traffic density of link l	38
ρ_l^t	traffic density of link l at time t	82
$\rho_{v,u}$	traffic density of the link from node v to node u	39
b	available bandwidth of message passing	104
c	latency at the sending and receiving ends during a message-passing	104
$\hat{k}_{i,j}$	obtainable optimum number of extended layers between LP_i and LP_j	104
$\hat{k}_{i,j}^m$	$\hat{k}_{i,j}$ in evaluation period τ_m	104
$l_{i,j}^t$	lookahead of LP_i towards LP_j at time t	44
$length_l$	length of link l	38
$o_{i,j}(\tau, k)$	total overhead for LP_i , relative to LP_j , during time period τ when k extended layers are replicated	104
$o_{i,j}(k)$	total overhead for LP_i , relative to LP_j , in one synchronisation cycle, when k extended layers are replicated	104
$s_{i,j}(k)$	time required by LP_i to receive a synchronisation message from LP_j in a synchronisation operation when k extended layers are replicated	104
$s_{m,n}$	similarity between the existing partition G_m and new partition G_n after a load-balancing operation	42
w_v	weight of vertex v in a partitioning graph	38

$w_{e(v,u)}$ weight of the edge connecting vertex v and vertex u in a partitioning graph 39

Acronyms

CR	MA protocol with computation replication	110
DLB	dynamic load-balancing	49
DVU	driver-vehicle-unit	1
EV	electric vehicle	13
GG	graph-growing	69
IDM	intelligent driver model	11
ITS	intelligent transportation systems	1
LP	logical process	vii, 2
LR	MA protocol relaxed by lowering the resolution of agent models	86
MA	mutual appointment	7
RCB	recursive coordinate bisection	17
SE	MA protocol relaxed by skipping and estimation	86

Chapter 1

Introduction

1.1 Background and Motivation of the Research

The world is projected to have 41 mega-cities, each with more than 10 million inhabitants by 2030; and 66 percent of the world's population is projected to be urban by 2050 [1]. Growing population in urban areas increases the load of transportation systems, which creates various problems such as long travelling time, downgraded road safety, and health problems due to polluted air [2]. Modelling and simulation of road traffic is a useful approach to study the problems of road traffic and experiment on potential solutions, such as deploying intelligent transportation systems (ITS) [3]. A traffic simulation is especially useful when the solutions are impractical or too expensive to test on a real system and too complicated to evaluate analytically.

The modelling and simulation of road traffic can be conducted with various levels of abstraction, where the spatio-temporal propagation of traffic flow is modelled differently. They are commonly known as macroscopic, mesoscopic, microscopic, and nanoscopic (a.k.a., sub-microscopic) models. Road traffic is a complex adaptive system, whose behaviour is difficult to predict. Traffic can be viewed as an emergent phenomenon resulting from the interaction of driver-vehicle-units (DVUs) on the road. The influence of the action from a DVU may propagate and have an impact on the whole system. To fully study this impact, the behaviour of DVUs needs to be modelled. This falls within

the scope of microscopic [4] and nanoscopic models [5]. Microscopic models describe a DVU as a single entity with behaviours. Nanoscopic models represent a DVU as a combination of a driver and a vehicle. The driver and the vehicle have individual behaviour models and component models. Microscopic and nanoscopic traffic simulations can both be conducted in an *agent-based* approach. In agent-based modelling and simulation, a system is modelled as a collection of autonomous decision-making entities called *agents*. They assess the environment individually and make decisions on the basis of a set of rules [6]. For microscopic and nanoscopic traffic simulations, agents are DVUs. In this research, we deal with agent-based microscopic and nanoscopic simulations. They belong to the discrete-event simulation family, where the *state* of the simulation is updated by *events* at discrete simulation times.

To study the behaviour of the traffic of a city or any other large areas, a large-scale simulation is usually required, which involves a large number of DVUs. Currently, one of the challenges in building a large-scale agent-based traffic simulation is to solve the issue of long simulation execution time. The issue arises due to a large amount of DVUs having models with high levels of detail. For example, a simulation of the traffic of modern big cities can contain millions of DVUs. This results in a slow execution of a traffic simulation, which makes the simulation unsuitable for real-time and faster-than-real-time applications. For instance, to make short-term traffic predictions requires a simulation to run much faster than real-time. Moreover, there are studies that require many simulation runs to obtain statistical results, and thus a slow execution is not acceptable. To alleviate the execution time issue of large-scale agent-based traffic simulations, parallel computing techniques have been used in some existing works to utilise more computational resources [7, 8, 9], such as high performance computer clusters and cloud computing platforms. In this thesis, we will focus on the algorithms for speeding up the execution of parallel agent-based road traffic simulations.

When a simulation is parallelised, it is decomposed into multiple partitions. Each partition contains a subset of the state of the simulation and a subset of the computational workload. A partition is executed by a logical process (LP), which is usually mapped to

a physical processor. There are two prominent overheads associated with parallelisation: load imbalance and inter-process communication. In our research, LPs communicate with each other by sending messages. The process of decomposing the workload and assigning it to LPs is called *partitioning* [10]. The optimisation objective of partitioning is to balance workload and minimise inter-process communication.

Load imbalance, as the name indicates, refers to the uneven distribution of computational workload among LPs. It directly affects the simulation speed, so the workload should be as balanced as possible. Moreover, workload in traffic simulation usually changes during simulation run-time. Therefore, *dynamic load-balancing* may be required during the simulation to re-distribute the workload. For inter-process communication, one of its purposes is to perform time synchronisation [11, 12]. Time synchronisation is required to ensure the correct execution order of events so as to avoid *causality errors*¹. Another purpose of inter-process communication is to fulfil data dependencies. In a parallel agent-based road traffic simulation, agents in one partition may subscribe to the states of agents in other partitions, and agents may move from one partition to another as well. State subscription and movement of agents lead to data read and write dependencies among LPs. Hereafter, we refer to the inter-process communication that manages both the simulation time and data dependencies of LPs to as *synchronisation*. Due to workload imbalance and synchronisation overhead, adding more computing resources does not always reduce the simulation execution time and sometimes even increases it. Reducing the overheads of workload imbalance and synchronisation therefore is critical for the performance of parallel agent-based traffic simulations.

In the existing works, partitioning of agent-based traffic simulations is commonly achieved by cutting the road network. Agents are assigned to LPs based on which road network partitions they are located in. To partition a road network, the workload information and possible data dependencies on links should be calculated. So the road network is first converted to a weighted graph, where vertices encode workload and edges encode data dependencies. There are generally two types of algorithms that partition

¹Causality errors are deviations of the simulation results obtained by parallel execution from those obtained by sequential execution due to out-of-order event processing.

the weighted graph. One type is spatial partitioning which uses the spatial information, i.e., geographical coordinates of vertices. The network is cut into stripes or grids in [13] and [14]. In [15], the network is cut using recursive bisection along the longitude and latitude directions. The drawback of considering only the geographical information is that it does not consider the minimisation of synchronisation overhead. Another type of partitioning algorithms is graph partitioning algorithms. Studies such as [8] and [9] have used METIS graph partitioning algorithm [16] in their agent-based traffic simulations. The objectives of METIS are to distribute the weights of vertices evenly among partitions as well as to minimise edge cut between partitions. Reducing edge cut reduces the data volume of inter-process communication. However, another crucial factor of synchronisation overhead that has not yet been considered in the works above is the number of synchronisation messages. Dynamic load-balancing of the simulation can be achieved by either incrementally moving the boundaries of partitions or repartitioning the whole road network from scratch. For spatial partitioning, re-partitioning is done in general by moving the boundaries of partitions according to the coordinates of vertices. For graph partitioning, diffusion-based adjustment or repartitioning from scratch can be applied [17].

Synchronisation in agent-based traffic simulations is commonly achieved by periodically applying global barriers [8, 18]. At a global barrier, all LPs are at the same virtual simulation time. They temporarily pause the execution of agent models and exchange messages. After exchanging messages, all LPs resume execution at the same simulation time. Microscopic and nanoscopic traffic simulations are conventionally executed in a time-stepped fashion. Global barriers are deployed at the end of time-steps (a.k.a. update intervals). This can be regarded as a *conservative* synchronisation protocol for parallel discrete-event simulation, which does not allow violation of data dependencies and causality errors to ever occur [12]. The limitation of using global barriers is that all LPs have to wait at a global barrier even though some LPs do not have data dependencies at certain barriers. This method does not fully exploit the parallelism of the simulation. Another type of conservative synchronisation protocol allows LPs to progress in

an *asynchronous* manner [19, 20]. "Asynchronous" means that no global barriers exist and not all LPs need to perform synchronisation at the same virtual simulation time. In [20], each LP collects time bounds from all neighbouring LPs within which it can progress without synchronisation. The minimum of the time bounds is taken as a time window. Then, the LP synchronises with all neighbouring LPs at the end of the time window. This is performed iteratively. The drawback of this approach is that LPs still need to communicate with *all* neighbouring LPs in a synchronisation operation, which is unnecessary, because some LPs may not have data dependencies at the end of time windows. For an asynchronous protocol, *lookahead* of LPs determines the frequency of synchronisation. Lookahead of LP_i towards LP_j at simulation time t is a time interval in the simulated future within which LP_i will not have data dependencies with LP_j . The larger the lookahead values are, the less synchronisation operations are required. The lookahead in agent-based traffic simulations is usually small due to the frequent interactions of agent. Using an asynchronous protocol solely does not solve the issue of high synchronisation overhead in parallel agent-based traffic simulations if the lookahead is small. Therefore, new strategies need to be proposed to improve the synchronisation.

To summarise, agent-based microscopic and nanoscopic road traffic simulations are useful tools for studying road traffic at the level of the behaviour of individual DVUs. Performing such simulations for a large area is computationally intensive. Parallel simulation can be conducted to make use of more computational resources. There are two major bottlenecks for an efficient parallel traffic simulation: workload imbalance and synchronisation. The load balancing, and synchronisation issues in parallel agent-based traffic simulation have not been sufficiently addressed by the past research.

1.2 Problem Statements

The objective of this research is to reduce the execution time of parallel agent-based traffic simulation. To achieve the objective, two particular issues are to be addressed: i) partitioning and balancing workload of LPs; and ii) reducing synchronisation overhead by making use of the characteristics of agent models in the simulation. The details of

the two issues are described below.

The first issue is on partitioning and load-balancing. Three objectives are considered in partitioning the road network: minimising workload imbalance, minimising synchronisation messages by minimising neighbouring partitions, and minimising communication data volume by minimising edge cut. Using geographical information only will not achieve all the objectives. Thus, road network must be treated as a weighted graph and partitioned using both graph-based algorithms and geographical information. Optimising the aforementioned objectives is not straightforward, because there may be trade-offs among them. For instance, to achieve load balance, a cluster of agents on the road network may be divided into different partitions. However, dividing the cluster may result in synchronisation overhead. Moreover, graph partitioning is a NP-complete combinatorial problem. The optimal solution may not exist or may not be practical to find using available computational resources, therefore, a heuristic must be developed to obtain feasible results. To achieve dynamic load-balancing, load-balancing operations need to be performed, where a partitioning algorithm will be used to repartition the road network from scratch.

The second issue is on reducing synchronisation overhead by improving the lookahead for asynchronous protocols. Because data dependencies are determined by the models of agents, the behaviour of agents must be studied to explore the potential lookahead. It is not a straightforward task to increase the lookahead since the interaction of agents causes frequent data dependencies among LPs. We have considered two ideas of improving lookahead and reducing synchronisation operations.

The first idea is to relax the synchronisation, provided that relaxation does not alter simulation results statistically. Relaxation can be achieved by either ignoring some data dependencies or reducing data dependencies by manipulating agent models. For instance, some synchronisation operations can just be skipped or agent models that incur data dependencies can be updated less frequently. It is obvious that relaxation may cause causality errors or violations of data dependencies, which may distort simulation results. The more relaxation is introduced, the more likelihood that simulation results

will be distorted. The challenge here is to maximise relaxation and at the same time keep simulation results statistically unaltered. It is non-trivial to study the behaviour of road traffic for a reasonable approach of synchronisation relaxation.

The second idea is to hide data dependencies by conducting computation replication. Using computation replication, LPs can compute the data received originally from synchronisation operations locally, so that those synchronisation operations will not be required. However, redundant computation is an overhead to the simulation and more data need to be exchanged among LPs to facilitate the redundant computation. There is a trade-off between the benefit and the overhead of computation replication. To reduce more synchronisation operations, more redundant computation will be required and each synchronisation operation will involve more data exchange. Therefore, it is not guaranteed that computation replication is always able to reduce the overall execution time of the parallel simulation. The challenges here are firstly to determine the suitable amount of redundant computation for achieving a certain lookahead, and secondly to determine a suitable lookahead that balances the trade-off between reduced synchronisation operations and the total overhead caused by redundant computation.

1.3 Contributions

We have conducted research on each of the issues mentioned above. The contributions are algorithms and heuristics to reduce the execution time of parallel agent-based traffic simulations. They are composed of four parts which are listed as follows.

Firstly, this research analyses the effectiveness of the existing partitioning, dynamic load-balancing, and synchronisation approaches on parallel agent-based traffic simulation. The performance of static partitioning and dynamic load-balancing are compared. Existing spatial and graph-based partitioning algorithms are evaluated. In order to generate the weighted graph from the road network, traffic density and flow are used to quantify the workload and data volume of communication. Meanwhile, the performance of synchronisation using global barrier and mutual appointment (MA) protocol is compared. The MA protocol allows LPs to synchronise in a peer-to-peer fashion

and progress *asynchronously*.

Secondly, this research proposes a graph-based partitioning algorithm used for dynamic load-balancing. The proposed graph partitioning algorithm is a heuristic that considers three optimisation objectives: minimising workload imbalance, minimising the number of synchronisation messages, and minimising data volume of communication. In order to reduce synchronisation messages, the heuristic tries to minimise the number of neighbouring partitions considering both geographical information and graph property of the road network. A design philosophy for the heuristic is to make it simple so as to keep the overhead of re-partitioning low for dynamic load-balancing.

Thirdly, this research studies the effectiveness of relaxing synchronisation for parallel agent-based traffic simulation. The relaxation of synchronisation takes advantage of the intrinsic uncertainties of traffic simulation. The uncertainties allow some noise in the simulation. Heuristics are designed to apply the suitable amount of relaxation according to traffic conditions at the boundary of partitions. Generally, more relaxation can be applied to relatively more stable traffic.

Lastly, this research explores the benefits of computation replication in reducing synchronisation overhead for parallel agent-based traffic simulation. Extended layers of partitions in the road network are defined. They determine the amount of redundant computation necessary for a certain lookahead. An analytical model of the total overhead caused by the redundant computation approach is developed. Using the model, the number of extended layers to be replicated is dynamically adjusted at simulation run-time to minimise the total overhead of redundant computation and message-passing.

1.4 Thesis Outline

This research is multidisciplinary and involves transportation engineering and computer science. Some background information of traffic simulation is explained first, and then issues of load-balancing and synchronisation in the existing parallel agent-based traffic simulations are reviewed. Subsequently, the work to solve the problems identified in Section 1.2 is presented in separate chapters. The details of the subsequent chapters are

listed below.

Chapter 2 reviews the works related to this research. Traffic models, and the existing parallel agent-based traffic simulations are introduced. Following that, the works on partitioning, dynamic load-balancing, synchronisation, synchronisation relaxation, and computation replication are reviewed.

Chapter 3 introduces a parallel agent-based traffic simulation *SEMSim Traffic* which is used to assess the performance of the existing load-balancing and synchronisation methods. The agent models, and decomposition of the simulation are first introduced. Performance issues caused by the parallelisation of the simulation are explained in detail. The performance of the parallel simulation using spatial-based stripe partitioning and graph-based METIS partitioning, with static partitioning and dynamic load-balancing, is investigated with experiments. Besides, two types of synchronisation protocols are compare: synchronisation using global barrier and MA protocol.

Chapter 4 proposes a graph partitioning heuristic that can be used in dynamic load-balancing. It uses the geographical information, as well as the connectivity of the road network to generate partitions with stripe-like shapes so as to reduce the number of neighbouring partitions for each partition.

Chapter 5 introduces the first approach of reducing synchronisation operations. Synchronisation is relaxed. Two heuristics of relaxing synchronisation are presented: one is to skip unnecessary synchronisation operations, and the other is to reduce the temporal resolutions of the agents at boundaries of partitions. The heuristics are designed for traffic simulation according to the behaviour of agents. Both heuristics ensure that simulation results are not altered statistically.

Chapter 6 presents the second approach of reducing synchronisation operations. Extended layers are defined based on the behaviours of agents and the topology of road networks. The total overhead caused by using redundant computation is analysed. The synchronisation frequency is adjusted dynamically at simulation run-time to maximise the benefit of computation replication.

Chapter 7 concludes this thesis and describes some potential directions of future work.

Chapter 2

Literature Review

This chapter reviews the existing work related to this research. Traffic models and simulations are firstly reviewed to provide some background information on traffic simulation. Following that, the works on different issues addressed in this study, namely, partitioning, dynamic load-balancing, synchronisation protocols, synchronisation relaxation, and computation replication, are reviewed.

2.1 Traffic Simulation and Models

Road traffic has been studied from different perspectives and at different levels of detail. The choice of different types of models typically depends on the purpose of study, available input data, available computational resources, and available time of the study. A summary of the typical models used in simulations across four levels of detail is shown in Table 2.1. The level of detail increases from the top to the bottom of the table.

2.1.1 Macroscopic and mesoscopic models

At the macroscopic level, traffic flow is modelled using fluid dynamics, which was pioneered by Lighthill-Whitham and Richards [29, 30]. Macroscopic models deal with the relationship of the aggregated characteristics of traffic flow: density, speed, and flow with respect to time and space [29, 31]. They describe collective phenomena such as

Table 2.1: Four levels of traffic models and examples of simulation tools

Level of Detail	Typical Models	Simulation Tool
Macroscopic	Fluid dynamics	METANET [21]
Mesoscopic	Speed-density relationship, Queueing models	MATSIM [22], DynaMIT [23]
Microscopic	Cellular automata	TRANSIMS [8]
	Car-following acceleration model, Lane-changing, Gap-acceptance	SUMO [24], MITSIM [25], VISSIM [26], AIMSUN [27]
Nanoscopeic	Driver Perception-Decision-Control, Vehicle Dynamic/Kinematic	PELOPS [28]

propagation of traffic waves and evolution of congested traffic. Macroscopic models are useful if only aggregated characteristics of traffic are of interest. One example of their applications is short-term traffic prediction which can act as a real-time route guidance tool and help to optimise traffic flow [21]. The disadvantage of macroscopic models is the inability to capture individual behaviour, due to their highly aggregated property.

Compared to macroscopic models, mesoscopic models take individual vehicles into account. They provide representation of traffic dynamics as well as simplified dynamics of individual vehicles, using the speed-density relationship and queueing theories [32]. The computational demand of mesoscopic models is still not very high, so mesoscopic traffic simulation is able to simulate large traffic networks with acceptable computational cost. Mesoscopic simulation is suitable for dynamic traffic demand modelling [23, 24, 27, 33]. It is able to make short-term traffic predictions in real-time, and thus is appropriate for applications involving the evaluation of individual vehicles and operating at or faster than real-time speed [27]. To improve the execution speed of large-scale mesoscopic traffic simulations, parallel simulation techniques have also been investigated for mesoscopic traffic simulation [34, 35]. However, mesoscopic models do not capture the detailed movement of vehicles (e.g., accelerate and decelerate) or the exact locations of individual vehicles in the street.

2.1.2 Microscopic models

In contrast to mesoscopic models, microscopic models capture the dynamics of movement of individual DVUs. These are characterised by the car-following, lane-changing, and gap-acceptance behaviour of drivers in response to the surrounding traffic [4, 36, 37, 38, 39, 40, 41]. The development of microscopic models started in the 1950s and 1960s [4, 41]. Traffic flow patterns such as shockwaves [30] are treated as emergent phenomena from microscopic individual behaviour. Microscopic models are useful for studies involving the behaviour of individual vehicles. However, detailed microscopic models require large amount of calibration data [42] and vast computational resources, which makes large-scale microscopic traffic simulation challenging. Therefore, microscopic traffic simulation is mostly used for off-line evaluation of systems. Some examples of popular microscopic acceleration models are Gipps' model [37], Wiedemann model [36], and intelligent driver model (IDM) [39]. They are in the form of ordinary differential equations. Cellular automaton model [43] is another type of microscopic models that describes the movement of DVUs in a discrete cellular space with simple localised rules. Although it is microscopic, the simplicity of the rules and discrete space make this approach less computationally intensive.

To solve the computational issue of large-scale microscopic traffic simulations, multi-resolution (hybrid) traffic simulation [44, 45, 46] and parallel traffic simulation [7, 8, 9, 18, 47] have been studied. The multi-resolution approach applies appropriate levels of detail in different areas of the simulation to reduce computational load (e.g., adding mesoscopic models in a microscopic simulation). The parallel simulation approach aims to speed up the simulation by making use of more computational resources. There are parallel simulators for large-scale microscopic traffic simulations, such as AIMSUN2 [7], PARAMICS [18], TRANSIMS [8], and the recent XAXIS [9]. AIMSUN2 is designed for a shared-memory environment using multi-threading. XAXIS also uses multi-threading based on a partitioned global address space system, i.e., an abstracted shared-memory space. Threads are able to access data through a shared global memory address. PARAMICS and TRANSIMS are designed for a distributed-memory environ-

ment where processes can only access the data from other processes through message-passing.

2.1.3 Nanoscopic models

Besides the three approaches of modelling the road traffic described above, traffic modelling and simulation has been extended to the *nanoscopic* level (a.k.a, sub-microscopic or picoscopic [5, 28, 48, 49, 50]). Instead of expressing the driver and the vehicle as a compound DVU model as in microscopic simulation, nanoscopic traffic models explicitly define the driver and the vehicle as distinct entities. The driver can contain three levels of behaviour models: strategical, tactical, and operational, as shown in Figure 2.1. Strategical behaviour describes the strategy (route) a driver will take to accomplish his trip; tactical behaviour describes lane changing and gap acceptance decision a driver makes when he drives to a junction or merging road, or wants to change lane; operational behaviour describes the preferred acceleration or deceleration of a driver while driving to achieve faster speed and avoid collision. For the vehicle, it contains a number of component models, such as, an engine model and a battery model of an electric vehicle (EV). The engine model can constrain the acceleration and deceleration of the vehicle, as shown in the example in Figure 2.1.

Figure 2.1: Illustration of driver behaviour models and vehicle component models

Traffic simulations are considered as nanoscopic if they contain either detailed driver behaviour models [51] or vehicle models [52, 53], or both [5, 28, 50]. By modelling the traffic at this level of detail, questions regarding the effect of individual vehicle components or driver behaviours, such as the influence of risky driving or EVs with different battery capacities, on the overall traffic can be analysed. Nanoscopic traffic simulation has been used for the analysis of advanced driver assistance systems in cars [51]. Both nanoscopic and microscopic traffic simulations can be conducted in an agent-based approach, since they both deal with entities (i.e., DVUs) that interact and have certain behaviour and goals.

Simulating traffic at nanoscopic level of detail is even more computationally intensive than microscopic simulation. Similar to microscopic simulation, to accelerate large-scale agent-based simulations, parallel simulation can be conducted. Compared to using hybrid models, it does not sacrifice simulation resolution. The parallelisation of agent-based (i.e., microscopic and nanoscopic) traffic simulations are reviewed in the rest of this chapter.

2.2 Partitioning and Dynamic Load-balancing of Parallel Agent-based Traffic Simulation

2.2.1 Introduction to partitioning and dynamic load-balancing

The partitioning of a simulation can be static or dynamic. When it is static, the partitioning is performed only at the beginning of the simulation and partitions are fixed throughout the simulation. Static partitioning sometimes makes use of a priori knowledge about the simulation to help make partitioning decisions, for instance, traffic condition information was used to help achieve load balance in [54]. It is attractive for applications of which the execution time of LPs and their communication requirements are predictable. Often, in some dynamic applications repartitioning during the simulation run-time is necessary, which is referred to as *dynamic load-balancing*. Dynamic

applications are the simulations in which workload of LPs is unpredictable or changing during run-time, such as road traffic simulation, molecular dynamics simulation, and crowd simulation. The primary advantage of static partitioning is the absence of recurrence of run-time overhead. However, the computation and communication costs of LPs may not always be predictable or stay stable. For a parallel agent-based traffic simulation that is decomposed by cutting the road network, the workload of LPs is generally dynamic. Therefore, dynamic load-balancing has great potential of outperforming static partitioning even though it incurs certain run-time overhead.

The load-balancing problem is not limited to road traffic simulation. It is generally involved in any parallel application. A dynamic load-balancing algorithm consists of four components: a load measurement rule, an information exchange rule, an initiation rule, and a load-balancing operation [10].

Load measurement refers to the quantification of workload of processes. The workload of each process should be quantified by a *load index* before any further decision of load-balancing is made. The calculation of load index should be very efficient, since it may occur very frequently.

An *information exchange rule* specifies how processes collect and maintain the workload information of other processes necessary for making load-balancing decisions. Workload information can be kept by either a master process or individual processes depending on the type of the load-balancing algorithm (i.e., master-slave or peer-to-peer). Ideally, this information should be most up-to-date. However, exchanging workload information also causes communication overhead. Therefore, there is a trade-off between the communication overhead caused by frequent information exchange and the accurate view of the workload [10].

After load exchange, an *initiation rule* is used to decide whether a load-balancing operation should be initiated. Since a load-balancing operation incurs overhead, the rule must evaluate the expected benefit gained from load-balancing operation against the overhead so as to determine whether a load-balancing operation is profitable. The operation can be initiated by an overloaded process (*sender-initiated*), or by an underloaded

process (*receiver-initiated*), or *periodically* at run-time. To facilitate the sender-initiated and receiver-initiated policies, a common practice is to define imbalance thresholds, i.e., an upper threshold for overloaded situations and a lower threshold for underloaded situations [10].

A *load-balancing operation* may be invoked by the initiation rule. To perform load-balancing, three decisions should be made: the sources and destinations of workload transfer, how much workload needs to be re-distributed among processes, and which part of the workload should be transferred to the destination processes [10]. Dynamic load-balancing operations can happen between neighbouring LPs (*nearest-neighbour* methods [55, 56], a.k.a. diffusive methods or incremental methods) or globally (*direct* methods, a.k.a. scratch-remapping methods or repartitioning-mapping methods [57]). In diffusive methods, LPs conduct load-balancing based on local workload and that from direct neighbours. Therefore, the global optimum may not be achieved. The repartitioning-mapping approach use global workload information and perform partitioning from scratch. The partitioning quality in terms of workload balance and edge cut using global information is usually better than those of the diffusive approach. However, the shapes of newly generated partitions may not be similar to previous partitions, thus, high overhead of relocating the workload may be caused. Graph repartitioning algorithms, such as those proposed by Hu and Blake [58], Walshaw et.al. [59], Schloegel et.al. [17], Meyerhenke et.al. [60] can be used for dynamic load-balancing for the workload that can be represented as graphs.

The stages of dynamic load-balancing are independent. Different approaches of each stage can be combined arbitrarily. A review of general load-balancing methods can be found in [61]. Efficient dynamic load-balancing methods for agent-based road traffic simulations are of interest in this study.

2.2.2 Partitioning of agent-based traffic simulation

In this subsection, the work on partitioning methods is reviewed first since partitioning has a substantial influence on both the performance of dynamic load-balancing and

the synchronisation overhead. In agent-based traffic simulations, the parallelism among agents has generally been explored, where an agent is regarded as the basic computational unit. The road network is divided into parts and agents are assigned to LPs according to their locations on the road network.

A straightforward approach to cut the road network is to make use of the geographical information of the road network (i.e., coordinates of nodes). The road network is cut into stripes or grids of equal sizes in [14,62]. The road network can also be cut with recursive coordinate bisection (RCB) method [63]. Using RCB, the road network is cut into two equally sized sub-regions by a plane orthogonal to one of the coordinate axes and then this cutting procedure is recursively applied until the desired number of sub-regions are reached. RCB partitioning method is implemented for traffic simulation in [8] and [15]. An example is shown in Figure 2.2(a). The limitation of cutting the road network using only geographical information is that they do not consider the connectivity of the network, which encodes the information of data dependencies among LPs. Hence, the minimisation of synchronisation overhead is not considered. In an environment where messages-passing is time-consuming, ignoring synchronisation overhead may be significantly detrimental to the performance of the parallel simulation.

Another partitioning approach uses graph partitioning algorithms. The partitioning problem is formulated as a graph partitioning problem by representing the workload (i.e., computation units) using vertices, and the relationships of computation units (i.e., data dependencies) using edges of a graph [64]. A graph partitioning algorithm groups the vertices of the graph to optimise certain objective functions subject to some constraints (typically, equal weights of partitions with minimised total edge cut). This approach considers not only equalising the workload, but also minimising the edge cut (i.e., communication data volume) between partitions. The graph partitioning problem is a combinatorial NP-complete problem, therefore, there are many heuristics to compute the approximate best partitioning results, e.g., multi-level graph partitioning in [16,64,65]. Genetic algorithms and evolutionary algorithms have also been used in [54,66] for partitioning. Examples of using graph partitioning algorithms in traffic simulations are

Figure 2.2: Examples of RCB and graph partitioning: (a) Portland road network partitioned using RCB reported in [8]; (b) Tokyo road network partitioned using METIS partitioning reported in [9].

reported in [8,9,54]. An example of partitioning Tokyo road network using METIS [16] is shown in Figure 2.2(b) [9].

Graph partitioning algorithms perform better than the geographical information based partitioning algorithms in terms of edge cut. However, the optimisation objectives of general graph partitioning algorithms still do not ideally match the objective of minimising the synchronisation overhead of parallel simulation, as pointed out in [64]. Edge cut only indicates the data volume during synchronisation. Minimising the total edge cut does not imply a minimisation of synchronisation overhead. Another crucial objective is to minimise the number of synchronisation messages [64]. How LPs send messages depends on the synchronisation protocol. Specifically for the MA protocol (to be described in Section 3.5), the number of communication messages is affected by the number of neighbouring partitions that each partition has. Therefore, the number of neighbouring partitions should be considered as an optimisation objective as well. In addition, graph partitioning algorithms are generally slower to compute than geographical methods. Some graph partitioning algorithms may not be suitable for dynamic load-balancing due to the run-time overhead.

For completeness, another partitioning method that should be mentioned is *scatter partitioning* used in [7]. This method is different from other methods that try to group nearby computation units, in terms of road links, into one partition. It distributes nearby road links to different partitions. This usually results in a highly balanced workload

distribution, because the traffic densities on nearby roads are similar. However, the partitioning method generally incurs high communication overhead due to a large amount of data dependencies between partitions. It is most suitable for traffic simulators that are designed for a shared memory environment where communication overhead is trivial. For this reason, this type of partitioning is not considered in our work.

2.2.3 Dynamic load-balancing in parallel agent-based traffic simulation

Dynamic load-balancing is used in simulations that demonstrate a significant shift of workload during simulation run-time; whereas it is usually not required for simulations parallelised using multi-threading in a shared-memory environment, since workload can usually be distributed evenly among threads [7]. The four steps of general dynamic load-balancing methods described in Section 2.2.1 also apply to parallel agent-based traffic simulation.

For load measurement in parallel agent-based traffic simulation, a common approach is to count the number of agents [15, 67, 68]. This may not be an accurate measure if agents have heterogeneous computation workload or the processors which run the LPs have different computing capacities. A more sophisticated way is to consider the average execution time of agents, which takes both the workload of the application and the possible influence of hardware utilisation into account [13]. However, the extra complexity of the measurement may not result in performance improvement for processors with similar workload. Load measurement is usually performed periodically at run-time in a distributed manner. The subsequent three steps can all be conducted either in a *distributed* manner where all processes make individual decisions, or a *centralised* manner where a master process makes all decisions. The load information exchange is also performed periodically. The exchange happens either between direct neighbours [13, 68] or between a master process and slave processes [13].

A load-balancing operation can be initiated either by processes with heavy or light workload, or by a master process which oversees the global load distribution in a cen-

tralised way [13]. Regarding the invocation criteria, a fixed imbalance threshold is used in some work [67, 68]. The threshold value plays a critical role in the performance of load-balancing. A smaller threshold triggers load-balancing operations more frequently and tends to result in better load balance; however, a worse overall speed-up may be obtained as a result of the higher communication overhead required by load-balancing operations [68]. To make an informed invocation decision, a profitability estimation can be included to ensure that the overhead of performing a load-balancing operation is less than the benefit of the operation [13]. When stripe and grid partitioning methods are used, dynamic load-balancing can be achieved easily by incrementally sweeping the boundary lines from heavily loaded simulation processes to the lightly loaded ones [13, 67, 68]. However, as mentioned earlier, these methods do not consider the synchronisation overhead of LPs, which may diminish the performance of parallelisation for real-world road networks having irregular shapes. To the best of our knowledge, graph partitioning algorithms have not been used in the dynamic load-balancing of parallel agent-based traffic simulation.

2.3 Time Synchronisation in Parallel Agent-based Traffic Simulation

2.3.1 General synchronisation protocols

Time synchronisation in parallel and distributed discrete-event simulation has been extensively studied. Synchronisation protocols broadly fall into two categories: *conservative* and *optimistic* [11, 69]. Conservative methods strictly avoid the possibility of causality errors [70]. In contrast, optimistic methods use a detection and recovery approach. The execution of events does not necessarily follow the order of their time stamps. Causality errors are detected during the simulation, and a roll-back mechanism is invoked to recover from the errors [71]. We present a brief review on the synchronisation protocols for parallel and distributed discrete-event simulation in this subsection.

In conservative synchronisation, LPs need to predict the events that they may send to other LPs and send time-stamped messages to notify the relevant LPs about the events. Based on the time stamps of messages received, LPs determine the local events that are safe to process without causing causality errors. The efficiency of conservative protocols depends on the ability of LPs in predicting future events that may affect other LPs, which is represented by a measure termed as *lookahead*. Lookahead of an LP towards another LP equals Δt if the messages sent by the LP at t do not affect the simulation in the receiving LP before time $t + \Delta t$. It is obvious that larger lookahead results in less synchronisation for parallel simulation using conservative methods.

A conservative synchronisation can be conducted in a *synchronous* manner (e.g., using global barriers or global reductions). All LPs are involved in a synchronisation operation. Correspondingly, the lookahead calculation of an LP involves all other LPs. Early works include Lubachevsky's bounded lag algorithm [72], Nicol et.al.'s YAWNS protocol [73], and Steinman's Time Buckets protocol [74]. In those works, LPs periodically synchronise globally, and cooperatively calculate a time window within which they process events. However, global communication may still cause a performance bottleneck. Conservative simulation time management can also be achieved in an *asynchronous* manner, where synchronisation operations do not need to be performed globally with all the LPs [12, 70]. An LP determines the events safe to process only based on the lookaheads from the LPs that directly affect it. The pioneer work for this type of synchronisation protocol is the null message algorithm originally developed by Chandy and Misra [70] and Bryant [75] independently. Variations of the Chandy-Misra-Bryant (CMB) null message algorithm are later developed aiming to improve it, such as message sending on demand [76] and the carrier-null message approach [77]. The concept of conservative time windows was proposed in Ayani and Rajaei's work [19]. Conservative time windows are the simulation time intervals within which events are safe to process. LPs calculate the windows using the time bounds received from all neighbouring LPs. Another notable work is Nicol's appointment protocol, where LPs make synchronisation appointments of sending messages with individual neighbouring LPs [78]. It is differ-

ent from conservative time windows in that LPs do not calculate unified time windows for all neighbouring LPs, in other words, LPs can have different time windows for different neighbouring LPs. Conservative protocols perform well in applications where large lookahead can be exploited, such as manufacturing lines [79] and queueing networks [78]. In the queueing network, lookahead is the predicted time when a job leaves a queue and joins the next queue that is in another LP [78].

For optimistic synchronisation, the best-known protocol by-far is the Time Warp mechanism [71]. LPs process their local events without considering possible causality errors first and advance the local simulation time accordingly. If there is any message received with a time stamp smaller than those of the processed events, the processed events that have larger time stamps should be "undone", because causality errors have occurred. The undoing process is known as a roll-back. There have been many approaches of conducting roll-back, including state-saving and reverse computation [80,81,82]. Compared to the conservative approach, the optimistic approach can generally exploit higher degrees of parallelism, because conservative protocols tend to force a sequential execution even when it is unnecessary. However, the optimistic approach needs to deal with the memory usage for storing states for roll-back if state-saving is used, or the overhead of performing roll-backs using reverse computation. Therefore, in the literature, there are many works focusing on reducing the amount of roll-backs or improving the efficiency of roll-backs. Examples are event lazy cancellation [83], incremental state-saving [84], and combining state-saving and reverse computation for roll-back [82].

2.3.2 Time synchronisation of parallel agent-based traffic simulations

Designing efficient time synchronisation protocols usually involves tuning the protocols to the models used in particular simulations. Existing microscopic and nanoscopic traffic simulations are conventionally executed in a time-stepped fashion [7,8,9,18,47]. There have been many approaches to conduct time synchronisation for these simulations. The most frequently used one is to employ a global barrier at the end of each update interval

due to its simplicity. For the simulations parallelised for a distributed-memory environment, LPs exchange messages and then proceed to the next update interval simultaneously [8, 9, 18, 85]. For the simulations parallelised by multi-threading, all threads access the shared memory directly at barriers [7, 47]. This simplicity frequently comes at the cost of LPs waiting at the global barriers. Hence, it does not fully exploit the parallelism of the simulation.

The work reported in [20] used an asynchronous protocol (i.e., conservative time window synchronisation protocol developed in [19]) for a microscopic traffic simulation. Each LP iteratively executes a three-phase loop: first, nominating the time stamp when the first event in the local event list affects neighbouring LPs and sending it to neighbouring LPs; second, calculating a time window based on the time stamps received from *all* neighbouring LPs; and third, processing the events in the local event list within this time window. LPs can progress to the end of time windows without communicating with neighbouring LPs. However, this is inefficient since LPs still need to communicate with *all* neighbours in a synchronisation operation.

Optimistic synchronisation has also been studied for agent-based traffic simulations in [86, 87]. LPs are allowed to progress over a synchronisation point and violations of data dependencies are examined. If there is a violation, the simulation is rolled-back to the point before the violation happens. As mentioned in the previous subsection, the drawback of an optimistic approach, compared to a conservative approach, is the overhead of storing states if state-saving is used for roll-back, and performing roll-backs using either saved states or through reverse computation. Our work aims to improve the efficiency of conservative synchronisation in agent-based traffic simulation¹.

2.4 Relaxation of Synchronisation

The purpose of synchronisation is to let LPs obtain necessary data for executing local events in an appropriate order to avoid causality errors. So, a fundamental question that

¹However, the methodologies developed in this thesis, namely, graph-based partitioning heuristic, synchronisation relaxation, and computation replication, are not limited to conservative protocols. They can also be adapted to optimistic protocols for improving the synchronisation.

needs to be considered is whether those causality errors make a difference to simulation results. If simulation results are not sensitive to the causality errors, the errors can be ignored. Hence, synchronisation of LPs can be relaxed by ignoring some causality errors, as long as the relaxation does not distort simulation results. This subsection introduces the work related to synchronisation relaxation.

2.4.1 Uncertainties in simulation

When modelling and simulation of a physical system is conducted, it always involves uncertainty and error [88, 89]. For example, there is uncertainty in traffic simulation [90]. Traffic simulation mimics the behaviour of the real world traffic, but there are always deviations between a simulation and the real traffic. Uncertainty can be categorised into two types according to the sources of uncertainty: aleatory uncertainty and epistemic uncertainty. *Aleatory uncertainty* is also known as stochastic uncertainty, irreducible uncertainty, inherent uncertainty, or variability [89]. It results from the inherent variation associated with the physical system, in this context, road traffic. For instance, it is highly unlikely that the traffic conditions of two days at the same location are identical. The traffic always varies from day to day. Aleatory uncertainty in a computer simulation is represented as a number of streams of random variables drawn from specified probability distributions, such as random number streams used in the distribution of agent trip starting time in a traffic simulation. Aleatory uncertainty can be quantified by repeated simulation runs with different random variable streams. In scientific studies, simulations are usually run multiple times to get statistically meaningful results.

The second category of uncertainty is *epistemic uncertainty*, a.k.a., reducible uncertainty, subjective uncertainty, or cognitive uncertainty [89]. It refers to any potential inaccuracy in the modelling process that results from a lack of knowledge about the simulated system. It can be further classified into input uncertainty and model uncertainty. The input for traffic simulation, such as traffic demand, is usually an estimation from observed or forecast real world data. *Input uncertainty* arises from the process of estimation and forecast. Models used in the traffic simulation are the abstraction of the real

world. *Model uncertainty* exists in both the model formulation (e.g., assumptions are usually used for the certain function forms and variables used) and the values of model parameters (usually estimated by calibration) [90]. A review of various car-following models based on different assumptions can be found in [41]. The models also take different forms such as linear models and fuzzy logic-based models.

Traffic simulations have inherent aleatory uncertainty and potential epistemic uncertainty [90,91,92]. The results of simulations have certain variability and there is no single correct result for an instance of simulation. This indicates that a parallel agent-based traffic simulation does *not* need to produce the *exact* result as the sequential simulation; instead, a statistically equivalent result is sufficient. Thus, synchronisation of LPs can be relaxed to improve the performance of parallel traffic simulations as long as relaxation preserves the result of the simulation statistically. This is similar to the concept proposed by Fujimoto in [93] and Nicol in [94] respectively, where uncertainties of models are exploited to improve the synchronisation of parallel and distributed simulations.

2.4.2 Relaxation of synchronisation utilising uncertainty

Uncertainty has been utilised in the existing work to improve the performance of parallel and distributed simulation. An approximate time causal order was proposed in [93] to increase the concurrency of events. It takes advantage of the fact that temporal uncertainty of events always exists in simulation. It relaxes the conventional strict time stamp order of events. Events can be executed in an approximate order instead of exact time order. More events can be executed concurrently by LPs. The approach was later studied with a queueing model in Loper's work [95]. Speed-up of the parallel simulation was significantly improved. To improve the speed-up further, a small deviation of the simulation results between parallel execution and sequential execution was allowed in the work. Another similar work is reported in [94]. The simulation application analyses the delays that packets experience when they pass through switches in a switch network. Certain approximation of individual latencies of packets is accepted provided that the average latency is preserved. Allowing such approximation improves the performance of the

simulation significantly. Although the concept of relaxation can be applied to parallel agent-based traffic simulation, the way to achieve relaxation depends on the characteristics of models. The behaviour of traffic models must be studied in order to design an effective relaxation synchronisation method.

There is also work on trading the accuracy of simulation for performance. In [96], events are clustered into intervals and synchronisation is only performed at the end of the intervals. It is intended to organise synchronisation operations into a synchronous manner which is more suitable for the GPU architecture. Another study is presented in [97] which studies the effect of relaxing the synchronisation of distributed agent-based simulations. Synchronisation operations are simply skipped. However, it was not investigated how to maintain the correctness of a distributed simulation using relaxed synchronisation. In our work, simulation results are kept statistically equivalent to the original sequential simulation with no loss of accuracy.

2.5 Reducing Inter-process Communication with Redundant Computation

The concept of reducing communication at the expense of performing some redundant computation has been presented in many previous works, such as [98, 99, 100, 101, 102]. Computational tasks near boundaries of partitions are replicated and computed by neighbouring LPs. Ding and He investigated its usage in solving partial differential equation problems using the finite difference method [98]. Each processor subdomain contains one or several boundary layers that are called *ghost cells*. Ghost cells contain the most recent values of the corresponding active cells in the neighbouring LPs. They are the representatives in the local process of the original active cells for updating local cells. The buffer region in our work (See Section 3.3) is a similar concept. Normally, synchronisation is required every time-step to update ghost cells. A ghost cell expansion method is proposed in that work where multiple layers of ghost cells are replicated between processes. Hence, synchronisation can be conducted less frequently. Using replicated ghost

cell layers has significantly reduced the synchronisation overhead in a 2D static heat distribution problem. However, the simulation space analysed in the work is a regular n -dimensional grid. The ghost cell cannot be directly applied to the traffic simulation since workload in traffic simulation is distributed in an irregular spatial network. The performance model developed is not applicable to traffic simulation either, because workload in traffic simulation is dynamic. A similar approach is used in [99] for sparse matrix-vector multiplication problems. The computational elements are modelled as vertices of a graph and the dependencies between them are edges. Extra layers of computational elements expand along the edges of the graph. Although a road network can be converted to a graph, this piece of work cannot be directly applied to traffic simulation because of dynamic workload.

Computation replication has also been applied in agent-based simulations, such as [100, 101, 102]. In [100], computation replication is used in a time-stepped simulation in a 2D grid space. The simulation space is partitioned into homogeneous grid cells, and a fixed number of replicated layers are used between all grid cells throughout the simulation. The total execution time of the simulation is derived as a function of the number of expanded layers. The work has analytically and empirically demonstrated that as the number of expanded layers increases, the execution time firstly decreases and then increases. There is an optimum number of expanded layers that can achieve minimum run time. However, whether this holds true for traffic simulation needs to be studied. In [101], computation replication is used to reduce the frequency of synchronisation for time-stepped applications in a cloud environment. Trading computation time for synchronisation time in a molecule simulation is studied in [102]. However, the problem of dynamic and heterogeneous agent distribution is not considered in these works. For agent-based traffic simulation, there are two factors that need to be considered. Firstly, the definition of extended layers should fit spatial network so that redundant computation is limited to a suitable amount. The behaviour of agents on the road network should be analysed. Secondly, traffic simulation is dynamic and the distribution of agents on the road network may not be uniform. The replication algorithm should be able to timely

adjust the suitable amount of replicated layers to minimise overall synchronisation and redundant computation overhead.

2.6 Summary

This chapter has reviewed the work related to our research. Traffic models are reviewed first, then partitioning, dynamic load-balancing, synchronisation relaxation, and computation replication are reviewed.

Traffic simulation can be conducted at various levels of detail. At microscopic and nanoscopic levels of detail, traffic flow is viewed as an emergent phenomenon from the interaction of DVUs. Both microscopic and nanoscopic can be conducted in an agent-based approach, where DVUs are modelled as agents. There are parallel microscopic traffic simulators that can be used to conduct large-scale simulations. However, the partitioning and synchronisation methods in those simulators are not optimised.

Partitioning using only geographical information of the road network may incur high synchronisation overhead due to large communication volumes. Meanwhile, general graph partitioning algorithms do not consider the minimisation of neighbouring partitions, which may incur high synchronisation overhead as well due to a large number of synchronisation messages. Dynamic load-balancing generally accelerates parallel simulations by minimising workload imbalance of LPs during the simulation. Partitioning algorithms used in load-balancing methods play a critical role in the overall performance improvement.

Synchronisation is required among LPs to exchange data and maintain the correctness of causality of events. The effectiveness of synchronisation protocols is largely dependent on the behaviour of models in simulations. Synchronisation in agent-based traffic simulations is commonly done by applying global barriers at the end of time-steps. An asynchronous protocol may explore more parallelism of the simulation. However, there is still an issue of small lookahead. Small lookahead means frequent synchronisation operations if conservative methods are used. Hence, there is a need to improve the lookahead.

Making use of uncertainties to accelerate parallel simulations has been used in queueing networks and switch networks. There are also uncertainties in traffic simulation. Relaxation can be used to reduce synchronisation operations in parallel agent-based traffic simulations as long as the relaxation does not alter the simulation significantly. To control the amount of relaxation and achieve good improvement on simulation performance at the same time, characteristics of traffic models must be studied.

Another approach of reducing synchronisation operations is to replicate some computation. The concept has been used in parallel applications, such as matrix multiplication and partial differential equation problems. However, the existing solutions cannot be applied to traffic simulations due to the topology of road networks and the dynamic nature of traffic. It is necessary to develop algorithms suitable for parallel agent-based traffic simulations.

Chapter 3

Parallel Agent-based Traffic

Simulation: SEMSim Traffic

3.1 Overview

In this chapter, we aim to understand the effectiveness of the existing partitioning and synchronisation methods for our parallel agent-based traffic simulation. The limitations and challenges will be identified.

First, the agent-based discrete-event simulator, SEMSim Traffic [103], will be introduced. The simulator is designed to flexibly accommodate the various models of microscopic and nanoscopic traffic simulations. The agent-based models and the time advancement of the simulation are described.

Second, the decomposition of the simulation for a distributed-memory environment is described. The decomposition incurs data dependencies between LPs. LPs exchange data only through message-passing. Two major overheads of parallelisation, i.e., workload imbalance and inter-process communication, are identified.

Third, partitioning of the road network and a dynamic load-balancing strategy are presented. To partition a road network, it needs to be converted to a weighted graph first. Then, partitioning algorithms operate on the weighted graph. A dynamic load-balancing

operation is triggered when the workload imbalance of LPs exceeds a certain threshold. A repartitioning of the road network is performed to balance the workload.

Fourth, the mutual appointment (MA) synchronisation protocol is introduced. It allows LPs to progress asynchronously and does not use global barriers. It is a conservative protocol in which no causality errors are allowed. The frequency of message-passing depends on the lookaheads of LPs. Lookaheads are calculated by LPs using the positions of agents in the road network.

Finally, experiments are conducted to evaluate the performance of the partitioning algorithms, dynamic load-balancing methods, and synchronisation protocols. Synchronisation using global barrier and MA protocols are evaluated; static partitioning and dynamic load-balancing are compared; and stripe partitioning and METIS partitioning are evaluated.

3.2 Simulation Models and Time Advancement

3.2.1 Simulation space

The simulation space of SEMSim Traffic is a *spatial network* that represents the road network in the real world, and consists of links and nodes. Links represent roads. Nodes contain the connectivity information of links. Nodes possess geographical coordinate information (i.e., longitude and latitude). A link connects two nodes. For the case of a traffic junction, it is represented as multiple inter-connected links. The traffic flow on a link is unidirectional from the *start node* to the *end node*. Each link has a speed limit and can have multiple lanes. The spatial location of an agent on the road network can be marked by a link id and its displacement of the link. A *displacement* of a point on a link is the distance between that point and the start node of the link.

3.2.2 Agents

In agent-based modelling and simulation, a system is modelled as a collection of autonomous decision-making entities called *agents* which individually assess the environ-

ment and make decisions on the basis of a set of rules [6]. In SEMSim Traffic, a DVU is an agent, which consists of a driver and a vehicle and interacts with other agents in the environment. The driver is realised by various driver behaviour models. For instance, car-following and lane-changing models make driving decisions at the operational and tactical level, and routing model makes driving decisions at the strategical level. Similarly, the vehicle is realised by models that characterise various vehicle components. For example, in the case of EVs, models of the battery and major energy consuming components, such as the air conditioning unit, should be included.

An agent has a *state* which is composed of many state variables. State variables are updated as the simulation progresses by executing time-stamped *events*. Simulation time advances directly from the time-stamp of the current event to the time-stamp of the next event. The events are scheduled by driver behaviour models and vehicle component models. The state variables are distinguished as two types: *agent-based state variables* and *component-based state variables*. Agent-based state variables are updated by driver behaviour models and are visible to other agents. For example, the position of a DVU on the road network is an agent-based state variable and it is often known by nearby DVUs on the road. Component-based state variables are the state variables updated by the vehicle component models and are only visible among the models inside an agent. For example, a battery model has a state of charge variable which indicates the amount of energy left, and it is not visible to other DVUs.

Each agent has front and back *sensing ranges* within which the states of other agents affect the agent's behaviour. So, it is important that an agent has access to the agent-based state variables of other agents in its sensing range. We say that an agent *subscribes* to the agent-based state variables of agents in its sensing ranges. Therefore, events that modify agent-based state variables influence other agents that subscribe to their variables. For example, in order to determine the preferred speed, an acceleration behaviour model needs to be able to access the positions and speeds of the agents ahead on the road. An illustration of sensing ranges is shown in Figure 3.1. Agent *B* has a front sensing range γ_f , and a back sensing range γ_b .

Figure 3.1: Agent B with front sensing range γ_f and back sensing range γ_b in a road network. Agent C is in its sensing range; thus, agent B subscribes to agent-based state variables of agent C .

We make an assumption that the events that update the agent-based state variables are scheduled periodically with a fixed interval. This assumption affects the synchronisation of LPs for the parallel simulation, which will be discussed in the next section. The interval is referred to as an *update interval*, denoted as δ . This fits to most microscopic and nanoscopic driver behaviour models since they are assumed to be executed with a certain resolution. The update interval can be interpreted as representing the driver's finite attention to the traffic: drivers only look at the traffic and adapt to the situation with a resolution of δ . Other events that do not modify agent-based state variables are scheduled in an event-driven manner. For instance, a charging event of an EV is triggered by a low battery charge. They are internal to an LP and does not affect the behaviour of other LPs.

3.3 Parallelisation in a Distributed-memory

Environment

3.3.1 Decomposition of the simulation

LP architecture and data read dependency

A discrete-event simulation has different levels of parallelism that can be exploited to achieve execution acceleration [104]. An agent-based simulation possesses inherent parallelism among agents [105]. SEMSim Traffic is decomposed by distributing agents into

different LPs. Figure 3.2 illustrates the architecture of an LP. Each LP can contain an arbitrary number of agents. LPs do not share a global memory. An LP only has access to the agents in the local space and is responsible for executing the events scheduled by the agents.

Figure 3.2: LP Architecture of SEMSim Traffic parallelised in a distributed-memory environment: an LP may contain an arbitrary number of agents. An agent is a compound of driver behaviour models and vehicle component models. Each LP has its own event queue.

It is obvious that two agents that have state subscriptions between them may be distributed to different LPs. If this happens, the subscription incurs data read dependencies between LPs, which is defined as follows:

Definition 3.3.1. [data read dependency] Data read dependency between LP_i and LP_j ($i \neq j$) at simulation time t is a condition in which there exists an agent in LP_i that has subscription of state variables of an agent in LP_j at simulation time t .

The subscribed state variables between LPs are referred to as *shared states*. Shared states are sent by the LP that updates the state variables to the LP that subscribes to the variables. The LPs that receive the shared states use the shared states to create *proxy agents* which act as representatives of the *real agents* in the original LPs. A proxy agent does not contain models or schedule events. They only hold agent-based state variables from the real agents.

Distributing agents by partitioning the road network

Partitioning of the simulation is achieved by dividing the road network into multiple spatial subregions. We denote the whole simulation space (i.e., road network) as G . The simulation space is divided into I disjoint spatial subregions, G_i ($0 \leq i \leq I-1$), where $G = \cup_{i=0}^{I-1} G_i$. The agents in G_i is executed by LP_i . Hence, there are totally I LPs. Nodes and links of the road network are distributed into I partitions and links are cut. The links that are cut and spanning two partitions are named *boundary links*. A boundary link is evenly divided between two partitions. If there is a boundary link between G_i and G_j , G_i and G_j are *neighbouring partitions* and LP_i and LP_j are *neighbouring LPs*. If the direction of traffic flow on the link is from G_i to G_j , the link is an *outgoing boundary link* of G_i and an *incoming boundary link* of G_j .

To identify which states are to be shared, *buffer regions* are defined. They are the regions along the boundary of partitions with the size equal to the sensing range of agents. This is because if an agent falls inside a buffer region, it is possible that the agent is in the sensing range of some agents in the neighbouring LP so that its agent-based state variables need to be shared. We denote the buffer region for partition G_i inside G_j as $B_{i,j}$ ($0 \leq i < I, 0 \leq j < I, i \neq j$). An example is shown in Figure 3.3.

Figure 3.3: (a) A road network cut into two partitions, with $link_2$ being the boundary link; (b) view of $link_2$ from LP_1 with agent C' being a proxy agent; and (c) view of $link_2$ from LP_2 with agent B' being a proxy agent.

In Figure 3.3(a), $B_{1,2}$ is the buffer region for G_1 inside G_2 , and $B_{2,1}$ is the buffer region for G_2 inside G_1 . Given the traffic flow direction of the link, the size of $B_{1,2}$ equals the front sensing range of agents and the size of $B_{2,1}$ equals the back sensing range of agents. Agent-based state variables of B and C are shared states. The shared states of agent B are sent to LP_2 and those of agent C to LP_1 . There is a proxy agent C' of real agent C in LP_1 (see Figure 3.3(b)), and a proxy agent B' of real agent B in LP_2 (see Figure 3.3(c)).

Data write dependency

When an agent moves beyond the boundary of one partition and enters another partition, *migration* of the agent from one LP to another LP happens. In this case, there is a *data write dependency* between the LPs. Data write dependency is defined as follows:

Definition 3.3.2. [data write dependency] Data write dependency between LP_i and LP_j ($i \neq j$) at simulation time t is a condition in which there exists an agent that was inside partition G_i and changes its position to be inside partition G_j at simulation time t .

For example, in Figure 3.3(a), supposing agent B executed by LP_1 continues moving on $link_2$ and its position falls inside G_2 which is mapped to LP_2 , LP_2 will be responsible for executing future events of agent B . Hence, agent B is removed from LP_1 and created anew in LP_2 . To do so, the information about the entire agent B , including all state variables (i.e., both agent-based and component-based state variables) and model parameters, should be sent from LP_1 to LP_2 . It is obvious that the data size involved in a data write dependency is larger than that in a data read dependency.

3.3.2 Synchronisation and load imbalance issues

Data dependencies necessitate synchronisation between LPs. When there is a data dependency, an LP should hold the progress of the simulation until the dependency is fulfilled by exchanging messages with relevant LPs. Synchronisation involves sending of migrating agents and receiving of shared states. It is an overhead for the parallel simulation. Therefore, in order to perform a parallel simulation efficiently, it is necessary to minimise the synchronisation overhead.

The execution time of one update interval of the simulation mainly comprises three elements: time for calculating agent states, time for sending and receiving messages, and waiting/idle time during sending and receiving messages. Workload imbalance usually results in more waiting time, because lighter-loaded LPs need to wait for messages from heavier-loaded LPs. This is another source of inefficiency. Therefore, workload imbalance should also be minimised.

The workload of LPs is dynamic during the simulation due to the movement of agents and potential dynamic creation and removal of agents. Considering traffic patterns in the real world, people usually drive to work in the morning and drive home in the evening, which creates diverse traffic densities and flows on roads during the day. Corresponding to this real world situation, positions of agents on the road network change in traffic simulations. Also, people start and end trips at different times. Therefore, agents are added to and removed from the road network dynamically. This leads to workload changes at different areas of the road network as simulation progresses. We conducted a primary study to investigate how dynamic the workload of LPs can be. A simulation is run using the Singapore road network and an origin-destination matrix for agents generated from a trip survey of households. More detail on the data is provided in the experiment section of this chapter. The road network is partitioned into 4 parts by METIS. The footprint of the percentage of workload in each LP throughout a 19 hour simulation, from 5 am to the midnight of the same day, is shown in Figure 3.4. The warm-up period of the simulation before 5 am is not shown in the figure. We can observe from the figure that workload of LPs is constantly changing. Dynamic workload makes static partitioning ineffective. Hence, a properly designed load-balancing mechanism is required to balance the workload of LPs according to the traffic conditions at simulation run-time.

3.4 Partitioning and Dynamic Load-balancing

This section introduces how a road network is partitioned. The existing partitioning methods, stripe spatial partitioning and METIS graph partitioning, are briefly described. A dynamic load-balancing strategy using repartitioning from scratch is also described.

Figure 3.4: Percentage of workload in each LP using static partitioning with METIS. x -axis is the simulation time in seconds. Simulations starts at 5 am, which is at the point of $x = 18000$. y -axis is the percentage of workload at particular simulation times.

3.4.1 Partitioning

Since a road network does not directly contain the workload and data dependency information, partitioning algorithms do not directly operate on road networks. A weighted graph is first constructed according to the workload and data dependency information on the road network. Then, partitioning algorithms operate on the weighted graph¹.

Constructing a weighted graph from the road network

A weighted graph $\hat{G} = (V, E)$ is constructed from the road network G , where V is the vertex set and E is the edge set. A vertex of the weighted graph corresponds to a node of the road network. The vertex inherits the geographical coordinates from the node. The weight of a vertex represents workload. The edge that connects two vertices in the weighted graph corresponds to the link (or links, in the case of bi-directional traffic) between the two corresponding nodes. The weight of an edge encodes the potential data dependencies between two partitions if the two vertices of the edge are assigned to two different partitions. Vertices u and v are *neighbours* if edge $e(u, v) \in E$.

Suppose vertex v is converted from node v and the weight of vertex v , w_v , is determined by

$$w_v = \sum_{l \in CL_v} \rho_l \cdot \frac{\text{length}_l}{2} \quad (3.1)$$

¹For spatial partitioning algorithms such as stripe partitioning, only workload information is required.

where CL_v is the set of links that connect to node v , ρ_l is the traffic density of link l , and $length_l$ is the length of link l . *Traffic density* is the number of agents per unit length of a link. Equation 3.1 shows that the weight of a vertex is determined by the number of agents that are in the neighbourhood of the corresponding node of the vertex (i.e., on the halves of the connecting links).

If neighbours v and u are assigned to different partitions, the weight of the edge between them represents the potential volume of data sent between LPs due to shared states and agent migration. Data volume for shared states and agent migration can be estimated by traffic density and traffic flow respectively. Given an edge $e(v, u)$, its weight, $w_{e(v, u)}$, is determined as follows:

$$w_{e(v, u)} = \alpha_f \cdot (Q_{v, u} + Q_{u, v}) + \alpha_d \cdot (\rho_{v, u} + \rho_{u, v}) \quad (3.2)$$

where $Q_{v, u}$ and $\rho_{v, u}$ are the traffic flow and traffic density on the link from node v to node u respectively ($\rho_{v, u}$ is equivalent to ρ_l if node v and node u are the start and end nodes of link l), and α_f and α_d are their weights. *Traffic flow* is the number of agents that travel through the link in a unit of time. Equation 3.2 indicates that the edge weight is a function of both traffic flow and density.

An example of a weighted graph is shown in Figure 3.5. A small road network having four nodes and five links is converted to a weighted graph having four vertices and three edges. The weight of vertex 2 and edge $e(2, 3)$ are shown in the figure.

Traffic flow is calculated by counting the number of agents passing through a link for a period of time. Traffic density is calculated by counting the average number of agents on a link for a period of time. For the initial partitioning, traffic flow and density can be obtained by estimation or from pre-generated simulation data. For repartitioning during dynamic load-balancing, the weights can be calculated on the fly using run-time traffic flow and density information. The construction of the weighted graph is performed once at the beginning of the simulation. The weights of vertices and edges should be updated every time before repartitioning. Usually, traffic density and flow are standard outputs of a traffic simulation, therefore, calculating the weight of vertices and edges does not

Figure 3.5: Using traffic flow and density information of the road network to generate a weighted graph: (a) road network - the agents covered by a shadowed oval are counted as the weight of the node at the centre of the oval; (b) weighted graph for partitioning - only the weights of vertex 2 and edge $e(2,3)$ are shown.

incur much of additional overhead.

Stripe spatial partitioning and METIS graph-based partitioning

The partitioning on the weighted graph can be formulated as follows: A partitioning solution $\mathcal{G}$ divides graph $\hat{G} = (V, E)$ into I disjoint partitions, $\mathcal{G} = \{\hat{G}_0, \hat{G}_1, \dots, \hat{G}_{I-1}\}$. Let the set of vertices in partition $\hat{G}_i$ ($0 \leq i < I$) be V_i , then $V = \cup_{i=0}^{I-1} V_i$ and $V_i \cap V_j = \emptyset$ ($0 \leq j < I, i \neq j$).

Spatial partitioning algorithms utilise the geographical coordinates and weights of vertices to partition the weighted graph. To cut the graph into stripes, the vertices are put in a list and sorted according to their longitude or latitude depending on along which direction the stripes are cut. Then, the vertices are visited according to their orders in the list and assigned to the partitions. The procedure starts with assigning vertices to the first partition. The cumulative weights of vertices are recorded. When the total weight of vertices in the current partition reaches the average weight per partition, the next partition begins. This procedure continues until all vertices are visited.

METIS uses a multi-level approach to partition graphs. There are three phases in the partitioning process. The first phase is the coarsening phase. The graph is recursively contracted to a coarser graph until a predefined size is reached. Edges are contracted and vertices are combined during the contraction. In the second phase, the coarsest graph is partitioned using a partitioning algorithm, such as recursive spectral bisection

and graph-growing. The third phase is the uncoarsening phase, where the coarsest graph is uncontracted back to the initial graph. Refinement on the partitions is also applied during the uncoarsening phase. More details can be found in [16] and [106].

3.4.2 Dynamic load-balancing

As described in Section 2.2.1, the dynamic load-balancing strategy contains four steps: load measurement, information exchange, initiation, and load-balancing operation. A load-balancing operation is only performed during the simulation if a severe workload imbalance is detected.

Load measurement and exchange

We assume that agents have similar models and processors have identical computing capacities. Therefore, instead of using the wall-clock time to measure workload, we can simplify the measurement using the number of agents. A master-slave approach of calculating the workload imbalance is adopted. LPs periodically take screen-shots of the number of agents in its partition. Then, all LPs report their workload information to a master LP. The master LP makes a decision on whether a load-balancing operation should be initiated based on the workload information. The measurement and computation of workload imbalance are performed periodically with a fixed interval.

Initiation of load-balancing operations

For dynamic load-balancing, it is unlikely to be beneficial to perform the balancing operation at every update interval, considering the extra overhead it introduces. Therefore, an invocation rule should trigger load-balancing operations only when its benefit is greater than its overhead. A common approach is adopted, which initiates load-balancing operations when the workload of LPs reaches a certain degree of imbalance.

Let us denote workload imbalance in an update interval in terms of the number of

agents at time t as Imb^t , then

$$Imb^t = \max_{0 \leq i < I} |A_i^t| - \frac{|A^t|}{I} \quad (3.3)$$

where A_i^t is the set of agents in LP_i at simulation time t , A^t is the set of the agents in the whole simulation at simulation time t , and I is the total number of LPs. A load-balancing operation is initiated when Imb^t is greater than a threshold. The threshold value can be estimated using the potential benefit and overhead of a load-balancing operation. The benefit is the reduction of execution time due to more balanced workload and the overhead is the time for adjusting road network partitions and re-distributing agents. A heuristic of the estimation is described in Appendix A.

Load-balancing operation

Recall that load-balancing can be achieved by either incrementally moving the boundaries of the existing partitions, or repartitioning the road network anew from scratch. Incrementally changing the existing partitions usually incurs less re-distribution of workload; however, new partitions may not have optimum boundary cuts, which may incur higher synchronisation overhead that offsets the benefit of load-balancing. We adopt the approach of repartitioning from scratch, trying to obtain partitions with as high quality as possible. To do this, the weights of the vertices and edges of the weighted graph should be updated first, according to the run-time traffic flow and density on links. Then, the weighted graph is repartitioned.

Depending on how workload changes, the new partitions generated by repartitioning may not have similar shapes as the existing ones. Therefore, to reduce the overhead of re-distributing the agents, new partitions and the existing partitions are mapped. A new partition and an existing partition are mapped if they have a large overlapping area. The mapping process is shown in Algorithm 1.

$\mathcal{G}_{exist}$ is the set of the existing partitions, and $\mathcal{G}_{new}$ is the set of new partitions. There is a similarity matrix $\mathcal{S} = (s_{m,n})$, $m = 0, 1, \dots, I-1$ and $n = 0, 1, \dots, I-1$, where $s_{m,n}$ is the *similarity* between an existing partition $\hat{G}_m$ and a new partition $\hat{G}_n$. $s_{m,n}$ is

Algorithm 1: Mapping new partitions to the existing partitions

```
1 Definitions:
2  $\mathcal{G}_{exist}$     set of the existing partitions
3  $\mathcal{G}_{new}$      set of new partitions
4  $V$           set of vertices in the weighted graph
5  $s_{m,n}$       similarity between partitions  $\hat{G}_m$  and  $\hat{G}_n$ , where  $\hat{G}_m \in \mathcal{G}_{exist}$ ,  $\hat{G}_n \in \mathcal{G}_{new}$ 
6  $w_v$         weight of vertex  $v$  measured by number of agents

// 1. create similarity matrix  $\mathcal{S}$ 
7 initialise  $s_{m,n} \leftarrow 0$  for all  $0 \leq m < I$ , and  $0 \leq n < I$ ;
8 foreach vertex  $v \in V$  do
9   | if  $v \in \hat{G}_m \wedge v \in \hat{G}_n$  then
10  |   |  $s_{m,n} \leftarrow s_{m,n} + w_v$ ;
11  | end
12 end

// 2. match the partitions
13 while  $\mathcal{S}$  is not empty do
14   | find the largest element  $s_{\tilde{m},\tilde{n}}$  in  $\mathcal{S}$ ;
15   | match  $\hat{G}_{\tilde{n}}$  to  $\hat{G}_{\tilde{m}}$ ;
16   | remove the entire row  $\tilde{m}$  and entire column  $\tilde{n}$  from  $\mathcal{S}$ ;
17 end
```

quantified with the number of agents that are inside both the existing partition $\hat{G}_m$ and the new partition $\hat{G}_n$. In the mapping process, the similarity matrix is first calculated by traversing the vertices of the weighted graph. For all m and n , $s_{m,n}$ is initialised as 0. If vertex v has an existing partition id m and a new partition id n , and vertex v has a weight of w_v , the value of $s_{m,n}$ is increased by w_v . This process stops when the whole set of vertices is traversed. Therefore, the calculation of similarity matrix has a complexity of $O(|V|)$. The partitions that have the largest similarity values should be mapped together. It is done iteratively. In each iteration, the largest element in matrix $\mathcal{S}$, $s_{\tilde{m},\tilde{n}}$, is identified and the new partition $\hat{G}_{\tilde{n}}$ is matched to the existing partition $\hat{G}_{\tilde{m}}$. At the end of an iteration, the entire row $\tilde{m}$ and the entire column $\tilde{n}$ are removed from $\mathcal{S}$. Then, the next iteration begins with the new $\mathcal{S}$. The matching iterations end when $\mathcal{S}$ is empty. All partitions are matched. The complexity of looking for the largest element in $\mathcal{S}$ is $O(I^2)$ and there are maximum of I iterations. Therefore, the overall complexity of this mapping process is $O(|V| + I^3)$.

3.5 Synchronisation of LPs

Global barrier synchronisation for parallel agent-based traffic simulation is straightforward. Hence, we will not discuss global barrier synchronisation in this section. As we have discussed in the literature review chapter, we focus our work on conservative synchronisation protocols. The protocols used in the existing parallel agent-based traffic simulation are mainly based on global barrier synchronisation. Hence, this section describes a conservative synchronisation protocol named mutual appointment (MA) that allows LPs to progress asynchronously.

The concept of MA synchronisation protocol is that LPs communicate with each other by making *appointments* in a peer-to-peer fashion. Appointments are synchronisation events. The synchronisation between two LPs does not involve any other LPs. It is adapted from Nicol's appointment protocol [78]. A distinct feature of our adaptation is that the lookahead for scheduling synchronisation events is mutually agreed by the LP pair. When the lookahead of LP_i towards LP_j at time t is $l_{i,j}^t$, and the lookahead from LP_j to LP_i is $l_{j,i}^t$, the *mutually agreed lookahead* for LP_i and LP_j at time t is $\min(l_{i,j}^t, l_{j,i}^t)$. Therefore, when two LPs communicate/synchronise at a particular time, the time to conduct the next synchronisation operation is predicted by both LPs.

An illustration of how MA protocol works is shown in Figure 3.6. LP_0 , LP_1 , and LP_2 are neighbouring LPs of each other. The mutually agreed lookahead between LP_i and LP_j at time t is denoted as $\Delta_{i,j}^t$ ². At simulation time 0, $\Delta_{0,1}^0 = \delta$, $\Delta_{1,2}^0 = 2\delta$, and $\Delta_{0,2}^0 = 3\delta$, where δ is one update interval. Thus, the next synchronisation events between LP_0 and LP_1 , LP_1 and LP_2 , and LP_0 and LP_2 are scheduled at time δ , 2δ , and 3δ respectively. Future synchronisation events are scheduled according to the new lookaheads.

A synchronisation event contains two tasks: exchange data required by data read and write dependencies and schedule the next synchronisation event according to the lookahead. The processing of a synchronisation event in LP_i with time stamp t is shown in Algorithm 2.

² $\Delta_{i,j}^t = \Delta_{j,i}^t$

Figure 3.6: Illustration of MA synchronisation protocol

Algorithm 2: MA synchronisation event in LP_i at simulation time t

Data:

- A_i^t set of local agents in LP_i at t
- $SynLP_i^t$ set of LPs having appointments with LP_i at t
- $M_{i,j}^t$ set of agents migrating from LP_i to LP_j at t
- $S_{i,j}^t$ set of shared states that should be sent by LP_i to LP_j at t
- $l_{i,j}^t$ lookahead of LP_i towards LP_j at t

Result: time interval before the next appointment between LP_i and LP_j , $\Delta t_{i,j}^t$

```

1 foreach  $LP_j \in SynLP_i^t$  do
2   determine  $M_{i,j}^t$ ,  $S_{i,j}^t$ , and  $l_{i,j}^t$ ;
3   send  $M_{i,j}^t$ ,  $S_{i,j}^t$ , and  $l_{i,j}^t$  to  $LP_j$ ;
4   receive  $M_{j,i}^t$ ,  $S_{j,i}^t$ , and  $l_{j,i}^t$  from  $LP_j$ ;
5    $A_i^t \leftarrow A_i^t \cup M_{j,i}^t \setminus M_{i,j}^t$ ;
6   update the set of proxy agents with  $S_{j,i}^t$ ;
7    $\Delta t_{i,j}^t \leftarrow \min(l_{i,j}^t, l_{j,i}^t)$ ;
8   make an appointment with  $LP_j$  at time  $t + \Delta t_{i,j}^t$ ;
9 end

```

Associated with each synchronisation event, there is a set of LPs that currently have appointments with LP_i , denoted as $SynLP_i^t$. An LP only synchronises with its direct neighbours; therefore, $SynLP_i^t$ only contains neighbouring LPs. Note that $SynLP_i^t$ includes all or only a subset of the neighbouring LPs of LP_i . For each LP_j in the set $SynLP_i^t$, LP_i performs the following five steps.

1. The first step is to prepare the data for fulfilling data dependencies (migrating agents and shared states). The lookahead for the next appointment is calculated.
2. In the second step, LP_i sends all that information to LP_j . A single synchronisation message is used. At the same time, it receives a message from LP_j which contains the migrating agents $M_{j,i}^t$, shared states $S_{j,i}^t$, and lookahead $l_{j,i}^t$.

3. The third step is to update the local agent set A_i^t . It is done by removing the agents in $M_{i,j}^t$, and adding the agents in $M_{j,i}^t$.
4. The fourth step is to update the proxy agent set with the latest shared state $S_{j,i}^t$. Proxy agents outside of buffer regions are removed. Remaining proxy agents are updated, and new proxy agents are created if necessary.
5. The last step is to make the next appointment. The next synchronisation event with LP_j will be scheduled at time $t + \Delta t_{i,j}^t$, where $\Delta t_{i,j}^t = \min(l_{i,j}^t, l_{j,i}^t)$.

Lookahead $l_{i,j}^t$ is calculated as the minimum time that LP_i may influence LP_j because of shared states and migrating agents. If there are agents that are already inside buffer region $B(j,i)$ or ready to be migrated to LP_j , since agents' states are updated periodically with an interval δ , $l_{i,j}^t$ is set to δ . Otherwise, $l_{i,j}^t$ is set to the minimum time that any agent in G_i requires to travel to the buffer region $B(j,i)$. So, the worst case of lookahead is that there are always agents inside $B(j,i)$ or ready for migration, in which case the lookahead always equals to one update interval. A detailed description of how to estimate lookahead according to the positions of agents is provided in Appendix B.1.

The initial synchronisation event for an LP is scheduled at the beginning of the simulation. The targets of the initial synchronisation event of an LP are *all* its neighbouring LPs. Lookahead is calculated using the initial positions of agents. Subsequent synchronisation events are scheduled based on lookahead of LPs.

3.6 Experiments and Results

3.6.1 Implementation and hardware

SEMSim Traffic simulator is implemented using C++. Communication between LPs is realized using OpenMPI 1.8.1. Each LP is implemented using a single threaded process. The hardware used is a cluster composed of four compute nodes. The configurations of each compute node are shown in Table 3.1. Each LP is mapped to one physical CPU core.

Table 3.1: Hardware configuration of each compute node of the cluster

CPU	<i>Octacore Intel(R) Xeon(R)/2.60 GHz</i> $\times 2$
Memory	192 GB
Network between nodes	56 Gbps InfiniBand

3.6.2 Scenario set-up

Data

We have used a real world road network in the scenario. It is a network of whole Singapore that consists of approximately 43,000 nodes and 84,000 links in our representation. Agents move on the road network according to their trips whose distribution is derived from the data of the Household Interview and Travel Survey (HITS) of 2008.

Agent models and agent population

Agents contain driver behaviour models such as Intelligent Driver Model [39] (see Appendix C). They form the major computation workload of the simulation. The computation time per agent in one update interval is approximately 2 micro-seconds. The data size of a complete migrating agent is approximately 200 bytes, and the size of agent-based state variables of an agent is approximately 90 bytes. The parameter values of agent models are set based on our previous studies. The update interval (i.e., δ) is 0.5 seconds. Agents have a front sensing range of 40 meters and a back sensing range of 20 meters.

The maximum number of agents during the peak traffic hours of the day is approximately 73,000 which is smaller than the real traffic. The traffic of 19 hours from 5 am to the midnight of the same day is simulated. The warm-up period before 5 am is excluded from simulation results. The number of agents on the road network throughout the simulation is shown in Figure 3.7. It is taken from one of the many simulation runs. We can observe that the agent population is dynamic. There are two traffic peak hour periods: one in the morning and the other in the evening.

Figure 3.7: The number of agents on the road network throughout the simulation.

Partitioning and synchronisation settings

For static partitioning, the average traffic flows and densities of all links throughout the simulation are observed first in one simulation run. Then, they are used for assigning weights to the weighted graph for partitioning in the subsequent simulation runs. Two partitioning methods are used: stripe partitioning and METIS partitioning. In stripe partitioning, the weighted graph is cut along the longitude direction. An example of the partitioned Singapore road network is shown in Figure 3.8.

Figure 3.8: Singapore road network in 16 partitions generated by (a) stripe partitioning, and (b) METIS partitioning. Different colours represent different partitions.

In the next two subsections, we first compare the performance of static partitioning vs. dynamic load-balancing using barrier synchronisation, then we compare the performance of global barrier synchronisation vs. MA protocol. Each scenario is run five times.

3.6.3 Comparison of static partitioning and dynamic load-balancing using global barrier synchronisation

Workload imbalance

We evaluated the workload imbalance throughout the simulation with static and dynamic load-balancing. To get an idea how the imbalance changes using static partitioning, a sample is shown in Figure 3.9. The result is taken from one of the many simulation runs. The imbalance is represented by the number of agents, i.e., imbalance equals the difference between the maximum and the average number of agents of all LPs, as formulated in Equation 3.3.

Figure 3.9: Workload imbalance of 16 LPs throughout the simulation using static partitioning.

Severe imbalance is observed in Figure 3.9 during traffic peak hours of the day, due to the highly dynamic workload. It indicates that static partitioning using pre-generated traffic flow and density information still leads to high imbalance in some simulation periods.

We used 500 as the threshold for triggering dynamic load-balancing operations for all cases, which is estimated using Formula A.6 in Appendix A. The load imbalance of LPs are periodically checked with a fixed interval. We have tried periods of 2, 5, and 10 minutes. The overall simulation execution time was not significantly different for different reevaluation periods. Hence, we only present the results using 10 minutes here. A comparison of the load imbalance using static partitioning and dynamic load-

Figure 3.10: Comparison of workload imbalance throughout the simulation between static partitioning and dynamic load-balancing, using 16 LPs with METIS partitioning.

balancing is shown in Figure 3.10. Without losing generality, only the results using METIS partitioning and 16 LPs are shown. The result using static partitioning is prefixed with *static*, and the one using dynamic load-balancing is prefixed with *DLB*.

It can be observed from Figure 3.10 that dynamic load-balancing is able to keep the imbalance much lower than static partitioning. The sudden drops of imbalance in DLB are caused by load-balancing operations. Load-balancing operations are performed more frequently during the traffic peak hours, and less frequently during other periods. This is because the workload changes faster and more unevenly during the peak hours.

The average load imbalance throughout the simulation for all cases is shown in Figure 3.11. The simulations using stripe partitioning with static partitioning and dynamic load-balancing are denoted as *static-Stripe* and *DLB-Stripe*, respectively.

Figure 3.11: Average load-imbalance using static partitioning and dynamic load-balancing with stripe partitioning and METIS partitioning. (The smaller, the better.)

Figure 3.11 shows that with dynamic load-balancing, the average load imbalance is significantly decreased for both stripe partitioning and METIS partitioning. When using static partitioning, the average load imbalance decreases as the number of LPs increases, because each LP shares a smaller portion of the total workload. The average load imbalances are similar for different numbers of LPs using dynamic load-balancing. This is because the maximum imbalance allowed (i.e., threshold) is the same for all cases.

Synchronisation overhead indicators

The indicators of synchronisation overhead are shown in Figure 3.12, including the total number of messages, the total number of migrated agents, and the total number of shared states. The total number of migrated agents and the total number of shared states together represent the data volume of communication. The indicators do not include the overhead involved in dynamic load-balancing operations.

Figure 3.12: Indicators of synchronisation overhead using dynamic load-balancing with stripe partitioning and METIS partitioning: (a) total number of messages; (b) total migrated agents; and (c) total shared states. For all measurements, smaller values are better.

It can be observed from Figure 3.12 that for Stripe, dynamic load-balancing has increased the number of synchronisation messages and shared states and decreased the number of migrated agents, compared to static partitioning. For METIS, all three measures are increased with dynamic load-balancing, compared to those of static partitioning. This means that in order to balance workload, synchronisation overhead *may* increase, because regions with high agent density may be divided to balance workload, which produces more migrating agents and shared states. METIS generally has more synchronisation messages, and less migrated agents and shared states than Stripe. This is because stripe partitioning generates less neighbouring LPs than METIS. Messages are sent between neighbouring LPs in every update interval. The average number of neighbouring LPs of each LP with dynamic load-balancing is shown in Table 3.2.

Table 3.2: Average number of neighbouring LPs of each LP with dynamic load-balancing using stripe partitioning and METIS partitioning

	8 LPs	16 LPs	32 LPs
Stripe	1.80 ± 0.01	2.27 ± 0.02	3.15 ± 0.07
METIS	4.56 ± 0.10	6.05 ± 0.04	6.83 ± 0.04

It can be easily proved that for a 2D space, cutting the space into stripes gives the smallest number of neighbouring LPs. Each partition has maximum two neighbours. However, that does not hold true for cutting the road network into stripes. As shown in Table 3.2, as the number of partitions increases, the average number of neighbouring partitions per partition increases for Stripe. This is because when there are more partitions, the area of each stripe in the road network is smaller and long links can stretch across many stripes and connect two partitions far away. For METIS, the number of neighbouring LPs generally increases as well.

Total simulation execution time and speed-up

The execution time and speed-up of the simulation are shown in Figure 3.13. The speed-up is calculated by dividing the execution time of the sequential simulation with the execution time of the parallel simulation. (The total execution time of the sequential simulation is 9327.2 ± 339.3 seconds.)

Figure 3.13: Performance of the parallel simulation using stripe partitioning and METIS partitioning with dynamic load-balancing: (a) simulation execution time; (b) speed-up.

The best speed-up for static-Stripe is achieved at 32 LPs. It results from the less average workload per LP for 32 LPs compared to that for 16 LPs. For the other three settings, the best speed-up is achieved at 16 LPs. The decreased speed-up for 32 LPs compared to that for 16 LPs is because the synchronisation overhead has increased when the number of LPs increases from 16 to 32 and the increase is more than the decreased average workload per LP. The best speed-ups of DLB-Stripe and DLB-METIS are 6.3 and 4.2 respectively. Compared to static-Stripe and static-METIS, which have the best speed-ups of 4.8 and 3.6 respectively, dynamic load-balancing has improved the best speed-ups of stripe partitioning and METIS partitioning by 31 % and 15 % respectively. This is a result of the reduced workload imbalance using dynamic load-balancing. However, for both stripe and METIS, the performance using dynamic load-balancing is worse than that of static partitioning for 32 LPs. This is because the synchronisation overhead has been increased by conducting dynamic load-balancing, which is indicated in Figure 3.12. Using 32 LPs, the total number of synchronisation messages has increased a lot by dynamic load-balancing compared to static partitioning.

Consider either static partitioning or dynamic load-balancing, the simulation using stripe partitioning always has a shorter execution time than METIS partitioning. Figure 3.12 shows that stripe partitioning performs only better in minimising synchronisation messages than METIS partitioning for the three indicators of synchronisation overhead. This indicates that reducing synchronisation messages is more critical than reducing the

communication data volume for reducing the synchronisation overhead in our parallel simulation.

Overhead of dynamic load-balancing

The overhead of the partitioning algorithms and re-distributing agents are shown in Table 3.3. The table also shows the number of dynamic load-balancing operations.

Table 3.3: Dynamic load-balancing count and total overhead of the partitioning algorithms and re-distributing agents

	8 LPs	16 LPs	32 LPs
Stripe count	47 $\pm$ 1	40 $\pm$ 2	31 $\pm$ 2
Stripe partitioning time (s)	1.20 $\pm$ 0.07	1.33 $\pm$ 0.04	1.23 $\pm$ 0.07
Re-distributing agents time (s)	2.56 $\pm$ 0.03	2.23 $\pm$ 0.14	1.76 $\pm$ 0.26
METIS count	47 $\pm$ 2	41 $\pm$ 1	31 $\pm$ 3
METIS partitioning time (s)	4.57 $\pm$ 0.62	6.25 $\pm$ 0.23	4.21 $\pm$ 0.54
Re-distributing agents time (s)	4.93 $\pm$ 0.07	2.98 $\pm$ 0.20	1.98 $\pm$ 0.1

The number of dynamic load-balancing operations decreases as the number of LPs increases. This is because the threshold to trigger load-balancing operations is fixed, and when there are more partitions, each partition shares less workload and the chance for the imbalance to exceed the threshold is lower. Correspondingly, the overhead of performing load-balancing operations also decreases. The total overhead of partitioning and re-distributing agents are at the magnitude of seconds, which ranges from 0.1 % to 0.3 % of the total simulation execution time. This is not significant compared to the overall simulation execution time.

3.6.4 Comparison of global barrier synchronisation and MA protocol

Average lookahead

We first investigated the lookahead of LPs using MA protocol. Dynamic load-balancing with the same configuration as in Section 3.6.3 is used. The average lookahead of the simulation is shown in Figure 3.14. In the figure, the simulation using MA protocol

Figure 3.14: Average lookahead in terms of update intervals using MA synchronisation, with (a) stripe partitioning and (b) METIS partitioning.

is prefixed with *MA*. For synchronisation using global barrier, since synchronisation is conducted every update interval, the lookahead of the simulation is shown as one update interval in Figure 3.14. The simulation using global barrier synchronisation is prefixed with *barrier*.

Figure 3.14 shows that the average lookahead of the simulation using MA protocol is larger than one update interval. This is because the worst case of lookahead, which is one update interval, did not always happen. The average lookahead increases as the number of LPs increases. The reason is that with a larger number of LPs, the area of a partition is smaller and there are generally less boundary links between two neighbouring partitions. Thus, the possibility of having a synchronisation operation in an update interval is lower. Comparing Stripe and METIS, Stripe has smaller lookahead than METIS for the same number of LPs. This is because there are more migrating agents and shared states using stripe partitioning than that using METIS partitioning (see Figure 3.12).

Number of synchronisation messages

The total numbers of synchronisation messages sent during the simulation using global barrier synchronisation and MA protocol are shown in Figure 3.15.

The figure shows that MA protocol has reduced synchronisation messages compared to global barrier synchronisation. When there are more LPs in the simulation, the reduction is more obvious. This corresponds to the observation that lookahead increases

Figure 3.15: Total synchronisation messages using MA and barrier synchronisation, with (a) stripe partitioning, and (b) METIS partitioning.

Figure 3.16: Execution time using MA and global barrier synchronisation, with (a) stripe partitioning, and (b) METIS partitioning.

as the LPs increases. METIS partitioning generally has more synchronisation messages than stripe partitioning. This is because METIS partitioning has a larger average number of neighbouring LPs compared to that of stripe partitioning (see Table 3.2).

Total simulation execution time and speed-up

Next, the overall simulation execution time and speed-up of the parallel simulation using the two synchronisation methods are evaluated. The results are shown in Figures 3.16 and 3.17 respectively.

It can be observed from Figures 3.16 and 3.17 that MA protocol always improves the speed-up compared to global barrier synchronisation. The improvement comes from the

Figure 3.17: Speed-up using MA and global barrier synchronisation, with (a) stripe partitioning, and (b) METIS partitioning.

decrease of synchronisation messages. (The numbers of migrating agents and shared states are not influenced by synchronisation protocol.) The improvement of speed-up is more obvious for a larger number of LPs. This is because the lookahead is larger when there are a larger number of LPs. This hints that the the performance of MA protocol can be further improved if the lookahead can be increased.

3.7 Summary

We introduced an agent-based discrete-event traffic simulator named SEMSim Traffic in this chapter. The agent in the simulation is a composition of driver behaviour models and vehicle component models. Agents have two types of state variables: agent-based state variables and component-based state variables. These state variables are updated by events scheduled by the models in the agent. Only agent-based state variables of an agent influence the behaviour of other agents. An agent subscribes to agent-based state variables of the agents in its sensing ranges.

The simulation is parallelised for a distributed-memory environment. Decomposition of the simulation is achieved by partitioning the road network. Agents on each sub-network are executed by an LP. There are data dependencies between LPs due to subscription of agent states and movement of agents. LPs need to communicate with each other during the simulation for migrating agents and sending shared states. Both

workload imbalance and communication overhead should be minimised to shorten the execution time of the parallel agent-based traffic simulation.

To reduce the workload imbalance of LPs, a dynamic load-balancing strategy was proposed. The imbalance is periodically checked and a load-balancing operation is triggered if the imbalance exceeds a pre-set threshold. A partitioning algorithm is applied to balance the workload by repartitioning the road network. Partitioning algorithms such as stripe spatial partitioning and METIS graph partitioning can be used.

To reduce the communication overhead, MA synchronisation protocol was introduced. It enables LPs to progress asynchronously. There are no global barriers. Neighbouring LPs synchronise with each other by scheduling synchronisation events based on their mutually agreed lookahead. The performance of MA protocol depends on the lookahead of the simulation.

The performance of the partitioning and load-balancing methods described above in this chapter has been evaluated. Static partitioning and dynamic load-balancing using stripe spatial partitioning and METIS partitioning are compared. Synchronisations using global barrier and MA protocol are compared. Experiments have shown that dynamic load-balancing can improve the speed-up of the parallel simulation compared to static partitioning. Stripe partitioning generally performed better than METIS partitioning in reducing the overall simulation execution time. The major reason is that stripe partitioning results in a smaller number of synchronisation messages, and thus less synchronisation overhead. The number of messages is affected by the number of neighbouring partitions that each partition has. The number of synchronisation messages is more critical for synchronisation overhead than data volume in our simulation. MA protocol has reduced the synchronisation overhead in the parallel simulation and improved the overall speed-up compared to global barrier synchronisation. The performance of MA protocol can be further improved if lookahead of the simulation is increased.

Chapter 4

A Graph-based Partitioning Heuristic

4.1 Overview

Graph partitioning is an NP-complete optimisation problem, therefore it becomes impractical to find the optimal solution when the problem size grows to a certain degree. A heuristic is often used to obtain an acceptable result. In this chapter, we propose a partitioning heuristic that balances the workload of LPs and reduces communication overhead between LPs at the same time. Reducing communication overhead is achieved by reducing synchronization messages. Since the MA synchronisation protocol is used, synchronisation messages are reduced by reducing neighbouring LPs.

The heuristic comprises two phases: the initial partitioning phase, and the refinement phase. The initial partitioning phase tries to minimise workload imbalance, as well as the number of neighbouring LPs. The refinement phase reduces edge cut to reduce communication data volume. The heuristic is designed to be simplistic so as to be suitable for dynamic load-balancing.

Experiments have been conducted to investigate the performance of the proposed partitioning heuristic on the parallel traffic simulation SEMSim Traffic. It is used in the dynamic load-balancing strategy described in the previous chapter for the performance studies. Compared to the stripe partitioning and METIS partitioning, the proposed partitioning heuristic is able to reduce the execution time of the parallel simulation.

4.2 Neighbour-restricting Partitioning Heuristic

4.2.1 Problem definition

As defined on page 40, a partitioning solution $\mathcal{G}$ divides graph $\hat{G} = (V, E)$ into I disjoint partitions, $\mathcal{G} = \{\hat{G}_0, \hat{G}_1, \dots, \hat{G}_{I-1}\}$. Let the set of vertices in partition $\hat{G}_i$ ($0 \leq i < I$) be V_i , then $V = \cup_{i=0}^{I-1} V_i$ and $V_i \cap V_j = \emptyset$ ($0 \leq j < I, i \neq j$). When vertices are grouped into partitions, we consider three optimisation objectives, which are described below.

Let the total weight of vertices in $\hat{G}_i$ be W_i . The first objective is to minimise workload imbalance, which is defined as follows:

$$Obj_1 = \arg \min_{\mathcal{G}} \left(\max_{0 \leq i < I} (W_i) - \bar{W} \right) \quad (4.1)$$

where $\bar{W}$ is the average weight of all partitions.

Let $N = (n_{i,j})$, $0 \leq i < I$, $0 \leq j < I$, be a matrix of connectivity of partitions. If G_i and G_j are neighbouring partitions, $n_{i,j} = 1$. Otherwise, $n_{i,j} = 0$. The second objective is to minimize the total number of neighbouring partitions, which is defined as follows:

$$Obj_2 = \arg \min_{\mathcal{G}} \left(\sum_{i=0}^{I-2} \sum_{j=i+1}^{I-1} n_{i,j} \right) \quad (4.2)$$

Let the total weight of the edges connecting partition $\hat{G}_i$ and partition $\hat{G}_j$ be $W_{i,j}$ ($i \neq j$). Then, the third objective is to minimise the total weight of edges between partitions, which is defined as follows:

$$Obj_3 = \arg \min_{\mathcal{G}} \left(\sum_{i=0}^{I-2} \sum_{j=i+1}^{I-1} W_{i,j} \right) \quad (4.3)$$

4.2.2 Graph-growing partitioning

The weighted graph is first partitioned by graph-growing. This phase tries to achieve Obj_1 and Obj_2 stated in Equations 4.1 and 4.2 respectively. Starting from an initial vertex, subgraphs are grown one by one along the edges of the graph. The number of

Algorithm 3: Partitioning a weighted graph into I partitions with graph-growing

```
1 Definitions:
2  $i$       id of the partition
3  $I$       total number of partitions
4  $W_i$     total weight of vertices in the partition  $G_i$ 
5  $\bar{W}$     average total weight of vertices per partition
6  $\tilde{W}$     cumulative weight of the current partition  $i$ 
7  $w_v$    weight of vertex  $v$  in the graph
8  $Q$      priority queue in which the tuples are ordered according to the partition the
9         vertices connect to and the coordinates of the vertices

10 initialise  $i \leftarrow 0, \tilde{W} \leftarrow 0, Q \leftarrow \emptyset$ ;
11 calculate the target average workload per LP,  $\bar{W}$ ;
12 determine the initial vertex  $v_{init}$  for graph-growing;
13 push tuple  $(i, v_{init})$  into  $Q$ ;
14 mark  $v_{init}$  as enqueued ;
15 while  $Q$  is not empty do
16     pop a tuple from  $Q$ , and let the vertex be  $v$ ;
17     generate a random double number  $rdm$  in the range  $[0,1]$ ;
18     if  $(\tilde{W} \geq \bar{W} \vee (\tilde{W} + w_v > \bar{W} \wedge rdm < 0.5)) \wedge i < I - 1$  then
19          $W_i \leftarrow \tilde{W}$  ; // a new partition begins
20          $\tilde{W} \leftarrow w_v$ ;
21          $i \leftarrow i + 1$ ;
22     else
23          $\tilde{W} \leftarrow \tilde{W} + w_v$ ;
24     end
25     assign vertex  $v$  to the  $i$ th partition;
26     foreach vertex  $u$ , where  $e(u, v) \in E$  do
27         if  $u$  has not been enqueued before then
28             push tuple  $(i, u)$  into  $Q$  ; //  $Q$  is kept sorted
29             mark  $u$  as enqueued;
30         end
31     end
32 end
```

neighbouring partitions is limited by utilising the geographical coordinates of vertices to make partitions have stripe-like shapes.

Graph-growing algorithm

The graph-growing algorithm is shown in Algorithm 3. It starts with selecting an initial vertex for graph-growing. The initial vertex should be on the edge of the minimum bounding box of the graph. The minimum bounding box is a rectangle with the min-

imum area that contains all the vertices of the graph. It can be determined using the geographical coordinates of the vertices.

After the initial vertex is fixed, partitions are grown starting from it. A *priority queue* is used to control the order of vertices to be visited. The initial vertex is pushed into the priority queue (line 13) and marked as *enqueued* (line 14). Then, vertices are iteratively popped from the front of the queue (line 16). Based on the cumulative weight and current vertex weight, it is determined whether a new partition should begin (lines 17 - 24). A new partition begins under two conditions (line 18):

1. the accumulated weight of vertices is equal to or greater than the average weight; or
2. the accumulated weight plus the weight of the current vertex is greater than the average weight, and a randomly generated number in the range $[0, 1]$ is less than 0.5 (the exact average weight cannot be achieved, thus the current vertex is assigned randomly to either the current or a new partition, i.e., a new partition begins by a fifty percent chance).

Additionally, the partition id should not exceed the maximum id.

Then, the current partition id is assigned to the vertex (line 25). All adjacent vertices of the current vertex that has not been enqueued before are pushed to the queue and marked as enqueued (lines 26 - 31). Note that a vertex is enqueued together with the current partition id, which forms a *tuple* (lines 13 and 28). Partition id i in the tuple marks that the vertex being enqueued has adjacent vertices that are already assigned to partition i . This information is used to reduce neighbouring partitions, which will be explained shortly.

Graph-growing terminates when every vertex is assigned a partition id. Graph-growing can reach all vertices, because the weighted graph is a connected graph. The complexity of this graph-growing algorithm is $O(|E| + |V| \cdot \log_2 |Q|)$, where $|E|$ and $|V|$ are the number of edges and vertices in the graph respectively, and $|Q|$ is the average queue length. Each edge and vertex is visited exactly once, so traversing the graph takes $O(|E| + |V|)$ time. The time for sorting the priority queue can be achieved at $O(\log_2 |Q|)$ whenever

a tuple is pushed, thus the total time for sorting is $O(|V| \cdot \log_2 |Q|)$. The complexity is not high compared to more complex heuristics, such as spectral bisection [107] and evolutionary algorithms [66]. Therefore, heuristics to reduce the size of the graph for reducing the partitioning time, such as multi-level coarsening [16], are not necessary.

Sorting rules of vertices in the priority queue

The essence of graph-growing lies in sorting the vertices in the priority queue. Here, the partition id marks in tuples are used. The vertices are sorted primarily according to the partition ids marked in their tuples and secondarily according to their coordinates. Rules of comparing of vertices are as follows: for two tuples (i, v_a) and (j, v_b) , (i, v_a) ranks in front of (j, v_b) in the queue, if

1. $i < j$; or
2. $i = j$ and $|v_a.coord - v_{init}.coord| < |v_b.coord - v_{init}.coord|$, where $v_x.coord$ is the coordinate of vertex v_x along the graph-grow direction, and v_{init} is the initial vertex; or
3. $i = j$ and $|v_a.coord - v_{init}.coord| = |v_b.coord - v_{init}.coord|$ and $v_a.\lambda < v_b.\lambda$, where $v_x.\lambda$ is another attribute of vertex v_x that can break the tie between two vertices.

The first rule attempts to reduce the number of neighbouring partitions. According to Algorithm 3, when the $(i+1)$ th partition begins, there exists a set of vertices in the queue marked with tuple partition ids equal to (or less than) i . These vertices are adjacent to the vertices at the frontier of the previous partition (or partitions). They will be ranked higher than any vertices pushed later to the queue. In this way, the vertices whose adjacent vertices are in the previous partitions are popped first and grouped into as few partitions as possible.

The second rule ranks vertices by their coordinates, when they have the same associated partition id in tuples. Vertices that are closer to the initial vertex along the graph-grow direction are ranked higher. The graph-grow direction, as the name indicates, is a general line along which partitions expands. This makes partitions generated

have overall stripe-like shapes. For example, if *coord* is the coordinate in the horizontal direction, resultant partitions are positioned generally next to each other along the horizontal direction.

The third rule is used to rank two vertices, when they have equal partition ids and coordinates. For instance, the attribute used to break the tie can be the degrees or ids of the vertices.

Comparison with stripe partitioning

This graph-growing algorithm is different from cutting the graph directly into stripes. A comparison is shown in Figure 4.1. In the figure, a small graph is partitioned into three partitions. Each partition contains two vertices. Figure 4.1(a) shows the result of the graph-growing algorithm. Vertex 1 is the initial vertex. For comparing coordinates, the coordinate in the horizontal direction is used. Hence, the visiting order of the vertices is 1, 2, 5, 4, 6, and 3. In Figure 4.1(b), the graph is simply cut into three partitions vertically. We can observe that the leftmost partition and rightmost partition are also neighbouring partitions due to the long edge $e(1,4)$. For a larger and more complex road network, the edge in the graph may have very different lengths. Simply cutting the graph into stripes may result in many neighbouring partitions.

Figure 4.1: Comparison of the proposed partitioning algorithm with stripe partitioning. A simple graph is divided into three partitions (assuming vertices have equal weights). (a) Partitions are generated with graph-growing from vertex 1 towards the right side. (b) The graph is directly cut into three equal-sized stripes without considering edges.

4.2.3 Refinement with local search

After the initial partitioning, partitions are refined in order to reduce edge cut (i.e., Obj_3). Workload balance is also improved should the imbalance exceeds a threshold after the initial partitioning phase (i.e., Obj_1). At the same time, Obj_2 is used as a constraint on the refinement. We propose a local search refinement heuristic similar to Karypis and Kumar's boundary refinement heuristic [106], which is a variation of the original heuristic of Kernighan and Lin [65]. A distinct feature of our proposed local search refinement heuristic is to restrict neighbouring partitions (i.e., Obj_2). The procedure is described below.

Gains of moving vertices

First, the *internal costs* and *external costs* of the vertices at the boundaries of partitions are calculated using the weights of edges. For a vertex v in partition G_i , its *internal cost* is the total weight of the edges that connect vertex v and its adjacent vertices in the same partition. Let Adj_v be the set of adjacent vertices of v in G_i . Then, the internal cost is calculated as:

$$I_v = \sum_{u \in Adj_v} w_{e(v,u)} \quad (4.4)$$

The *external cost* of vertex v towards partition G_j (G_i and G_j are neighbouring partitions) is the total weight of the edges connecting v and its adjacent vertices in G_j . Let $Adj_v^{(j)}$ be the set of adjacent vertices of v in partition G_j . Then, the external cost towards partition G_j is calculated as:

$$E_v^{(j)} = \sum_{u \in Adj_v^{(j)}} w_{e(v,u)} \quad (4.5)$$

A vertex can have only one internal cost, but multiple external costs if its adjacent vertices are in multiple neighbouring partitions. The external cost represents the data volume of communication between neighbouring LPs. Therefore, the external costs of vertices should be reduced in the refinement phase.

The *gain* of moving a vertex from one partition to another is calculated according to the costs. It is the reduction of total external costs if a vertex is *moved*. To move a vertex means that the partition id assigned to it needs to be changed. The gain of moving vertex v from G_i to G_j is the difference between its external cost towards G_j and its internal cost, that is,

$$gain_v^{i \rightarrow j} = E_v^{(j)} - I_v \quad (4.6)$$

When $E_v^{(j)} > I_v$, $gain_v^{i \rightarrow j} > 0$. In this case, moving vertex v from G_i to G_j can reduce the total external cost (i.e., the total weight of edge cut). However, if $E_v^{(j)} < I_v$ the gain will be negative. The gains of moving a vertex towards different neighbouring partitions may be different.

Moving rules

After the gains of boundary vertices are computed, the vertices are examined for moving from their current partitions to neighbouring partitions. The order of examination is from the higher gains to the lower gains. A vertex v will be moved from its origin partition G_i to a neighbouring partition G_j in two cases: i) when the external cost can be reduced; and ii) when there is severe weight imbalance that needs to be alleviated. The first case needs to satisfy all the following criteria:

1. $gain_v^{i \rightarrow j} > 0$
2. $W_i > W_{min}$ after v is moved, where W_{min} is the minimum weight an LP must contain, that is, the weight of the origin partition is above a low threshold after moving the vertex, for example, $W_{min} = 0.9 \cdot \bar{W}$
3. $W_j < W_{max}$ after v is moved, where W_{max} is the maximum weight an LP can contain, that is, the weight of the target partition is below a high threshold after moving the vertex, for example, $W_{max} = 1.1 \cdot \bar{W}$
4. $\forall k, l$ that $n_{i,k}=0$ and $n_{j,l}=0$ (i, j, k, l being partition ids, and $n_{i,j}$ the indicator whether partitions i and j are neighbours), $n_{i,k}$ and $n_{j,l}$ remain to be 0 after v is moved, that is, moving vertex v does not create new neighbouring partitions

The second case needs to satisfy all the following criteria:

1. $W_i > W_{max}$ before the move, that is, the weight of the origin partition has exceeded a high threshold
2. $W_j < W_i$ after v is moved, that is, the move is only from a partition with more weights to another partition with less weights
3. $w_v > 0$
4. $\forall k, l$ that $n_{i,k}=0$ and $n_{j,l}=0$, $n_{i,k}$ and $n_{j,l}$ remain to be 0 if v is moved (in line with the fourth criterion in the first case)

The fourth criterion in both cases is a neighbour-restricting constraint, which is illustrated with an example in Figure 4.2. G_2 is a neighbouring partition of both G_1 and G_3 . Vertices 1 in G_2 has external costs with both other two partitions. However, it is forbidden to move to G_1 or G_3 according to the neighbour-restricting constraint, because G_1 and G_3 are not neighbouring partitions and they would become neighbouring partitions if vertex 1 were moved to G_1 or G_3 . The same applies to vertex 2.

Figure 4.2: A simple graph in three partitions. Due to the neighbour-restricting constraint, vertices 1 and 2 are not allowed to be moved to G_1 or G_3 .

Whenever a vertex is moved, its internal cost and external costs are updated, as well as the costs of its adjacent vertices. A *pass* of refinement is finished when all boundary vertices are examined for moving. At the end of a pass, the boundary vertex set is updated. Then, a new pass begins. This process repeats until there is no vertex movement in a pass or the number of passes has reached a pre-defined maximum value. The maximum number of passes is a parameter that can be tuned in order to control the overhead. The complexity of the refinement phase is $O(|V_B| \cdot \log_2 |V_B| \cdot p)$ where $|V_B|$ is the number of

boundary vertices and p is the number of passes. The factor $\log_2 |V_B|$ is for sorting the vertices according to their gains.

4.3 Experiments and Results

The performance of the proposed partitioning heuristic in SEMSim Traffic using dynamic load-balancing and MA synchronisation protocol is investigated. Comparison with METIS graph partitioning and stripe spatial partitioning is performed.

4.3.1 Set-up

The hardware and scenario set-up are the same as those in Section 3.6. The simulation was run with different number of LPs. For each setting, four simulation runs were performed using different seeds for random number generation.

4.3.2 Results

An example of the Singapore road network partitioned with the proposed partitioning algorithm is shown in Figure 4.3(a). To distinguish from stripe partitioning, partitions generated by stripe partitioning are shown in Figure 4.3(b). Different colours represent different partitions.

Figure 4.3: Singapore road network in 16 partitions generated by (a) the proposed partitioning heuristic and (b) stripe partitioning.

It can be noticed that the shapes of partitions generated by the proposed partitioning algorithm are similar to stripes. However, the partitions have zigzag boundaries, which

is the result of graph-growing and local-search refinement.

Workload imbalance

The average load imbalance throughout the simulation using different partitioning methods is compared. The proposed graph partitioning algorithm using graph-grow is denoted as GG, and the simulation using MA protocol and dynamic load-balancing with GG as *MA-DLB-GG*. The simulation using MA protocol and dynamic load-balancing with METIS partitioning [16] is denoted as *MA-DLB-METIS* and that with stripe spatial partitioning is denoted as *MA-DLB-Stripe*. The results are shown in Figure 4.4.

Figure 4.4: Average load imbalance of the simulation in terms of number of agents.

Figure 4.4 has shown that there is no significant difference between the workload imbalance generated by the proposed partitioning heuristic and that generated by other two partitioning algorithms. This is because the same threshold for initiating load-balancing operations is used in all different partitioning algorithms.

Average number of neighbouring partitions

The average number of neighbouring partitions per partition is shown in Figure 4.5.

It can be observed from Figure 4.5 that the proposed partitioning algorithm, i.e., GG, has the lowest average number of neighbouring partitions per partition. Minimising neighbouring partitions is not an optimisation objective of METIS, therefore the average number of neighbouring partitions per partition generated by METIS is much higher than that generated by GG. With 8 LPs, stripe has a similar result as GG, because when

Figure 4.5: The average number of neighbouring partitions per partition using various partitioning methods (smaller is better).

the number of LPs is small, cutting a road network into stripes can also generate a small average number of neighbouring partitions.

For all partitioning methods, the average number of neighbouring partitions increases as the number of LPs increases. However, the increase for stripe partitioning and METIS partitioning is much faster than that for GG partitioning. For stripe partitioning, the increase is because as the number of LPs increases, the stripes become thinner and there is a higher chance for long roads to span further than one stripe, which leads to more neighbouring partitions. GG partitioning has reduced this effect and the average number of neighbouring partitions is kept lower than that of stripe partitioning.

Synchronisation overhead

The three indicators of synchronisation overhead, i.e., the total number of synchronisation messages, the total number of migrated agents, and the total number of sent shared states, are shown in Figure 4.6.

From Figure 4.6(a), comparing the graph-based algorithms (i.e., METIS and GG), simulation using GG has significantly less synchronisation messages than METIS, as a result of the smaller number of neighbouring partitions. However, communication data volume has increased, which is indicated by the number of migrating agents in Figure 4.6(b) and shared states in Figure 4.6(c). This shows that there is a trade-off between minimising the number of neighbouring partitions and minimising boundary link cuts. Whether GG can reduce the execution time of the parallel simulation compared

Figure 4.6: Indicators of synchronisation overhead with various partitioning methods: (a) number of synchronisation messages sent throughout the simulation; (b) total number of migrated agents; and (c) total number of shared states.

to METIS, depends on which indicator plays a bigger role in synchronisation overhead.

Comparing GG and Stripe, GG has a smaller number of messages (especially when the number of LPs increases), with a similar number of migrated agents and shared states. As the number of LPs increases, the number of messages in Stripe increases more rapidly than that in GG, resulting from the faster increase of the average number of neighbouring partitions in Stripe than GG. Hence, GG partitioning is more scalable than Stripe partitioning.

Overall execution time of the simulation

The execution times and speed-ups of the parallel simulation against sequential execution are shown in Figure 4.7.

It can be observed that the simulation using GG always has the least execution time

Figure 4.7: Execution times and speed-ups of the simulation using various partitioning methods: (a) execution time; and (b) speed-up.

Table 4.1: Total overhead of dynamic load-balancing using the proposed partitioning algorithm

	8 LPs	16 LPs	32 LPs
Balancing operations count	48±2	41±1	30±2
Partitioning time (s)	4.42±0.16	4.94±0.18	5.00±0.27
Re-distribute agents time (s)	4.24±0.20	4.24±0.13	4.09±0.11

and the highest speed-up among the three cases. In contrast, METIS has the worst performance among the three cases, resulting from the large number of synchronisation messages. Since the MA synchronisation protocol is used, the number of messages plays a more important role than communication data volume in the overall performance of the simulation.

Stripe has a similar performance as GG with 8 and 16 LPs. However, with 32 LPs, the performance of Stripe downgrades obviously faster than that of GG. This is because with 32 LPs, more messages are sent during the simulation in Stripe than that of GG (see Figure 4.6(a)). The proposed partitioning method is much more scalable than stripe partitioning.

The overheads of dynamic load-balancing using the proposed partitioning algorithm introduced to the simulation are shown in Table 4.1. The overheads include the time for partitioning and the time for re-distributing agents.

Table 4.1 shows that the number of load-balancing operations using the proposed partitioning algorithm is similar to that of other two partitioning algorithms (see Table

3.3). The time spent on the partitioning algorithm is more than that of stripe. This is expected, since graph-growing and local-search refinement have more complexity than dividing vertices only by coordinates.

From the experiment results above, it can be inferred that the extra objective that our algorithm considers, i.e., restricting the number of neighbouring partitions, can reduce synchronisation messages compared to both general graph partitioning algorithms such as METIS and spatial partitioning algorithms such as Stripe. However, it can come at a cost of increased communication data volume.

4.4 Summary

In this chapter, a graph-based partitioning heuristic for parallel agent-based traffic simulation has been proposed. Compared to the existing graph partitioning algorithms, it considers the objective of limiting the number of neighbouring partitions, in order to reduce synchronisation messages. The partitions are first generated by growing graphs along edges. Geographical coordinates of vertices are used to make the shapes of partitions similar to stripes, trying to minimise the number of neighbouring partitions. Then, a local-search refinement is performed on the partitions to reduce edge cut. The movement of vertices is restricted during the refinement so that partitions that are not neighbouring partitions do not become neighbouring partitions.

Experiments have shown that the proposed partitioning heuristic is able to improve the speed-up of the parallel simulation compared to the commonly used METIS graph partitioning and stripe spatial partitioning. Reducing the number of neighbouring partitions reduces synchronisation messages and thus improves the overall speed-up. However, we observed that reducing the number of neighbouring partitions resulted in an increase of communication data volume. Future work can be conducted to develop a unified solution with weights for the different objectives: minimising synchronisation messages and minimising communication data volume. The objective that has more influence on the overall performance can be prioritised.

Chapter 5

Heuristics of Relaxing Synchronisation

5.1 Overview

The lookahead of LPs using MA protocol can approach to one update interval when the traffic is always dense along the boundaries of partitions. In that case, synchronisation overhead is not reduced compared to global barrier synchronisation. To mitigate this situation, we would like to seek methods that make use of the characteristics of agent models in traffic simulation to improve lookahead.

There are always uncertainties in a traffic simulation. For instance, stochastic uncertainty exists due to the variability of traffic in the real world, and model uncertainty exists since models are abstractions of the real world. Therefore, there does not exist a single correct simulation of the real world. Hence, a parallel traffic simulation does not need to produce the exact results as a sequential simulation. Statistically equivalent results are sufficient. This motivates the work presented in this chapter, that is, investigating heuristics that take advantage of the uncertainties of traffic simulation to relax synchronisation and increase the lookahead of LPs.

Two heuristics will be introduced. The first heuristic simply skips some synchronisation operations. This results in obsolete proxy agent states. A estimation function is used to update the states of proxy agents. However, there may be discrepancies between estimated states and synchronised states. This will lead to the discrepancy between the

parallel simulation and the sequential simulation. The degree of discrepancy is affected by how often synchronisation operations are skipped and how accurate the estimation function estimates the state of proxy agents. The second heuristic reduces the temporal resolutions of the agent models at the boundaries of partitions. Update intervals for the agent models that are affected by data dependencies are increased. This produces less frequent synchronisation operations, since data dependencies are reduced. The discrepancy in this case comes from the different resolutions of agent models. The challenge in designing the heuristics is to maximise the lookahead while keeping the simulation results of the parallel execution and sequential execution statistically equivalent. Different degrees of relaxation are applied to different traffic conditions on boundary links.

Performance of the two heuristics is investigated in experiments. Statistical tests are first conducted to obtain the suitable parameter values for applying relaxation without altering simulation results. Then, performance of the two heuristics is compared to the case without relaxation.

5.2 Synchronisation-skipping and Dead-reckoning

5.2.1 Relaxation Heuristic

Consider a scenario where the traffic is jammed on a boundary link, and agent states on the link such as positions and accelerations do not change. Buffer regions on the link are always non-empty, thus synchronisation is performed every update interval. However, this frequent synchronisation is unnecessary because agent states do not change and skipping some synchronisation operations does not affect the simulation result. This is an ideal case that motivates the following relaxation method.

Relaxation of synchronisation is achieved by simply increasing lookahead values and skipping some synchronisation operations. Skipping synchronisation operations may have influence on agent states. Some agents may not be migrated in time and some proxy agents may not be updated in time either. So, the consequence is that some agents use obsolete states of proxy agents as input in their models. To reduce the discrepancy

between real agent states and obsolete agent states, states of proxy agents are updated approximately with a estimation function. However, discrepancy may still exist between the states obtained by estimation and synchronisation. An illustration of possible discrepancy is shown in Figure 5.1.

Figure 5.1: Positions of agent B and proxy agent C' on a boundary link seen by LP_i . The real agent C of proxy agent C' is in LP_j ($i \neq j$). (a) Agents are at time t . (b) Synchronisation is skipped at time $t + \delta$. The position of proxy agent C' is calculated with estimation. (c) Synchronisation is performed normally at time $t + \delta$. Proxy agent C' is updated by synchronisation. Agent C accelerated in this case. (d) Synchronisation is performed normally at time $t + \delta$. Agent C changed its lane in this case.

Agent B is a local agent and agent C' is a proxy agent in LP_i at time t (Figure 5.1(a)). The real agent C of proxy agent C' is in LP_j ($i \neq j$). As the simulation progresses to time $t + \delta$, agent B is updated by LP_i , and agent C by LP_j . Proxy agent C' should be updated by a synchronisation operation with the shared states of agent C . When synchronisation is not performed, C' is updated with estimation. The state of C' updated with estimation may be different from that update by synchronisation. Figures 5.1(b) and (c) show a case where the estimated C' and the updated C' by synchronisation have different positions. Figure 5.1(d) shows another case where agent C has performed lane-change. As a result, the lanes of the estimated C' and the updated C' by synchronisation are different. The discrepancy of states of proxy agent C' may or may not lead to a discrepancy on agent B in the simulated future depending on their relative position, relative speed, and the sensitivity of drivers' behaviour models. If there is a discrepancy on agent B , the discrepancy may affect the results of the simulation.

It is not difficult to see that for a large-scale simulation, the impact of relaxation on

simulation results is influenced by the estimation function as well as the traffic condition on the boundary links. Lookahead can be increased in the traffic conditions where potential discrepancy is insignificant. A simple estimation function can just assume that agents move at constant velocities and update positions of agents accordingly. In this case, the discrepancy is correlated with how much the velocities of the real agents change.

We first analyse the characteristics of various traffic conditions. Recall that traffic condition is typically characterised by traffic density, speed, and flow. *Density* is the number of vehicles per unit length of a roadway. *Speed* is the average distance that vehicles travel per unit time. *Flow* is the number of vehicles passing a reference point per unit time. Flow is the product of speed and density. *Stability* of traffic describes how traffic is affected by perturbations (e.g., a sudden drop of speed of a vehicle). A traffic is unstable if it tends to have oscillations of speed, traffic waves, and stop-and-go traffic [108, 109]. A *stability diagram* which plots the different stabilities of traffic under different traffic density ranges is shown in Figure 5.2 [108]. The entire range of traffic density is $[0, \rho_{max}]$, inside which there are four thresholds ρ_{c1} , ρ_{c2} , ρ_{c3} , and ρ_{c4} ($0 < \rho_{c1} \leq \rho_{c2} \leq \rho_{c3} \leq \rho_{c4} \leq \rho_{max}$). They are the densities where traffic transitions between different stability states. In the stable range ($\rho < \rho_{c1}$ or $\rho \geq \rho_{c4}$), perturbations in traffic remain small and eventually dissolve. In the metastable range ($\rho_{c1} < \rho < \rho_{c2}$ or $\rho_{c3} < \rho < \rho_{c4}$), sufficiently small initial perturbations dissolve while larger perturbations develop into persistent traffic waves. In the linear unstable range ($\rho_{c2} < \rho < \rho_{c3}$), arbitrarily small perturbations will grow through time. A stability state may not exist for certain traffic models or parameter specifications. Figures 5.2(a) and (b) show two different sets of threshold values which generate different traffic patterns. Readers can refer to [108] for more detail on traffic patterns.

Skipping synchronisation operations is considered as a source of perturbation to traffic since it exerts certain disturbance on the normal movement of vehicles. Synchronisation operations can be skipped more under stable traffic and less in unstable traffic. Figure 5.2 shows that when traffic flow is very low, density is either less than ρ_{c1} or greater than ρ_{c4} . Therefore, the conceptual basis for the first heuristic is that lookahead can be larger

Figure 5.2: Illustration of stable, linearly unstable, and metastable density regimes within flow-density diagrams $Q_e(\rho)$ [108]. Traffic is stable for density $\rho < \rho_{c1}$ and $\rho \geq \rho_{c4}$; metastable in the intervals $\rho_{c1} < \rho < \rho_{c2}$ and $\rho_{c3} < \rho < \rho_{c4}$; and linearly unstable for $\rho_{c2} < \rho < \rho_{c3}$. The relative position of the thresholds with respect to the density at maximum flow Q_{max} (denoted as $\rho_{Q_{max}}$) influences the possible types of congested traffic patterns. (a) shows the situation for $\rho_{c2} < \rho_{Q_{max}}$, (b) shows the situation for $\rho_{c2} > \rho_{Q_{max}}$. The traffic flows at densities ρ_{c1} , ρ_{c2} , ρ_{c3} , and ρ_{c4} are Q_{c1} , Q_{c2} , Q_{c3} , and Q_{c4} respectively. Q_{out} is the out flow of traffic.

when flow is lower (more stable traffic) and lookahead should be smaller when flow is higher (more unstable traffic).

5.2.2 Lookahead Determination

A simplistic but effective heuristic for relaxation is proposed. It contains only one input parameter. To begin with, a time window $window_l^t$ is defined as the longest time period at time t , in which synchronisation can be skipped without affecting the traffic flow of link l . Here, l is an outgoing boundary link from partition G_i to G_j . The window is calculated using

$$window_l^t = \alpha \cdot \frac{1}{Q_l^t} \quad (5.1)$$

where α is a sensitivity factor, and Q_l^t is the traffic flow on link l at time t . When $Q_l^t = 0$, $window_l^t = maxw_l$, where $maxw_l$ is a maximum window. In fact, the physical meaning of $\frac{1}{Q_l^t}$ is the *time headway* which is the time difference between two consecutive vehicles passing a reference point on the road. Thus, this equation can also be interpreted as the time window on a link is proportional to the time headway on the link. The lookahead between two LPs should be the minimum of the time windows of all boundary links

between them. Since the appointments are negotiated by both of the synchronising LPs, for the lookahead of one LP only outgoing boundary links of the partition need to be considered. The lookahead from LP_i to LP_j at time t is

$$l_{i,j}^t = \min_{l \in O_{i,j}^t} (window_l^t) \quad (5.2)$$

where $O_{i,j}^t$ is the set of outgoing boundary links from G_i to G_j .

The sensitivity factor α controls the amount of relaxation introduced. It is a fixed value throughout the simulation. The optimal value of α is the one that maximises lookahead and does not distort the results of the simulation statistically (e.g., traffic flow of roads, or trip durations of agents). The value of α is influenced by the models used in the traffic simulation, topology of the road network, and partitioning. When a large-scale road network is simulated, discrepancies that occur on boundary links may propagate to the upstream and downstream directions of the traffic. Therefore, it is not straightforward to obtain the optimal α value. In practical applications, a naive way of conducting trial experiments can be used to obtain an appropriate α value (as illustrated in Section 5.4.2).

The lookahead determination algorithm for this relaxation method is shown in Algorithm 4. The algorithm begins with checking the local agent population: if the set of local agents is empty, the lookahead is set to the minimum value of maximum windows directly; otherwise, the outgoing boundary links are checked one by one and the minimum value of time windows is taken as the lookahead. If the lookahead is less than one update interval, it will be set to one update interval. It is evident that the $l_{i,j}^t$ is limited by the link with the highest flow. The maximum window of l is calculated using the minimum travelling time of l (length divided by speed limit).

Algorithm 4: Lookahead of synchronisation relaxed with skipping and estimation, in LP_i towards LP_j

Data:

A_i^t set of agents in LP_i at time t
 $O_{i,j}^t$ set of outgoing boundary links from G_i to G_j at time t
 Q_l^t traffic flow on link l at time t , $l \in O_{i,j}^t$
 $maxw_l$ maximum window of link l , $l \in O_{i,j}^t$
 δ an update interval

Result: lookahead from LP_i towards LP_j at time t , $l_{i,j}^t$

```
1 initialize  $l_{i,j}^t \leftarrow \text{maximum double}$ ;  
2 if  $A_i^t = \emptyset$  then  
3    $l_{i,j}^t \leftarrow \min_{l \in O_{i,j}^t} (maxw_l)$ ;  
4 else  
5   foreach  $l \in O_{i,j}^t$  do  
6     if  $Q_l^t = 0$  then  
7        $l_{i,j}^t \leftarrow \min(l_{i,j}^t, maxw_l)$ ;  
8     else  
9        $l_{i,j}^t \leftarrow \min(l_{i,j}^t, \alpha \cdot \frac{1}{Q_l^t})$ ;  
10    end  
11  end  
12 end  
13  $l_{i,j}^t \leftarrow \max(l_{i,j}^t, \delta)$ ;
```

5.3 Reducing the Temporal Resolution of Agent-based Models

5.3.1 Relaxation heuristic

The necessity of synchronisation is originated from the update of agent-based states at boundaries of partitions. If the update interval of those agent-based states is increased, synchronisation will be required less frequently. This inspired another way of relaxing synchronisation. A heuristic that relaxes synchronisation by increasing the update interval of agent-based states is introduced in this section. Increasing the update interval reduces the temporal resolutions of some agent models which may alter simulation results. The challenge is not to alter simulation results statistically.

Behaviours that agents commonly possess are acceleration, lane-changing, and gap-acceptance. Acceleration models describe the longitudinal movement of agents. Agents update their accelerations every update interval and the acceleration is kept constant within an update interval. The positions and velocities of agents are calculated by numerical integration of the acceleration through time [110]. Lane-changing and gap-acceptance models describe the discrete decisions of an agent whether to change its lane and enter another lane [111].

At time t , agents scan the surrounding environment and calculate their states for time $t + \delta$. δ represents a driver's finite attention to traffic, that is, how fast an agent is able to adapt to the surrounding traffic condition [112]. Update intervals in different studies of agent-based traffic simulation vary. They range from a fraction of a second to a few seconds [37, 112, 113, 114]. It is also reported that lengthening the update interval within a certain range does not significantly influence the stability of traffic [112].

Figure 5.3: Positions of agent B and proxy agent C' on a boundary link seen by LP_i . (a) Agents are at time t . (b) Agents are at time $t + \delta$, updated from t with update interval δ . (c) Agents are at time $t + 2\delta$, updated from $t + \delta$ with update interval δ . (d) Agent are at time $t + 2\delta$, updated from t with update interval 2δ .

Increasing update interval reduces the frequency of agents making decisions on their movement. An example of discrepancy between agent states using different update intervals is illustrated in Figure 5.3. The positions of agent B in LP_i from time t to $t + 2\delta$ with update interval being δ are shown in Figures 5.3(a), (b), and (c). C' is a proxy agent in LP_i and its real agent C is in LP_j ($i \neq j$). Agent B makes driving decisions twice. The accelerations it exerts at t and $t + \delta$ are $\dot{v}_B(t)$ and $\dot{v}_B(t + \delta)$ respectively. In comparison,

Figure 5.3(d) is evolved directly from time t with the update interval equal to 2δ . Agent B only makes driving decisions once. It has an acceleration of $\dot{v}_B(t)$ from t to $t + 2\delta$ in Figure 5.3(d). If $\dot{v}_B(t) \neq \dot{v}_B(t + \delta)$, there will be a discrepancy between the positions of agent B in Figures 5.3(c) and (d).

To minimise discrepancy, the number of agents simulated with reduced temporal resolution should be as small as possible. An agent affects other LPs when its position is inside a buffer region. Therefore, to reduce synchronisation overhead, only the agents that are inside buffer regions are required to be simulated with a reduced temporal resolution (i.e., larger update interval). The buffer regions between G_i and G_j are $B_{i,j}$ and $B_{j,i}$. Let us denote the relaxed update interval as $\delta_{i,j}^t$ for $B_{i,j}$ and $B_{j,i}$ at time t , where $\delta_{i,j}^t = k \cdot \delta$ ($k \in \mathbb{Z}, k > 1$)¹. The agents inside $B_{i,j}$ and $B_{j,i}$ are simulated with update interval $\delta_{i,j}^t$ at time t . Other agents outside buffer regions are simulated with the original update interval δ . Since agents move on the road network, when they move into $B_{i,j}$ and $B_{j,i}$, their update intervals are increased to $\delta_{i,j}^t$. When they exit the buffer regions, their update intervals are reverted to the original value, δ . In other words, update intervals of agents will change when they move through a boundary link between partitions.

5.3.2 Lookahead determination

Recall that traffic density is segmented into five ranges in the stability diagram in Figure 5.2. We propose that relaxed update intervals in this relaxation heuristic are determined according to traffic densities of boundary links. The amount of relaxation should be customised to stable, metastable, and unstable traffic states. The heuristic utilises four density thresholds ρ_1 , ρ_2 , ρ_3 and ρ_4 . They correspond to the thresholds ρ_{c1} , ρ_{c2} , ρ_{c3} , and ρ_{c4} in the stability diagram respectively. The traffic density on boundary link l at time t is denoted as ρ_l^t , and the relaxed update interval allowed on boundary link l as δ_l^t ,

¹Note that $\delta_{i,j}^t = \delta_{j,i}^t$.

then δ_l^t is determined by the following rules:

$$\delta_l^t = \begin{cases} \delta_{gen}, & \text{if } \rho_l^t < \rho_1 \text{ or } \rho_l^t \geq \rho_4 \\ \delta_{mod}, & \text{if } \rho_1 \leq \rho_l^t < \rho_2 \text{ or } \rho_3 \leq \rho_l^t < \rho_4 \\ \delta, & \text{otherwise} \end{cases}$$

where $\delta_{gen} = k_{gen} \cdot \delta$ and $\delta_{mod} = k_{mod} \cdot \delta$ which satisfy $k_{gen} \in \mathbb{Z}$, $k_{mod} \in \mathbb{Z}$, and $1 < k_{mod} < k_{gen}$. The rationale for the relaxation heuristic is as follows:

1. When the traffic density of a link is below ρ_1 , vehicles are travelling freely on the road. When the density is higher than ρ_4 , the traffic is jammed. In both cases, relevant agent models can have a *generously* relaxed update interval δ_{gen} .
2. When the density of a link is between ρ_1 and ρ_2 , or between ρ_3 and ρ_4 , the traffic is in a metastable state. A moderate amount of relaxation can be applied. Relevant agent models have a *moderately* relaxed update interval δ_{mod} .
3. Otherwise, the traffic density is between ρ_2 and ρ_3 . The traffic on the link is highly sensitive to perturbations. Drivers may tend to be more attentive to traffic, so relaxation should not be applied. The update interval remains unchanged.

ρ_1 , ρ_2 , ρ_3 , and ρ_4 can be tuned to adjust the density ranges that allow different degrees of relaxation. Their values can be obtained by trial experiments, as illustrated in Section 5.4.2. The values of δ_{mod} and δ_{gen} can be tuned to adjust the update intervals for different degrees of relaxation. Their values can be estimated by the modeller, since the update interval corresponds to the reaction time of drivers. Note that the stable state or metastable state may not exist when density is high (i.e., $\rho_3 = \rho_4 = \rho_{max}$), which depends on the traffic models and parameter values used in the simulation. The relaxed update interval $\delta_{i,j}^t$ takes the minimum value of the allowed relaxed update intervals on all boundary links between G_i and G_j . Thus, potential discrepancies on all boundary links are controlled.

It is formulated as follows

$$\delta_{i,j}^t = \min \left(\min_{l \in O_{i,j}^t} (\delta_l^t), \min_{l \in O_{j,i}^t} (\delta_l^t) \right) \quad (5.3)$$

The calculation of the relaxed update interval $\delta_{i,j}^t$ is shown in Algorithm 5.

Algorithm 5: Relaxed update interval $\delta_{i,j}^t$ by LP_i

Data:

A_i^t	set of agents in LP_i at time t
$O_{i,j}^t$	set of outgoing boundary links from G_i to G_j at time t
ρ_l^t	traffic density on link l at time t , $l \in O_{i,j}^t$
δ	original update interval
δ_{gen}	update interval when generous relaxation is applied
δ_{mod}	update interval when moderate relaxation is applied
$\rho_1, \rho_2, \rho_3, \rho_4$	density thresholds separating the degrees of relaxation, $\rho_1 \leq \rho_2 \leq \rho_3 \leq \rho_4$
Result: relaxed update interval in buffer regions $B_{i,j}$ and $B_{j,i}$, $\delta_{i,j}^t$	

```

1 initialise  $\delta_{i,j}^t \leftarrow \delta_{gen}$ ;
2 if  $A_i^t \neq \emptyset$  then
3   foreach  $l \in O_{i,j}^t$  do
4     if  $\rho_l^t < \rho_1 \vee \rho_l^t \geq \rho_4$  then
5        $\delta_{i,j}^t \leftarrow \min(\delta_{i,j}^t, \delta_{gen})$ ;
6     else if  $\rho_1 \leq \rho_l^t < \rho_2 \vee \rho_3 \leq \rho_l^t < \rho_4$  then
7        $\delta_{i,j}^t \leftarrow \min(\delta_{i,j}^t, \delta_{mod})$ ;
8     else
9        $\delta_{i,j}^t \leftarrow \delta$ ;
10    end
11  end
12 end
13 send  $\delta_{i,j}^t$  to  $LP_j$ , and receive  $\delta_{j,i}^t$  from  $LP_j$ ;
14  $\delta_{i,j}^t \leftarrow \min(\delta_{i,j}^t, \delta_{j,i}^t)$ 

```

$\delta_{i,j}^t$ is initialised to δ_{gen} . Then, traffic densities of the outgoing boundary links from G_i to G_j are checked one after another against the four density thresholds. The minimum update interval is taken. After this, LP_i sends $\delta_{i,j}^t$ to LP_j and receives $\delta_{j,i}^t$ from LP_j . It must be ensured that relaxation applied should not be too much to distort the traffic condition. Thus, the smaller one of $\delta_{i,j}^t$ and $\delta_{j,i}^t$ is taken as the resultant relaxed update interval. All agents inside the buffer regions $B_{i,j}$ and $B_{j,i}$ now have the same update interval $\delta_{i,j}^t$ ($\delta_{i,j}^t = \delta_{j,i}^t$). It is evident that relaxation is limited by the boundary link that allows the least relaxation. The relaxed update intervals may be different for

the buffer regions between different LP pairs. Algorithm 5 is run by every LP for every neighbouring LP.

The relaxed update intervals need to be reevaluated when the traffic densities of links have changed as the simulation progresses. One way to do so is to periodically check the traffic densities of the links in the road network, and trigger Algorithm 5 if a significant change is detected. In this way, the suitable amount of relaxation between LP pairs can be automatically adjusted. The lookahead determination for this relaxation heuristic is given in Appendix B.2.

5.4 Experiments and Results

5.4.1 Set-up

The input data, agent models, and agent population used in the experiments are the same as those in the experiments of MA synchronisation protocol in Section 3.6. Partitioning algorithms evaluated in the parallel simulation are METIS partitioning and GG partitioning. Dynamic load-balancing is used. During repartitioning, partitioning algorithms attempt to cut the links with low traffic flow, so as to maximise relaxation. Hence, the weights of edges in the graph for partitioning are calculated using traffic flow during simulation run-time. The hardware platform is the same as well. Three compute nodes are used. LPs are evenly distributed on the compute nodes and each LP is mapped to a physical CPU core.

5.4.2 Results

Uncertainty control in output

Depending on specific applications, different amounts of relaxation may be allowed. It is necessary to control the relaxation introduced to the parallel simulation so that it does not magnify the uncertainty compared to the sequential simulation. To do that, we run the sequential simulation and relaxed parallel simulation and make sure that their outputs

are statistically equivalent.

We first need to choose output variables to measure and a statistical test. The output variables measured may be different for different studies, for example, some studies are interested in trip durations of agents, and some studies are interested in traffic flows on roads [90]. The output variable investigated in our study was the *average trip duration* of agents throughout the simulation. For statistical test of equivalence, the Kolmogorov-Smirnov two-sample test or the Bootstrap method [115] can be used. They do not make assumptions on the probability distribution of the tested variable, and thus are fit for this study. The statistical test of equivalence used was the *Bootstrap method*.

MA synchronisation protocol relaxed using skipping and estimation is denoted as *SE*, and MA synchronisation relaxed using lower temporal resolutions as *LR*. The first step is to find suitable parameter values (i.e. α for SE; and $\rho_1, \rho_2, \rho_3, \rho_4, \delta_{mod}$, and δ_{gen} for LR). The values are expected to vary with different partitioning on the network. A range of values was experimented to find out the effect. The sequential simulation was run twelve times with different random seeds, as the referential group of results. Then, the parallel simulation was run with different sets of parameter values and different number of LPs. For each set of parameter values and each number of LPs, the parallel simulation was run twelve times as one group using the same twelve random seeds as the sequential group. Statistical tests of equivalence were performed between the sequential referential group and each parallel group.

The null hypothesis of the test, H_0 , is that there is no difference between the average trip duration of agents in the sequential simulation and that in the parallel simulation. Failure to reject the hypothesis means that there is insufficient evidence to prove that the results from both simulations are different. Suppose the sequential output group is $A = \{a_1, a_2, \dots, a_{12}\}$, and a parallel output group is $B = \{b_1, b_2, \dots, b_{12}\}$. The first step of the Bootstrap method is *resampling*. In each resampling, two new sample groups $A' = \{a'_1, a'_2, \dots, a'_{12}\}$ and $B' = \{b'_1, b'_2, \dots, b'_{12}\}$ are obtained, where $a'_i \in A$, $b'_i \in B$, and $1 \leq i \leq 12$. Then, the *difference of the means* of the two resampled groups is calculated by $d = \overline{A'} - \overline{B'}$. Resampling is performed 1000 times. Thus, 1000 values of the difference

of means are obtained. The 1000 values form a distribution of the difference of means. Using a 95% confidence interval, H_0 will be rejected if zero falls outside of the percentile range between 2.5% and 97.5%. Otherwise, H_0 cannot be rejected.

There are infinite combinations of parameter values. It is impractical to traverse all possible parameter values for relaxation to look for the optimum. Thus, we used a heuristic to search for the suitable parameter values. We first select an initial value set based on the stability diagram in the literature. Then, we gradually adjust the values until statistical results are satisfactory. If the initial set is rejected, we gradually reduce the relaxation until the value set is non-rejected. The non-rejected set will be used. If the initial value set is non-rejected, we gradually increase the relaxation until the value set is rejected. The last non-rejected value set will be used. For SE, reducing relaxation can be achieved by reducing α . For LR, reducing relaxation can be achieved by increasing the nonrelaxed interval $[\rho_2, \rho_3]$ or reducing the generously relaxed intervals $[0, \rho_1]$ and $[\rho_4, \rho_{max}]$. The values evaluated for METIS and GG partitioning using SE and LR are shown in Tables 5.1 - 5.4, respectively.

Table 5.1: Statistical test of equivalence using different sensitivity factor values in SE, using METIS partitioning

8 LPs	16 LPs	32 LPs
1.0 (reject)	1.0 (reject)	1.0 (reject)
0.4 (reject)	0.4 (reject)	0.4 (reject)
0.3 (not rej)	0.3 (reject)	0.3 (reject)
	0.2 (not rej)	0.2 (reject)
		0.15 (not rej)

Table 5.1 shows the results for METIS partitioning using SE. We observed that simulation results were rejected for the initial value, hence, α was decreased until the result could not be rejected. According to Table 5.1, the suitable sensitivity factors in SE are 0.3, 0.2, and 0.15 for 8, 16, and 32 LPs, respectively.

For LR, the values of δ_{mod} and δ_{gen} are fixed to 1.0 s and 1.5 s. These are reasonable values for the agent-based models we used (recall that they represent driver's reaction time). The values of other four parameters tested for METIS partitioning are shown in Table 5.2. The suitable parameter values are boldfaced. Similar to the values in Table

Table 5.2: Statistical test of equivalence using different traffic density threshold values in LR, using METIS partitioning

Tested parameter groups (in the order of $\rho_1, \rho_2, \rho_3, \rho_4$)*, $\delta_{mod}=1.0s$, $\delta_{gen}=1.5s$		
8 LPs	16 LPs	32 LPs
50, 100, 250, 333 (reject)	50, 100, 250, 333 (reject)	50, 100, 250, 333 (reject)
50, 100, 333, 333 (reject)	50, 100, 333, 333 (reject)	50, 100, 333, 333 (reject)
25, 100, 333, 333 (not rej)	25, 100, 333, 333 (reject)	25, 100, 333, 333 (reject)
	25, 50, 333, 333 (reject)	25, 50, 333, 333 (reject)
	25, 30, 333, 333 (not rej)	25, 30, 333, 333 (reject)
		25, 25, 333, 333 (not rej)

*the unit is l/km

5.1, the first set of values were rejected. Therefore, we reduced the relaxation until they could not be rejected.

It can be observed from Tables 5.1 and 5.2 that when there are more partitions, the non-rejected values for both SE and LR tend to be smaller, which means less relaxation on a certain boundary link. This is because when there are more partitions, there are generally more boundary links. Thus, with a certain amount of relaxation on each boundary link, there will be more noise introduced to the simulation. Therefore, there should be less relaxation for a boundary link as the number of LPs increases.

The parameter values tested for SE and LR using GG partitioning are shown in Tables 5.3 and 5.4, respectively. The suitable parameter values are boldfaced as well.

Table 5.3: Statistical test of equivalence using different sensitivity factor values in SE, using GG partitioning

8 LPs	16 LPs	32 LPs
1.0 (reject)	1.0 (reject)	1.0 (reject)
0.8 (reject)	0.8 (reject)	0.8 (reject)
0.7 (not rej)	0.7 (reject)	0.7 (reject)
	0.6 (not rej)	0.6 (reject)
		0.5 (not rej)

The values tested in Table 5.3 follow a similar order to that of Table 5.1, i.e., values are decreased until simulation results could not be rejected. However, in Table 5.4, the first set of tested values were non-rejected. Hence, we gradually increased relaxation until parameter values are rejected. Increasing relaxation was achieved by shrinking the

Table 5.4: Statistical test of equivalence using different traffic density threshold values in LR, using GG partitioning

Tested parameter groups (in the order of $\rho_1, \rho_2, \rho_3, \rho_4$)*, $\delta_{mod}=1.0s$, $\delta_{gen}=1.5s$		
8 LPs	16 LPs	32 LPs
50, 100, 250, 333 (not rej)	50, 100, 250, 333 (not rej)	50, 100, 250, 333 (not rej)
100, 250, 300, 333 (not rej)	100, 250, 300, 333 (not rej)	100, 250, 300, 333 (not rej)
150, 250, 300, 333 (not rej)	150, 250, 300, 333 (not rej)	150, 250, 300, 333 (not rej)
200, 250, 300, 333 (not rej)	200, 250, 300, 333 (not rej)	200, 250, 300, 333 (not rej)
200, 250, 250, 333 (reject)	200, 250, 250, 333 (reject)	200, 250, 250, 333 (reject)

*the unit is l/km

nonrelaxed interval $[\rho_2, \rho_3]$ and expanding other relaxed intervals. The last non-rejected parameter value set, i.e., the fourth set, was chosen as the suitable value set.

As mentioned earlier, due to the infinite search space, the values obtained may not be the optimum. A more efficient and less costly algorithm for searching for the optimum values should be a future work.

We can observe from the tables above that METIS and GG have different parameters values for relaxation. This is because different partitioning algorithms generate different boundary links, which have different traffic conditions. It can be observed from Table 5.3 that the amount of relaxation applied to a boundary link generally decreases as the number of LPs increases, which is similar to the results of SE and LR using METIS partitioning. However, this trend is not observed in Table 5.4. To investigate this, we conducted extra experiments for LR using GG with the density thresholds $\rho_1=200$, $\rho_2=250$, $\rho_3=280$, and $\rho_4=333$. They apply more relaxation to the synchronisation than the fifth set of values in Table 5.4. The result is that they could not be rejected for 8 and 16 LPs, but rejected for 32 LPs. This result agrees to the claims that relaxation on a boundary link generally decreases as the number of LPs increases.

Execution time and speed-up

The execution time and speed-up of the parallel simulation using SE and LR, with GG and METIS partitioning, are shown in Figure 5.4. Parameter values for relaxation are taken from the statistical tests. For comparison purposes, the execution time of the sim-

Figure 5.4: Execution time of the parallel simulation using different synchronisation methods: (a) execution time using GG partitioning, (b) execution time using METIS partitioning, (c) speed-up using GG partitioning, and (d) speed-up using METIS partitioning.

ulations using global barrier synchronisation and MA protocol without relaxation is also shown.

From Figure 5.4, it can be observed that both SE and LR relaxation methods have reduced the execution time. Since the relaxation parameter values tested are not exhaustive, the best performance of the two relaxation methods could be better than the ones shown in the figures. Moreover, a concrete conclusion cannot be drawn on which heuristic of the two has a better lookahead. In theory, the second heuristic contains more input parameters, and thus is more flexible in fine-tuning the relaxation.

The performance gain of relaxation is the result of the increased lookahead and reduced communication overhead between LPs. The number of synchronisation messages exchanged during the simulation and lookahead are shown in Figures 5.5 and 5.6 respectively. Both relaxation heuristics have increased lookahead and reduced synchronisation

Figure 5.5: Total number of synchronisation messages sent using different synchronisation methods (smaller is better): (a) GG partitioning, and (b) METIS partitioning.

Figure 5.6: Average lookahead of relaxation methods using different synchronisation methods (larger is better): (a) GG partitioning, and (b) METIS partitioning.

messages by a large percentage.

A question may arise here is whether it is possible to simply skip synchronisation operations or reduce temporal resolutions without considering traffic flow or density. Experiments have been conducted using the same set-up of road network and agent trips as previous experiments. Relaxation is achieved either by synchronising LPs every two update intervals, or doubling the update intervals of agents in buffer regions. In both scenarios, the simulation result (i.e., average trip duration of agents) is distorted. This observation infers that considering traffic flow or density is necessary in the relaxation of synchronisation, since the same relaxed update interval has different effects under different traffic conditions.

Overhead of LR lookahead calculation

SE does not incur notable overhead, since the traffic flow can be easily tracked during the simulation. The lookahead is recalculated whenever synchronisation is performed. For LR, Algorithm 5 which determines the suitable relaxed update intervals is triggered periodically during the simulation. Reevaluation is performed every 30 seconds of simulation time in the experiments. 30 seconds is an estimation of simulation time period within which traffic densities of links may change. Thus in a 19-hour simulation, reevaluation is performed 2280 times. The overhead of running the algorithm is shown in Table 5.5.

Table 5.5: Total time spent on Algorithm 5 and the average time per invocation

		8 LPs	16 LPs	32 LPs
METIS partitioning	Total (second)	9.99±0.83	10.1±1.16	15.6±0.97
	Average (second)	0.0043	0.0044	0.0068
GG partitioning	Total (second)	6.95±0.66	5.66±0.60	11.3±0.66
	Average (second)	0.003	0.0025	0.005

The overhead of the algorithm is not significant compared to the total execution time of the simulation. Reevaluation is performed every 30 seconds, so if there is an emergency, such as vehicles breaking down, the relaxed resolution of agent state update can adapt to the situation in less than 30 seconds. The frequency of reevaluation can be adjusted according to the dynamics of traffic flow and density. If it is known prior to the simulation that the traffic flow and density will not vary much during the simulation, reevaluation can be less frequent.

From the experiments above, we can conclude that applying relaxation is able to improve the speed-up of parallel agent-based traffic simulation while keeping the simulation results statistically unchanged. Since the stability of traffic flow is different under different traffic conditions, the amount of relaxation can be customised for different traffic conditions.

5.5 Summary

Uncertainty exists throughout the process of the modelling and simulation of road traffic. There is not a unique correct result of a simulation run for studies that require statistically significant results. Given that models have deviations from reality anyway, some noise of a small scale may not affect the results statistically. This motivates the work on the relaxation of synchronisation. Thus, we proposed two heuristics for relaxing the synchronisation of parallel agent-based road traffic simulations. In both heuristics, more relaxation is applied when the traffic flow is more stable; whereas less relaxation is applied for relatively unstable traffic flow.

The first relaxation heuristic skips some synchronisation operations. It uses an estimation function to approximately compute the states of proxy agents that failed to synchronise among LPs. The mismatch between the estimated states of proxy agents and the shared states of real agents introduces noise to the simulation. A sensitivity factor is used to control how much relaxation is introduced into the simulation. The lookahead at simulation run-time is calculated with the traffic flow of boundary links and the sensitivity factor. The second relaxation heuristic reduces the temporal resolution of agent models at the boundaries of partitions so as to reduce synchronisation messages. Traffic densities on boundary links are classified into three categories according to the stability of the traffic, which allow different degrees of relaxation.

Experiments have shown that both approaches are able to improve the overall performance of the parallel simulation. As shown in experiments, suitable values of parameters used in the approaches can be determined by conducting statistical tests to ensure that simulation results are statistically unaltered by the relaxation.

Chapter 6

Reducing Synchronisation Messages with Computation Replication

6.1 Overview

Synchronisation is required when there are data dependencies that occur along boundaries of partitions, i.e., when there are migrating agents and shared states. Suppose that the data exchanged by a synchronisation operation can be computed by the LPs locally, the synchronisation operation can be skipped. This inspired another method to reduce synchronisation overhead called computation replication.

Computation replication has been an effective approach of reducing inter-process communication in parallel applications reported in the literature [98, 99, 100, 101, 102]. However, the existing solutions are not suitable for traffic simulations. More specifically, they are not designed for computing tasks that are distributed in a spatial network and for dynamic workload. There is a trade-off between the benefit of reduced synchronisation operations and redundant computation. In general, to reduce more synchronisation operations, there will be more redundant computation and more data sent in synchronisation messages. For parallel agent-based traffic simulation, it is possible that redundant computation exceeds the benefit of reduced synchronisation operations. Therefore, an

effective computation replication algorithm should be designed to manage the trade-off to achieve minimised total overhead.

In this chapter, we will describe an adaptive computation replication method to improve the speed-up of our parallel traffic simulation using MA protocol. Computation replication affects the lookaheads of LPs. Recall that when MA protocol is used, LPs predict their lookaheads by estimating when their local agents will share their states and migrate to other LPs. When computation replication is used on MA protocol, lookahead depends on how much computation is replicated. The structure of this chapter is organised as follows. Firstly, extended layers of partitions are defined to determine the computation to be replicated for a certain lookahead. The sizes of extended layers are decided by agent behaviours. Secondly, the overhead of conducting computation replication is analysed. Then, an adaptive method is presented, which adjusts the number of extended layers to be replicated at simulation run-time to maximise the benefit of computation replication. Experiments are conducted to investigate the efficiency of the adaptive computation replication method in terms of reducing synchronisation messages and speeding up the parallel simulation.

6.2 Adaptive Computation Replication

6.2.1 Extended layers

The underlying concept is to reduce the amount of synchronisation in order to speed up simulation execution. This is achieved by performing redundant computation in LPs so as to reduce the required frequency of synchronisation. Lookahead can be increased by copying more information at each synchronisation operation from neighbouring LPs. A method to determine the extra information for redundant computation is introduced below.

Extended layers for replicating agents

Interactions of agents are bounded by their locations on the road network; thus, we determine the extra information based on locations of agents on the road network. An *extended layer* of a partition is defined as the space in the road network immediately next to the boundary of the partition that is required to calculate the agent states in an LP until the next synchronisation. A partition can have multiple extended layers. The layers expand from the boundary of the partition towards neighbouring partitions. The first layer is immediately next to the boundary of the partition. The k th extended layer of partition G_i in neighbour G_j is denoted as $X_{i,j}^k$. Buffer regions are shifted next to the outermost extended layer. The buffer region of G_i inside G_j , next to $X_{i,j}^k$ is denoted as $B_{i,j}^k$.

We start with explaining using an example of one extended layer and new buffer regions shown in Figure 6.1. The network is the same as that shown in Figure 3.3(a). $X_{i,j}^1$ ($i \in \{1,2\}, j \in \{1,2\}, i \neq j$) is the extended layer of G_i inside G_j , and $B_{i,j}^1$ is the new buffer region of G_i inside G_j .

Figure 6.1: Extended layers and new buffer regions: (a) the view of network from LP_1 ; and (b) the view of network from LP_2 .

When synchronisation is conducted, each LP receives complete agents in its extended layers and shared states in buffer regions from neighbouring LPs. Agents in the extended layers still exist in the original LPs, but they are replicated in the receiving LP. They are referred to as *external agents* of the receiving LP. LPs update the states of external agents together with their local agents, so as to produce the agent states that should have been

received through synchronisation operations. The computation of the states of these external agents is *redundant computation*, as they are computed twice, once by each LP.

In the example shown in Figure 6.1, complete agents in $X_{1,2}^1$ and shared states in $B_{1,2}^1$ are sent to LP_1 by LP_2 at time t . The agents in $X_{1,2}^1$ are reconstructed and form external agents of LP_1 . States of the agents together with local agents are updated by LP_1 at time t . At time $t + \delta$, the agents in $X_{1,2}^1$ will function as proxy agents. They are still external agents and there are exact replicas of those agents in LP_2 . Note that due to movement of agents, agents currently in $X_{1,2}^1$ may be different from those at time t . Using agent states currently in $X_{1,2}^1$, agents in G_1 can be correctly updated till time $t + 2\delta$ when the next synchronisation operation is performed. The same procedure applies to LP_2 using extended layer $X_{2,1}^1$. Thus, agent states in regions G_1 and G_2 are updated until $t + 2\delta$ with only one synchronisation.

Now we generalise the idea to multiple extended layers. Supposing that LP_i and LP_j use $\hat{k}$ ($\hat{k} \geq 1$) extended layers between them, the lookahead between the two LPs will be $(\hat{k} + 1) \cdot \delta$. After a synchronisation operation is performed, during the next m th ($1 \leq m \leq \hat{k}$) update interval, the states of local agents and the agents in the external layers $\{X_{i,j}^1, X_{i,j}^2, \dots, X_{i,j}^{\hat{k}-(m-1)}\}$ are updated. After the m th update interval, agents in the extended layer $X_{i,j}^{\hat{k}-(m-1)}$ function as proxy agents. In the $(\hat{k} + 1)$ th update interval, LP_i only updates the states of its local agents. After that, another synchronisation is required. A similar procedure is conducted by LP_j . Hence, agent states that LP_i and LP_j are responsible for are updated correctly.

There are two extreme cases for the number of extended layers replicated between LPs. One case is when there is no replication. Synchronisation is performed every update interval. Another case is when there are so many extended layers that each partition encompasses the whole simulation space. In this case, all LPs execute the whole simulation and no synchronisation is required. However, the speed-up of the parallel simulation compared to the sequential simulation will be less than one due to overheads.

Requirements for agent models

Since the computation of external agents is replicated in LPs, agent models should satisfy the following requirement: given the same input values, models of agents always produce the same agent states, including agent-based and component-based state variables, and the order of the agents being updated should not affect the result. Otherwise, replicas may generate different states. This requirement is met for deterministic models. For stochastic models, this can be achieved by manipulating the seeds of the random number generators in the two replicating LPs to produce the same random number sequence.

6.2.2 Sizes of extended layers

Using computation replication to reduce frequent synchronisation, first of all, a method to determine locations and sizes of extended layers is required.

Representation of extended layers

As described previously, a road network is represented as a collection of links and nodes, and agents are positioned on links. Sensing ranges of agents usually do not cover entire links, thus an extended layer may only cover a certain part of a link. Therefore, we represent an extended layer as a collection of link segments. A *link segment* is a portion of a link between two points on the link. A link segment can be uniquely identified by specifying the link id that the segment is on and displacements of the two points. (Recall that a displacement of a point on a link is the distance between that point and the start node of the link.) Thus, a link segment can be denoted as $seg_{(id,s,e)}$, where id is the id of the link, s is the starting displacement of the segment on the link, e is the ending displacement. The size of this segment is $|e - s|$.

Size of link segments

To determine the sizes of extended layers, we need to analyse the behaviour of agents and the topology of the road network, since front and back sensing ranges of agents may

be different and links of the road network can have different lengths and connectivity too.

The agents in the k th extended layer should allow the agents in the $(k-1)$ th extended layer (or local agents if $k=1$) to advance one update interval. Therefore, the k th extended layer must cover the sensing ranges of the agents in the $(k-1)$ th layer (or those in the local area if $k=1$). Meanwhile, due to the movement of agents, the layer must also consider the agents that may move into it. The sizes of segments in extended layers are illustrated in Figure 6.2.

Figure 6.2: Different sizes of link segments in extended layers of G_i depending on the directions of extension. The solid arrows indicate the traffic directions on links. The empty arrows indicate the directions of extension of extended layers. The direction of extension of $X_{i,j}^k$ is opposite to the traffic direction of link l , i.e., extending upstream; for $X_{i,q}^k$, the direction of extension is the same as the traffic direction of link m , i.e., extending downstream.

In Figure 6.2, it is assumed that the links l and m are long enough to cover $X_{i,j}^k$ and $X_{i,q}^k$ respectively. For extended layers, the direction of extension is from the $(k-1)$ th layer to the k th layer (or from the local area to the first layer if $k=1$). The direction of extension can be the same or opposite to the traffic direction of a link. For $X_{i,j}^{k-1}$ shown in Figure 6.2, the direction of extension is opposite to the traffic direction. $X_{i,j}^k$ is in the upstream direction of $X_{i,j}^{k-1}$. The segment of $X_{i,j}^k$ consists of two parts: the first part of size γ_b and the second part of size $\hat{v}_l \cdot \delta$, where $\hat{v}_l$ is the speed limit of link l . The second part, which is marked with a wavy pattern, ensures that after agents in $X_{i,j}^k$ are updated, the first part of $X_{i,j}^k$ contains all the agents required for the next round of state update for $X_{i,j}^{k-1}$. Its size is the maximum distance that agents can travel on link l in one update interval. In contrast, for $X_{i,q}^{k-1}$, the direction of extension is the same as the traffic direction. $X_{i,q}^k$ is in the downstream direction of $X_{i,q}^{k-1}$. The segment of $X_{i,q}^k$ only requires a size of γ_f , because after an update, $X_{i,q}^k$ will still contain all the agents needed to update $X_{i,q}^{k-1}$ for the next update interval.

An example with two extended layers is depicted in Figure 6.3. A synchronisation operation is performed at time t . LP_i receives complete agents in the two extended layers and shared states in the buffer regions from LP_j and LP_q . As shown in Figure 6.3(a), states of the agents in $X_{i,j}^2$, $X_{i,j}^1$, G_i , $X_{i,q}^1$, and $X_{i,q}^2$ (i.e., the dark grey region) are updated by LP_i . Figure 6.3(b) shows that at time $t + \delta$, agents in $X_{i,q}^2$ and the first part of $X_{i,j}^2$ become new proxy agents. The second part of $X_{i,j}^2$ (marked with a cross) is dropped since it is no longer needed. Agents in $X_{i,j}^1$, G_i , and $X_{i,q}^1$ are updated. As the simulation progresses to time $t + 2\delta$ as shown in Figure 6.3(c), agents in $X_{i,q}^1$ and the first part of $X_{i,j}^1$ become new proxy agents. Similar to the previous update interval, the region marked with a cross in $X_{i,j}^1$ in Figure 6.3(c) is dropped. Agents in G_i will be updated. In this way, we can update agents in G_i for three update intervals without synchronisation.

Figure 6.3: An example of LP_i progressing through time using two extended layers: (a) At time t , there are two extended layers for G_i in G_j and G_q . (b) At time $t + \delta$, $B_{i,j}^2$ and $B_{i,q}^2$ are removed and agents in $X_{i,j}^2$ and $X_{i,q}^2$, excluding the crossed region, become new proxy agents. (c) At time $t + 2\delta$, $X_{i,j}^2$ and $X_{i,q}^2$ are removed and agents in $X_{i,j}^1$ and $X_{i,q}^1$, excluding the crossed region, become new proxy agents.

Segments on connecting links

In the cases described above, it is assumed that link l and link m are long enough to cover extended layers. When a link is not long enough for extended layers, the layers will expand to its connecting links. Link segments on the connecting links may have various sizes depending on how they are connected to the link. There are four cases of how a link x can connect to another link y : i) they share the same start node; ii) they share the same end node; iii) the start node of link x is the end node of the link y ; and iv) the end node of link x is the start node of link y . The segment sizes for the four cases are

Figure 6.4: Four basic cases of extended layer segments on connecting links, illustrated by links a , b , c , and d . (a) At time t , the k th extended layers $X_{i,j}^k$ and $X_{i,q}^k$ have extra segments on link a and link c respectively (marked by a wavy pattern). (b) At time $t + \delta$, the first parts of $X_{i,j}^k$ and $X_{i,q}^k$ become buffer regions.

illustrated in Figure 6.4.

In Figure 6.4, link a has the start node of link l as its end node. Link b and link l share the same start node. Link c and link m share the same end node. Link d has the end node of link m as its start node. Figure 6.4(a) shows k extended layers and buffer regions at simulation time t . $X_{i,j}^k$ contains an extra segment on link a , and $X_{i,q}^k$ contains an extra segment on link c . Similar to the reasons explained above in Figure 6.2, the extra segments are required so that the buffer regions for the next round of state update will contain all necessary proxy agents. Figure 6.4(b) shows the extended layers and buffer regions when the simulation progresses to time $t + \delta$.

A real-world road network usually has a large number of links and very complex connectivity. The four cases described above can be arbitrarily combined. Depending on the topology of the road network and sizes of partitions, one extended layer can contain many segments. A searching algorithm is designed to determine the segments of extended layers in the next subsection.

Searching algorithm for determining extended layers

The extended layers of G_i inside G_j is calculated by LP_j . A search is performed by LP_j starting from all the boundary links between G_i and G_j towards the inner area of G_j . The

Algorithm 6: Searching for link segments of extended layers of G_i inside G_j by LP_j

```
1 Definitions:
2  $k$       index of the extended layer  $X_{i,j}^k$ 
3  $I_{j,i}$    set of incoming boundary links of  $G_j$  from  $G_i$ 
4  $O_{j,i}$    set of outgoing boundary links of  $G_j$  towards  $G_i$ 
5  $S_{add}$    candidate segments to be added to an extended layer
6  $S_{cont}$    segments in  $S_{add}$  from which the next layer continues,  $S_{cont} \subseteq S_{add}$ 
7  $S_{check}$   record of the segments that are already inside extended layers
8  $length_l$  length of link  $l$ 
9  $\hat{v}_l$      speed-limit of the traffic on link  $l$ 
10  $\zeta$       boolean - a flag that indicates if the searching should continue

11 initialize  $k \leftarrow 1$ ,  $\zeta \leftarrow true$ ;
12 foreach link  $l \in I_{j,i}$  do
13   | put segment  $seg_{(l, \frac{length_l}{2}, \frac{length_l}{2} + \gamma_f)}$  into  $S_{add}$ ,  $S_{cont}$  and  $S_{check}$ ;
14 end
15 foreach link  $l \in O_{j,i}$  do
16   | put segment  $seg_{(l, \frac{length_l}{2}, \frac{length_l}{2} - \gamma_b - \hat{v}_l \cdot \delta)}$  into  $S_{add}$ ,  $S_{cont}$  and  $S_{check}$ ;
17 end
    // search layer by layer
18 while  $\zeta$  do
19   | foreach  $seg_{(id,s,e)} \in S_{add}$  do
20     | put  $seg_{(id,s,e)}$  into layer  $X_{i,j}^k$ ;
21   | end
22   |  $S_{temp\_cont} \leftarrow S_{cont}$ ;
23   |  $S_{add} \leftarrow \emptyset$ ,  $S_{cont} \leftarrow \emptyset$ ;
    // search segments for the next layer
24   | foreach  $seg_{(id,s,e)} \in S_{temp\_cont}$  do
25     | if  $\zeta \wedge (s < e)$  then
26       | |  $searchDownstream(id, e, \gamma_f)$ ; // direction of extension is downstream
27     | else if  $\zeta \wedge (e < s)$  then
28       | |  $searchUpstream(id, e, \gamma_b + \hat{v}_{id} \cdot \delta)$ ; // direction of extension is upstream
29     | end
30   | end
31   | if  $S_{add} = \emptyset$  then
32     | |  $\zeta \leftarrow false$ ; // the whole  $G_j$  have been searched
33   | end
34   |  $k \leftarrow k + 1$ ;
35 end
```

searching algorithm for extended layers is shown in Algorithm 6.

The algorithm starts with initializing the layer index and a flag that controls the termination of the algorithm. A set S_{add} is used to store the segments in the current layer. A set S_{cont} is used to store the segments in S_{add} from which the next layer continues. Note

that some segments in S_{add} may not connect to the next layer (a detailed description is provided in Appendix D). A set S_{check} is used to keep track of all the segments that have been added to the extended layers. Areas occupied by the segments in S_{check} will not be searched again.

The first extended layer is obtained by adding segments from boundary links (lines 12 - 17). It is assumed that boundary links are long enough to accommodate the segments of the first layer. Subsequently, the algorithm continues to search for segments of the next layer based on the current layer (*while* loop in lines 18 - 35). Each iteration of the *while* loop determines one layer of extended layers. In each iteration, segments in S_{add} are added to the current layer. Segments in S_{cont} are added to a temporary set S_{temp_cont} . Then, S_{add} and S_{cont} are emptied and reused to store new segments for the next layer. After that, the algorithm searches for segments in the next layer connecting to the segments in S_{temp_cont} . If $s < e$, the search continues towards the downstream direction of the link, using function *searchDownstream*. If $e < s$, the search continues towards the upstream direction of the link, using function *searchUpstream*. The detail of functions *searchDownstream* and *searchUpstream* is explained in Algorithms 9 and 10 in Appendix D respectively. In functions *searchDownstream* and *searchUpstream*, when the remaining space on a link is not enough for a layer, the searching will expand to the connecting links.

The searching terminates if the whole area of G_j has been searched (Algorithm 6 line 32) or the extension reaches a third partition (Algorithm 9 line 14 and Algorithm 10 line 14). In other words, extended layers of G_i inside G_j is restricted within partition G_j only. Otherwise, if they were extended to a third partition G_q , LP_q would also need to communicate with LP_i during the synchronisation between LP_i and LP_j . This will introduce extra communication which may downgrade the benefit of computation replication.

Algorithm 6 is executed after the partitions of LPs are determined. If partitions of LPs are changed dynamically during the simulation run-time, the extended layers need to be recalculated.

6.2.3 Adaptive extended layers

A partition may have different numbers of extended layers available inside different neighbouring partitions. It is decided by the topology of the road network and the partitioning. However, to reduce the total execution time of the simulation, it may not be beneficial to replicate all available extended layers. If the density of agents in the extended layers is high, the computation overhead introduced by redundant computation may exceed the benefit of reduced synchronisation operations. There should be an optimum number of extended layers to replicate at a given simulation time. Since traffic density and flow on the roads often change throughout the simulation, the optimum number of extended layers to replicate may change as well. It may not be practical to always adapt to the optimum whenever the traffic condition changes. Hence, to minimize overall simulation execution time, in this section, we propose a heuristic that periodically adjust the number of extended layers for computation replication. The symbols used in this section are listed in Table 6.1.

Analysis of overhead

We first formulate the total overhead incurred between a pair of LPs when a certain number of extended layers is replicated. Discussion here assumes that MA protocol is used for synchronisation between LPs. Therefore, the number of extended layers replicated for G_i inside G_j and for G_j inside G_i will always be equal.

Considering a pair of neighbouring partitions G_i and G_j , there are $K_{i,j}$ available extended layers between the two. Let a unit of computation be the wall-clock time required for processing the events of an agent during one update interval. If k ($1 \leq k \leq K_{i,j}$) layers are replicated, the total units of redundant computation in LP_i between two consecutive synchronisation operations is:

$$\sum_{x=1}^k \left(|A_{i,j}^x| \cdot (k+1-x) \right) \quad (6.1)$$

where $|A_{i,j}^x|$ is the number of agents in layer $X_{i,j}^x$. It can be estimated using the sizes of

Table 6.1: Symbols used in the analysis

Symbol	Description
$A_{i,j}^k$	set of agents in extended layer $X_{i,j}^k$, $k \geq 1$
$K_{i,j}$	available extended layers between LP_i and LP_j
$K_{i,j}^m$	adaptive range for LP_i and LP_j at m th evaluation of optimum k
$ M_{i,j} $	number of migrating agents from LP_i to LP_j in one update interval
$ S_{i,j}^k $	number of shared states that LP_i sends to LP_j (i.e., in $B_{j,i}^k$) in one message
Ta	computational workload of one agent in one update interval
b	available bandwidth of message-passing
c	latency at the sending and receiving ends for one message-passing
$\hat{k}_{i,j}$	obtainable optimum number of extended layers to replicate between LP_i and LP_j ($\hat{k}_{i,j} = \hat{k}_{j,i}$ since MA protocol is used)
$\hat{k}_{i,j}^m$	$\hat{k}_{i,j}$ for the m th evaluation period
$o_{i,j}(k)$	total overhead for LP_i , relative to LP_j in one synchronisation cycle, when k layers are replicated, including redundant computation and message-passing overhead
$o_{i,j}(\tau, k)$	total overhead for LP_i , relative to LP_j during time period τ when k layers are replicated
$s_{i,j}(k)$	time required by LP_i to receive a synchronisation message from LP_j in a synchronisation operation when k layers are replicated
$\Gamma_{i,j}(k)$	redundant computation between two consecutive synchronisation operations in LP_i , when k extended layers are replicated

the segments in the extended layer and agent densities on the segments.

Therefore, the redundant computation in terms of wall-clock time in LP_i , relative to LP_j between two consecutive synchronisation operations is:

$$\Gamma_{i,j}(k) = Ta \cdot \sum_{x=1}^k \left(|A_{i,j}^x| \cdot (k+1-x) \right) \quad (6.2)$$

where Ta is a unit of computation, i.e., the wall-clock time for computing the events of an agent in one update interval. From Equation 6.2, we can derive the difference quotient

between $\Gamma_{i,j}(k+1)$ and $\Gamma_{i,j}(k)$ as:

$$\Delta\Gamma_{i,j}(k+1,k) = Ta \cdot \sum_{x=1}^{k+1} |A_{i,j}^x| \quad (6.3)$$

It is known that $|A_{i,j}^x| \geq 0$; therefore, $\Delta\Gamma_{i,j}(k+1,k) \geq 0$. This equation shows that if extended layers are not empty (i.e., $|A_{i,j}^x| > 0$), the redundant computation increases with an increasing k and the increase becomes faster as k increases.

The total overhead for LP_i , relative to LP_j , in one synchronisation cycle using k extended layers is:

$$o_{i,j}(k) = \Gamma_{i,j}(k) + s_{i,j}(k) \quad (6.4)$$

where $s_{i,j}(k)$ is the time spent by LP_i on waiting for the synchronisation messages from LP_j to arrive, when k extended layers are replicated. LP_i can only resume computing agent states after the message from LP_j has arrived. LP_j will do the same calculation on $o_{j,i}(k)$; then, the final total overhead for LP_i and LP_j is the maximum of the two, that is,

$$\max(o_{i,j}(k), o_{j,i}(k)) \quad (6.5)$$

Now, we represent overhead $o_{i,j}(k)$ using parameters that can be measured in the simulation. Assume that the time for message-passing can be approximated using the following model:

$$time = \frac{data_volume}{bandwidth} + send_recv_latency \quad (6.6)$$

where the first item is the transmission delay and the second item is some latency at the sending and receiving ends. Note that bandwidth and latency may not be constants, i.e., they may vary with different message sizes. As indicated in the literature (i.e., [116, 117]), it is not straightforward or even impossible to define an accurate model for message-passing time that can be universally applied. Hence, this model needs to be adapted to specific applications.

Based on Equation 6.6, $s_{i,j}(k)$ can be defined as:

$$s_{i,j}(k) = \frac{SizeA}{b} \cdot \left(\sum_{x=1}^k |A_{i,j}^x| + |M_{j,i}| \right) + \frac{SizeS}{b} \cdot |S_{j,i}^k| + c \quad (6.7)$$

where $SizeA$ and $SizeS$ are the data sizes of a complete agent and the shared state of an agent respectively, $M_{j,i}$ is the set of migrating agents from LP_j to LP_i at the time of synchronisation, $S_{j,i}^k$ is the set of shared states that LP_j sends to LP_i during the synchronisation, b is the available bandwidth for message-passing, and c is a latency at message sending and receiving ends. This equation shows that as the number of replicated extended layers increases, the overhead of passing one synchronisation message increases as well.

In order to compare the overhead of synchronisation using different numbers of extended layers, we define the total overhead for LP_i , relative to LP_j , during a simulation period τ , $\tau \gg \delta$, with k extended layers, as follows:

$$o_{i,j}(\tau, k) = \frac{\tau}{(k+1) \cdot \delta} \cdot o_{i,j}(k) \quad (6.8)$$

Equation 6.8 shows that as the number of replicated extended layers (i.e., k) increases, the number of synchronisation operations during period τ (i.e., the first term of the right-hand side of Equation 6.8) decreases; however the total overhead per synchronisation cycle (i.e., the second term of the right-hand side of Equation 6.8) increases. The optimum k gives the best trade-off between the benefit of reducing the synchronisation frequency and the total overhead caused by redundant computation.

It can be noticed from Equations 6.2 and 6.7 that the total overhead is influenced by many factors, including the numbers of agents in the extended layers, time to compute agent states for one update interval, data sizes of complete agents and shared states, and message-passing bandwidth and latency at sending and receiving ends. Obviously, $o_{i,j}(\tau, k)$ is not a simple convex or concave function, thus the optimum k cannot be determined mathematically using Equation 6.8. So, in the next subsection we describe a heuristic algorithm to empirically determine the optimum k (referred to as obtainable

optimum k and denoted as $\hat{k}_{i,j}$).

Dynamic determination of the obtainable optimum number of replicated extended layers

The previous subsection has formulated the total overhead. To empirically determine $\hat{k}_{i,j}$, $o_{i,j}(\tau, k)$ and $o_{j,i}(\tau, k)$ need to be evaluated for each k for the evaluation period τ . To do so, the information on the distribution of agents on the road network needs to be obtained at simulation run-time. If traffic conditions change, a reevaluation of $\hat{k}_{i,j}$ may be required. In cases when there are a large number of available extended layers, the overhead of determining $\hat{k}_{i,j}$ may become significant. So, this section introduces a strategy to reduce the overhead of determining $\hat{k}_{i,j}$.

Suppose that the simulation time from the start to the end can be segmented into M ($M \geq 1$) periods, and in each period the traffic flow and density of links have no or only marginal changes. Let us denote m th ($1 \leq m \leq M$) period as τ_m . Reevaluation of $\hat{k}_{i,j}$ is only required at the beginning of each period. So, lookahead remains constant during the period. Obviously, the smaller the period is, the more accurately $\hat{k}_{i,j}$ can be determined, but the more frequent reevaluation will be required. The size of a period (i.e., τ) should be much larger than one update interval (i.e., δ).

To reduce the overhead of reevaluating $\hat{k}_{i,j}$, the following method is proposed to limit the range of k . In consecutive periods τ_{m-1} and τ_m , the obtainable optimum lookahead may not vary much, thus it is very likely that the obtainable optimum lookahead in τ_m has a similar value to that of τ_{m-1} . Thus, the lookahead in period τ_{m-1} can be utilized to limit the range of k to be evaluated in period τ_m . An adaptive range in time period τ_m , denoted as $K_{i,j}^m$, determines the number of extended layers between LP_i and LP_j that need to be evaluated:

$$K_{i,j}^m = \begin{cases} K_{i,j}, & \text{if } m = 1 \\ \min(K_{i,j}, 2 \cdot \hat{k}_{i,j}^{m-1} + 1), & \text{otherwise} \end{cases}$$

where $\hat{k}_{i,j}^{m-1}$ is the obtainable optimum k value between LP_i and LP_j in period τ_{m-1} . The

adaptive range limits the search range in period τ_m , such that the resultant lookahead is at most twice of that in τ_{m-1} . Meanwhile, it ensures that the lookahead is able to grow exponentially if necessary. In the first period (i.e., τ_1), the entire range of $[0, K_{i,j}]$ is searched. In period τ_m ($m > 1$), the search range of k is $[0, \min(K_{i,j}, 2 \cdot \hat{k}_{i,j}^{m-1} + 1)]$. $k = 0$ here refers to the case where there are no extended layers. When $k = 0$, $o_{i,j}(0) = s_{i,j}(0)$. Based on Equation 6.7, $s_{i,j}(0)$ can be derived as follows:

$$s_{i,j}(0) = \frac{SizeA}{b} \cdot |M_{j,i}| + \frac{SizeS}{b} \cdot |S_{j,i}| + C \quad (6.9)$$

where $|S_{j,i}|$ is the size of the set of shared states LP_j sends to LP_i during synchronisation when there are no extended layers.

The search algorithm for $\hat{k}_{i,j}^m$ is shown in Algorithm 7. Since MA synchronisation protocol is used, $\hat{k}_{i,j}^m = \hat{k}_{j,i}^m$.

Algorithm 7: Search algorithm for $\hat{k}_{i,j}^m$, by LP_i

```

1 Definitions:
2  $K_{i,j}$     maximum available number of extended layers between  $G_i$  and  $G_j$ 
3  $K_{i,j}^m$    adaptive range of available extended layers between  $LP_i$  and  $LP_j$  in time period  $\tau_m$ 
4  $o_{min}$     minimum total overhead in period  $\tau_m$ , considering both  $LP_i$  and  $LP_j$ 
   Result:  $\hat{k}_{i,j}^m$ 
5 if  $m = 1$  then
6   |  $K_{i,j}^m \leftarrow K_{i,j}$ ;
7 else
8   |  $K_{i,j}^m \leftarrow \min(K_{i,j}, 2 \cdot \hat{k}_{i,j}^{m-1} + 1)$ ;
9 end
10  $k \leftarrow 0$ ,  $o_{min} \leftarrow \max(o_{i,j}(\tau_m, 0), o_{j,i}(\tau_m, 0))$ ,  $\hat{k}_{i,j}^m \leftarrow 0$ ;
11 while  $k < K_{i,j}^m$  do
12   |  $k \leftarrow k + 1$ ;
13   | if  $o_{min} > \max(o_{i,j}(\tau_m, k), o_{j,i}(\tau_m, k))$  then
14     |  $o_{min} \leftarrow \max(o_{i,j}(\tau_m, k), o_{j,i}(\tau_m, k))$ ;
15     |  $\hat{k}_{i,j}^m \leftarrow k$ ;
16   | end
17 end

```

The adaptive range is first determined (lines 5 - 9). Hence, for each value k in the adaptive range, the overhead is determined by the higher value between $o_{i,j}(\tau_m, k)$ and

$o_{j,i}(\tau_m, k)$. The algorithm returns $\hat{k}_{i,j}^m$ that results in the minimum overhead for all the values in the adaptive range (lines 10 - 17).

Using computation replication, synchronisation events using MA protocol shown in Algorithm 2 need to be slightly altered. The first change is that instead of determining lookahead in synchronisation operations, the lookahead is determined at the beginning of each evaluation period (by determining $\hat{k}_{i,j}$) and is kept fixed during the period. The second change is that external agents are included in the synchronisation messages.

The effectiveness of the proposed method of dynamically determining the number of extended layers to be replicated has been investigated in experiments. The results will be presented in the next section.

6.3 Experiments and Results

6.3.1 Set-up

The input data, agent models, and agent population used in the experiments are the same as those in the experiments in Section 3.6. The parallel simulation is partitioned using METIS partitioning and GG partitioning. Dynamic load-balancing is used with the same settings as that in Section 3.6 as well. Algorithm 6 is executed whenever a load-balancing operation is performed due to the change of partitions. The hardware platform is the same as well. Three compute nodes are used. LPs are evenly distributed on the compute nodes and each LP is mapped to a physical CPU core.

6.3.2 Results

Lookahead

We investigated whether the average lookahead of the simulation using MA protocol is increased. The interval of reevaluating the obtainable optimum numbers of replicated layers is 10 minutes of simulation time, which is the same as the interval of examining workload imbalance. (Different evaluation periods are also evaluated, which will be pre-

sented at the end of this result section.) Let us denote MA synchronisation protocol with adaptive computational replication as *CR*. The average lookahead and synchronisation messages of MA and MA-CR, using METIS and GG partitioning, are shown in Figure 6.5.

Figure 6.5: Average lookahead and total number of synchronisation messages in MA and MA-CR synchronisation: (a) average lookahead using GG partitioning, (b) average lookahead using METIS partitioning, (c) total messages using GG partitioning, and (d) total messages using METIS partitioning.

It can be observed from Figures 6.5(a) and (b) that the lookahead values of MA-CR are significantly larger than those of MA. For example, there is approximately a threefold increase for the case of METIS partitioning with 32 LPs (see Figure 6.5(b)). Correspondingly, Figures 6.5(c) and (d) show that compared to MA, there are much fewer synchronisation messages when MA-CR is used. The improvement on lookahead is more obvious for METIS partitioning than GG partitioning. This can be explained by the smaller number of boundary links generated by METIS partitioning compared to GG partitioning. Less boundary links generally contain less agents, which result in more

extended layers being replicated.

Overall execution time and speed-up

Comparisons of the overall execution time and speed-up of the parallel simulation with and without computation replication are shown in Figure 6.6.

Figure 6.6: Execution time and speed-up of the parallel simulation using barrier, MA, and MA-CR synchronisation: (a) execution time using GG partitioning, (b) execution time using METIS partitioning, (c) speed-up using GG partitioning, and (d) speed-up using METIS partitioning.

The MA-CR method has the shortest execution time for both partitioning methods and for all different numbers of LPs. This shows that using adaptive computation replication has improved the overall performance of the simulation. The improvement is the result of the reduced synchronisation messages, as shown in Figure 6.5. This indicates that the benefit of reduced synchronisation messages is greater than the overhead of redundant computation and increased message sizes.

When the number of LPs increases from 16 to 32, the overall simulation execution time using MA-CR decreases further; whereas the simulation execution time using MA increases. This is because the absolute number of messages using MA-CR does not increase so drastically from 16 LPs to 32 LPs. Meanwhile for MA, the increase of the absolute number of messages from 16 LPs to 32 LPs is much more than the increase for MA-CR (see Figure 6.5).

Redundant computation and overhead of the adaptive algorithms

The amount of redundant computation and the overhead of the MA-CR method throughout the simulation are investigated in this subsection. The average redundant computation per LP is shown in Table 6.2.

Table 6.2: Average redundant computation per LP (unit=second)

	8 LPs	16 LPs	32 LPs
GG partitioning	9.83 ± 0.41	11.1 ± 0.32	14.5 ± 0.21
METIS partitioning	14.7 ± 0.66	16.2 ± 0.52	18.8 ± 0.42

It can be observed that the redundant computation is less than 20 seconds for all cases, which is less than 1.6 % of the total simulation execution time. It generally increases when the number of LPs increases. The increase is due to the increased total replication areas and the increased lookahead when there are more partitions.

The overhead of the adaptive algorithms includes calculating extended layers (Algorithm 6), and adaptively evaluating the optimum lookahead (Algorithm 7). Algorithm 6 was executed whenever a load-balancing operation was performed. The number of load-balancing operations is the same as those shown in Table 4.1. Algorithm 7 was executed 114 times (i.e., every 10 minutes in the 19-hour simulation time). Both algorithms are executed by every LP; hence, the maximum overhead amongst the LPs is collected. Total overheads of the algorithms throughout the simulation are shown in Tables 6.3 and 6.4 respectively.

Results in Tables 6.3 and 6.4 have shown that the overheads of the algorithms are at the magnitude of seconds, which is insignificant with respect to the overall simulation

Table 6.3: Total overhead of calculating extended layers (Algorithm 6) (unit=second)

	8 LPs	16 LPs	32 LPs
GG partitioning	6.20 ± 0.68	2.98 ± 0.28	0.86 ± 0.19
METIS partitioning	6.08 ± 1.85	1.61 ± 0.46	0.58 ± 0.09

Table 6.4: Total overhead of evaluating the optimum lookahead (Algorithm 7) (unit=second)

	8 LPs	16 LPs	32 LPs
GG partitioning	2.21 ± 0.10	1.31 ± 0.06	0.55 ± 0.02
METIS partitioning	2.64 ± 0.11	1.16 ± 0.13	0.79 ± 0.07

execution time. The overhead of the algorithms generally decreases as the number of LPs increases. As the average area of partitions decreases, it requires less computation for determining extended layers and the obtainable optimum lookahead.

Frequency of lookahead reevaluation

We have also analysed the effect of different evaluation periods on the average lookahead, the overall simulation execution time, and the overhead of adapting the optimum number of extended layers (i.e., Algorithm 7). The results using GG and METIS partitioning and evaluation periods of 1 minute, 5 minutes, and 10 minutes, with 32 LPs, are shown in Tables 6.5 and 6.6.

Table 6.5: Comparison of average lookahead, simulation execution time, and overhead of evaluating optimum lookahead using different evaluation periods with 32 LPs, using GG partitioning

reevaluation period	average lookahead (update intervals)	execution time (second)	overhead (second)
1 min	2.54 ± 0.02	1210.5 ± 40.6	32.96 ± 0.42
5 min	2.57 ± 0.04	1216.0 ± 13.8	4.30 ± 0.33
10 min	2.51 ± 0.03	1185.1 ± 43.0	0.55 ± 0.02

There is no significant difference in the average lookahead using different evaluation periods (approximately 1% difference between them). The overall simulation execution time is not improved by increasing the evaluation frequency. This is because the traffic condition does not change so often. Therefore, frequent reevaluation of the optimum number of replicated extended layers is not necessary. At the same time, the overhead

Table 6.6: Comparison of average lookahead, simulation execution time, and overhead of evaluating optimum lookahead using different evaluation periods with 32 LPs, using METIS partitioning

reevaluation period	average lookahead (update intervals)	execution time (second)	overhead (second)
1 min	4.50 ± 0.15	1275.6 ± 36.8	20.91 ± 1.34
5 min	4.51 ± 0.09	1256.8 ± 23.6	2.98 ± 0.30
10 min	4.56 ± 0.05	1253.8 ± 36.6	0.79 ± 0.07

of the algorithm for calculating the optimum lookahead increases when the frequency of reevaluation increases, which also results in a slight increase of the total execution time.

Effect of adaptive range

We compared the lookahead values obtained with and without adaptive range. The result is shown in Figure 6.7.

Figure 6.7: Average lookahead of all LPs throughout the simulation with and without adaptive range.

The figure shows that the lookahead values between with and without adaptive range are similar. There is no significant difference between the two sets of values. Hence, the adaptive range does not affect the lookahead significantly.

The overhead of evaluating the optimal lookahead with adaptive range is also compared to that without adaptive range. When there are 32 LPs, the overhead *without* adaptive range is similar to that with adaptive range which is shown in Tables 6.5 and 6.6. This is because the average area of partitions is relatively small compared to those of 8

and 16 LPs. Adaptive range cannot reduce the search range much. However, when there are 8 LPs and the evaluation period is 1 minute, the overheads with and without adaptive range for GG partitioning are 19.1 ± 1.3 and 41.4 ± 3.5 seconds, respectively; and those for METIS partitioning are 18.1 ± 0.9 , and 43.6 ± 6.6 seconds, respectively. Hence, adaptive range has reduced the overheads by 53.9 % and 58.5 % for GG and METIS partitioning, respectively. Therefore, adaptive range is more beneficial for a smaller number of LPs (since each partition has a larger area) and more frequent evaluations.

From our experiments above, we studied that computation replication can reduce the overall execution time of parallel agent-based road traffic simulation by reducing the number of synchronisation messages. When there are less agents at the boundaries of partitions, computation replication is more beneficial.

Comparing computation replication to the relaxation heuristics proposed in the previous chapter, they are both able to reduce simulation execution time. However, they each have their own advantages. The advantage of using computation replication compared to relaxing synchronisation is that it does not change simulation results at all. However, the disadvantage is that it requires the replicated computation in different LPs to produce identical agent states. For instance, the seeds of random number generation need to be manipulated for stochastic models in order to use computation replication. Another difference between the two methods is that the obtainable optimum number of extended layers in computation replication depends on hardware configurations and computation time of agent models. However, relaxation is not affected by these two factors.

6.4 Summary

Computation replication has been applied mainly in applications such as matrix multiplication and solving partial differential equations to reduce inter-process communication. However, there is little work done to apply this approach to parallel agent-based road traffic simulation.

In this chapter, we mainly focused on two problems for effectively applying the concept of computation replication to agent-based traffic simulation: i) how to determine

the redundant computation required to achieve a certain synchronisation frequency, and ii) how to manage the trade-off between the benefit of reduced synchronisation message and the overhead resulting from redundant computation and the increased message size. The first problem was solved by analysing the characteristics of agents and road networks to determine extended layers of partitions in a road network. Moving directions and sensing ranges of agents were considered in the determination. The second problem was solved by developing an analytical model of the total overhead of redundant computation and synchronisation, and using it to dynamically adjust the number of extended layers to replicate. To reduce the overhead of determining the optimum number of replicated extended layers when traffic condition changes, an adaptive approach was used to limit the search range.

Experiments have shown that the adaptive computation replication method is able to reduce synchronisation messages significantly by increasing lookahead of LPs. The overall execution time of the parallel simulation has been significantly reduced, though there is redundant computation.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Long execution time is still one of the impediments to the use of large-scale microscopic and nanoscopic traffic simulations in transportation studies. This study was set out to reduce the execution time of large-scale agent-based traffic simulations by utilising parallel computing techniques. Algorithms are developed in this work to accelerate parallel microscopic and nanoscopic traffic simulations. In order to make use of parallel computing techniques for traffic simulation, it is important to reduce the two prominent overheads: load imbalance and synchronisation overhead. Therefore, this study sought to reduce load imbalance and synchronisation overhead by taking advantage of characteristics of the models in traffic simulation. In particular, this study consists of four parts, which are summarised as follows.

Firstly, this research analysed the effectiveness of the existing load-balancing and synchronisation methodologies in parallel agent-based traffic simulation. Static partitioning and dynamic load-balancing using the existing stripe spatial partitioning and METIS graph partitioning are evaluated. In partitioning, the workload on the road network and possible data dependencies are represented as a weighted partitioning graph. Then partitioning algorithms can operate on the weighted graph using the geographical information and connectivity information. For dynamic load-balancing, the workload of LPs is

periodically checked during the simulation. If the imbalance exceeds a threshold, a load-balancing operation is triggered by a master LP. The weights of vertices and edges of the partitioning graph are recomputed. The road network is repartitioned from scratch and newly generated partitions are mapped to the previous partitions. Then, agents are re-distributed to new partitions. For synchronisation protocols, the effectiveness of global barrier and MA protocol is investigated. Using MA protocol, neighbouring LPs form synchronisation pairs and communicate through point-to-point message-passing. There are no global barriers required and thus LPs can progress asynchronously. The frequency of synchronisation between two neighbouring LPs depends on their mutually agreed lookahead. The larger the lookahead values are, the less frequent synchronisation operations are performed. The lookahead value is recalculated every time after a synchronisation operation. However, the lookahead values could be small during the simulation, resulting from potential dense traffic on boundary links.

Secondly, we designed a graph partitioning heuristic for partitioning road networks. It considers the objective of minimising the number of neighbouring partitions in order to minimise the number of synchronisation messages. Compared to the METIS graph partitioning and stripe spatial partitioning, the proposed partitioning algorithm has improved the performance of the parallel simulation in our experiment. However, when MA protocol is used, its performance ultimately depends on the value of the mutually agreed lookahead between two neighbouring LPs.

Thirdly, one approach of increasing the lookahead of LPs using MA protocol was studied, which is to relax the synchronisation. Synchronisation is relaxed either by simply skipping synchronisation operations, or by reducing the temporal resolution of the agents at the boundaries of partitions. Both methods apply a suitable amount of relaxation according to the stability of traffic under different traffic conditions, with the goal to maximise relaxation and meanwhile not to distort simulation results statistically. Experiments have shown that relaxation does not sacrifice the accuracy of the simulation when the amount of relaxation is controlled. Lookahead of LPs is increased and the speed-up of the parallel simulation has been improved.

Lastly, another approach of increasing the lookahead of LPs using MA protocol was studied, which uses computation replication to replace some synchronisation operations. The concept is that some synchronisation operations are not required if the data exchanged originally by those operations can be generated by the LPs themselves. The challenges are firstly how the suitable amount of redundant computation and data to exchange for a certain synchronisation frequency can be determined, and secondly how the trade-off between the overhead of computation replication and the benefit of reduced synchronisation messages can be managed. To address the challenges, extended layers of partitions are first defined based on agent behaviours, in order to regulate the necessary computation replication to achieve a certain synchronisation frequency. Then, the optimum number of extended layers to be replicated is dynamically determined according to traffic conditions during the simulation. The benefit of this approach depends on many factors, including computation time of agents and message-passing overhead. This approach works best for the case where replicating computation is not so time-consuming as sending messages.

This work has advanced the state-of-the-art of parallel agent-based road traffic simulation in the following aspects:

1. It complements the study of the partitioning and load-balancing of parallel agent-based road traffic simulation by considering one more objective in partitioning the road network for reducing the synchronisation overhead.
2. It explores the uncertainty in traffic simulation to reduce synchronisations messages in the simulation.
3. It provides an accurate definition of extended layers for computation replication in parallel agent-based road traffic simulation and an analysis of the overhead of computation replication.

The findings in this work can be of interest to engineers and researchers who conduct large-scale microscopic or nanoscopic road traffic simulations for their respective problems. Fast execution of parallel simulation can help in time-critical decision-making

applications, and studies where a large number of repeated simulation runs are required.

Examples are:

- faster-than-real-time short-term traffic prediction
- study on the impact of adopting autonomous vehicles on the road traffic in a large-scale
- study on the impact of adopting EVs on the road traffic, and environment
- study on the impact of adopting certain applications of ITS on the road traffic, for example, vehicle-to-vehicle communication

This research may also be of interest to researchers from other application domains, such as agent-based crowd simulation. The load-balancing and synchronisation methodologies can be adapted to those domains.

7.2 Future Work

As a result of this study, further research can be conducted in many areas. They are listed as follows.

Firstly, the neighbour-restricting constraint in our proposed graph partitioning heuristic can be investigated for diffusion-based dynamic graph repartitioning (i.e., incrementally changing the areas of partitions to balance the workload of LPs). Meanwhile, the trade-off between reducing neighbouring partitions and reducing potential communication data volume can be considered.

Secondly, for the relaxation of synchronisation by skipping synchronisation operations, more intelligent estimation functions could be developed and different functions can be developed for different traffic conditions. Better estimation functions can reduce the uncertainty introduced by relaxation.

Thirdly, for the relaxation of synchronisation by reducing the temporal resolution of models, mesoscopic or macroscopic can be considered to use on boundary links. Those models may allow larger lookahead. If the models with coarser resolution are calibrated,

statistical tests may not be required to check simulation results. This will make it easier for the practical adoption of the proposed approach.

Fourthly, for both of the relaxation methods, smarter ways of obtaining the parameter values than doing trial experiments are useful, since doing trial experiments may compensate the benefit of relaxing the synchronisation.

Fifthly, the definition of extended layers can be adapted for the inner areas of partitions. It can be used for overlapping communication and computation to further improve the performance of parallel simulation. Overlapping communication and computation is a way to hide the message-passing overhead of synchronisation.

Lastly, partitioning of the road network affects locations and shapes of extended layers, thus the performance of computation replication may be further improved if the partitioning of the road network considers computation replication.

Bibliography

- [1] United Nations, *World Urbanization Prospects, the 2014 Revision*, 2014.
- [2] United Nations Human Settlements Programme (UN-HABITAT), United Nations Advisory Committee of Local Authorities (UNACLA), “Transport and mobility snapshots 2011/2012,” 2013. [Online]. Available: www.unhabitat.org
- [3] M. Persula, “Simulation of traffic systems - an overview,” *Journal of Geographic Information and Decision Analysis*, vol. 3, no. 1, pp. 1–8, 1999.
- [4] L. A. Pipes, “An operational analysis of traffic dynamics,” *Journal of Applied Physics*, vol. 24, no. 3, pp. 274–281, 1953.
- [5] D. Ni, “2DSIM: a prototype of nanoscopic traffic simulation,” in *Proceedings of the 2003 IEEE Intelligent Vehicles Symposium*. Columbus, OH, USA: IEEE, June 9-11, 2003, pp. 47–52.
- [6] E. Bonabeau, “Agent-based modeling: methods and techniques for simulating human systems,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. Suppl 3, pp. 7280–7287, May 2002.
- [7] J. Barceló, J. L. Ferrer, D. García, M. Florian, and E. L. Saux, “Parallelization of microscopic traffic simulation for ATIS systems analysis,” in *Equilibrium and Advanced Transportation Modelling*, P. Marcotte and S. Nguyen, Eds. Springer US, 1998, pp. 1–26.
- [8] K. Nagel and M. Rickert, “Parallel implementation of the TRANSIMS,” *Parallel Computing*, vol. 27, no. 12, pp. 1611–1639, 2001.

- [9] T. Suzumura and H. Kanezashi, “Highly scalable X10-based agent simulation platform and its application to large-scale traffic simulation,” in *Proceedings of the 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*. Dublin, Ireland: IEEE, October 25-27, 2012, pp. 243–250.
- [10] C. Xu and F. Lau, *Load Balancing in Parallel Computers Theory and Practice*, ser. The Springer International Series in Engineering and Computer Science. Boston, MA: Springer US, 1997, vol. 381.
- [11] R. M. Fujimoto, *Parallel and distributed simulation*. New York, NY, USA: John Wiley & Sons, Inc., 1999.
- [12] D. M. Nicol, “Principles of conservative parallel simulation,” in *Proceedings of the 1996 Winter simulation Conference*. Coronado, CA, USA: IEEE, December 08-11, 1996, pp. 128–135.
- [13] D. Igbe, “Dynamic load balancing of parallel road traffic simulation,” PhD Thesis, University of Westminster, 2010.
- [14] R. Klefstad and Y. Zhang, “A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation,” in *Proceedings of the 2005 IEEE Intelligent Transportation Systems Conference*. Vienna, Austria: IEEE, September 13-16, 2005, pp. 813–818.
- [15] D. Wei, F. Chen, and X. Sun, “An improved road network partition algorithm for parallel microscopic traffic simulation,” in *Proceedings of the 2010 International Conference on Mechanic Automation and Control Engineering (MACE)*, Wuhan, China, June 26-28, 2010, pp. 2777 – 2782.
- [16] G. Karypis and V. Kumar, “Multilevel k-way partitioning scheme for irregular graphs,” *Journal of Parallel and Distributed Computing*, vol. 48, no. 1, pp. 96–129, 1998.

- [17] K. Schloegel, G. Karypis, and V. Kumar, “Wavefront diffusion and LMSR: Algorithms for dynamic repartitioning of adaptive meshes,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 5, pp. 451–466, 2001.
- [18] G. Cameron, “PARAMICS—parallel microscopic simulation of road traffic,” *The Journal of Supercomputing*, vol. 53, no. 1, pp. 25–53, 1996.
- [19] R. Ayani and H. Rajaei, “Parallel simulation using conservative time windows,” in *Proceedings of the 24th Winter Simulation Conference*. Arlington, VA, USA: ACM, December 13-16, 1992, pp. 709–717.
- [20] M. Namekawa, A. Satoh, H. Mori, K. Yikai, and T. Nakanishi, “Clock synchronization algorithm for parallel road-traffic simulation system in a wide area,” *Mathematics and Computers in Simulation*, vol. 48, no. 4-6, pp. 351–359, 1999.
- [21] M. Papageorgiou, I. Papamichail, A. Messmer, and Y. Wang, “Traffic simulation with METANET,” in *Fundamentals of Traffic Simulation*, ser. International Series in Operations Research & Management Science, J. Barceló, Ed. Springer New York, 2010, vol. 145, pp. 399–430.
- [22] M. Balmer, K. Meister, M. Rieser, and K. Nagel, “Agent-based simulation of travel demand: structure and computational performance of MATSim-T,” in *the 2nd TRB Conference on Innovations in Travel Modeling*, Portland, USA, August, 2008.
- [23] M. Ben-akiva, H. N. Koutsopoulos, C. Antoniou, and R. Balakrishna, “Traffic simulation with DynaMIT,” in *Fundamentals of Traffic Simulation*, ser. International Series in Operations Research & Management Science, J. Barceló, Ed. Springer New York, 2010, vol. 145, pp. 363–398.
- [24] D. Krajzewicz, “Traffic simulation with SUMO - Simulation of Urban Mobility,” in *Fundamentals of Traffic Simulation*, ser. International Series in Operations Research & Management Science, J. Barceló, Ed. New York, NY: Springer New York, 2010, vol. 145, pp. 269–293.

- [25] M. Ben-Akiva, H. N. Koutsopoulos, T. Toledo, Q. Yang, C. F. Choudhury, C. Antoniou, R. Balakrishna, and J. Barceló, “Traffic simulation with MITSIMLab,” in *Fundamentals of Traffic Simulation*, ser. International Series in Operations Research & Management Science, J. Barceló, Ed. Springer New York, 2010, vol. 145, pp. 233–268.
- [26] M. Fellendorf, P. Vortisch, and J. Barceló, “Microscopic traffic flow simulator VISSIM,” in *Fundamentals of Traffic Simulation*, ser. International Series in Operations Research & Management Science, J. Barceló, Ed. Springer New York, 2010, vol. 145, pp. 63–93.
- [27] J. Casas, J. L. Ferrer, D. Garcia, J. Perarnau, A. Torday, and J. Barceló, “Traffic simulation with Aimsun,” in *Fundamentals of Traffic Simulation*, ser. International Series in Operations Research & Management Science, J. Barceló, Ed. Springer New York, 2010, vol. 145, pp. 173–232.
- [28] D. Neunzig, M. Weilkes, A. Hochstädter, and J. Ludmann, “Assessment of advanced vehicle control systems with the vehicle oriented traffic simulation tool PELOPS,” in *SAE Technical Paper 981890*, 1998, pp. 1–7.
- [29] M. J. Lighthill and G. B. Whitham, “On kinematic waves. ii. a theory of traffic flow on long crowded roads,” *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 229, no. 1178, pp. 317–345, 1955.
- [30] P. I. Richards, “Shock waves on the highway,” *Operations Research*, vol. 4, no. 1, pp. 42–51, 1956.
- [31] D. Helbing, “Theoretical foundation of macroscopic traffic models,” *Physica A: Statistical and Theoretical Physics*, vol. 219, no. 3, pp. 375–390, 1995.
- [32] J. Barceló, “Models, traffic models, simulation, and traffic simulation,” in *Fundamentals of Traffic Simulation*, ser. International Series in Operations Research & Management Science, J. Barceló, Ed. Springer New York, 2010, vol. 145, pp. 1–62.

- [33] M. Mahut, M. Florian, and M. B. Principles, “Traffic simulation with Dynameq,” in *Fundamentals of Traffic Simulation*, ser. International Series in Operations Research & Management Science, J. Barceló, Ed. Springer New York, 2010, vol. 145, pp. 323–361.
- [34] K. S. Perumalla, B. G. Aaby, S. B. Yoginath, and S. K. Seal, “GPU-based real-time execution of vehicular mobility models in large-scale road network scenarios,” in *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation (PADS’09)*, Lake Placid, New York, USA, June 22-25, 2009, pp. 95–103.
- [35] Y. Xu, G. Tan, X. Li, and X. Song, “Mesoscopic Traffic Simulation on CPU/GPU,” in *Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM PADS’14)*, Denver, CO, USA, May 18-21, 2014, pp. 39–49.
- [36] R. Wiedemann, “Simulation des Straßenverkehrsflusses,” in *Schriftenreihe des Instituts für Verkehrswesen der Universität Karlsruhe*. University of Karlsruhe, Germany, 1974.
- [37] P. G. Gipps, “A behavioural car-following model for computer simulation,” *Transportation Research Part B: Methodological*, vol. 15, no. 2, pp. 105–111, 1981.
- [38] ———, “A model for the structure of lane-changing decisions,” *Transportation Research Part B: Methodological*, vol. 20, no. 5, pp. 403–414, 1986.
- [39] M. Treiber, A. Hennecke, and D. Helbing, “Congested traffic states in empirical observations and microscopic simulations,” *Physical Review E*, vol. 62, no. 2, pp. 1805–1824, 2000.
- [40] A. Kesting, M. Treiber, and D. Helbing, “General lane-changing model MOBIL for car-following models,” *Transportation Research Record*, vol. 1999, no. 1, pp. 86–94, Jan. 2007.

- [41] M. Brackstone and M. McDonald, “Car-following: a historical review,” *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 2, no. 4, pp. 181–196, Dec. 1999.
- [42] Y. Hollander and R. Liu, “The principles of calibrating traffic microsimulation models,” *Transportation*, vol. 35, no. 3, pp. 347–362, Jan. 2008.
- [43] K. Nagel, M. Schreckenberg, and Others, “A cellular automaton model for free-way traffic,” *Journal de Physique I*, vol. 2, no. 12, pp. 2221–2229, 1992.
- [44] W. Burghout, H. Koutsopoulos, and I. Andreasson, “A discrete-event mesoscopic traffic simulation model for hybrid traffic simulation,” in *Proceedings of the 2006 IEEE Intelligent Transportation Systems Conference (ITSC’06)*. Toronto, ON, Canada: IEEE, September 17-20, 2006, pp. 1102–1107.
- [45] J. Vilaro, A. Torday, and A. Gerodimos, “Combining mesoscopic and microscopic simulation in an integrated environment as a hybrid solution,” *Intelligent Transportation Systems Magazine, IEEE*, vol. 2, no. 3, pp. 25–33, 2010.
- [46] R. Claes and T. Holvoet, “Multi-model traffic microsimulations,” in *Proceedings of the 2009 Winter Simulation Conference (WSC’09)*. Austin, TX, USA: IEEE, December 13-16 2009, pp. 1113–1123.
- [47] H. Aydt, Y. Xu, M. Lees, and A. Knoll, “A multi-threaded execution model for the agent-based SEMSim traffic simulation,” in *Proceedings of the 13th International Conference on Systems Simulation*. Singapore: Springer Berlin Heidelberg, November 06–08, 2013, pp. 1–12.
- [48] D. Ni, “Multiscale modeling of traffic flow,” *Mathematica Aeterna*, vol. 1, no. 1, pp. 27–54, 2011.
- [49] L. Schnieder, “Systems-theoretic foundation for advanced driver assistance systems,” in *Proceedings of the 4th Annual IEEE Systems Conference*. San Diego, CA, USA: IEEE, April 5-8, 2010, pp. 365–370.

- [50] M. Schiller, M. Dupuis, D. Krajzewicz, A. Kern, and A. Knoll, "Multi-resolution traffic simulation for large scale high fidelity evaluation of VANET applications," in *SUMO User Conference 2015 - Intermodal Simulation for Intermodal Transport*, 2015.
- [51] N. Yuhara and J. Tajima, "Multi-driver agent-based traffic simulation systems for evaluating the effects of advanced driver assistance systems on road traffic accidents," *Cognition, Technology & Work*, vol. 8, no. 4, pp. 283–300, Sep. 2006.
- [52] R. Maia, M. Silva, R. Araujo, and U. Nunes, "Electric vehicle simulator for energy consumption studies in electric mobility systems," in *Proceedings of the 2011 IEEE Forum on Integrated and Sustainable Transportation System (FISTS)*. Vienna, Austria: IEEE, June 29-July 1, 2011, pp. 227–232.
- [53] S. Goyal, Y. Zhang, and A. Martinoli, "A realistic simulator for the design and evaluation of intelligent vehicles," in *Proceedings of the 13th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. Madeira, Portugal: IEEE, September 19-22, 2010, pp. 1039–1044.
- [54] T. Potuzak, "Methods for division of road traffic networks focused on load-balancing," *Advances in Computing*, vol. 2, no. 4, pp. 42–53, 2012.
- [55] G. Cybenko, "Dynamic load balancing for distributed memory multiprocessors," *Journal of parallel and distributed computing*, vol. 7, no. 2, pp. 279–301, 1989.
- [56] S. Hosseini, B. Litow, M. Malkawi, J. McPherson, and K. Vairavan, "Analysis of a graph coloring based distributed load balancing algorithm," *Journal of Parallel and Distributed Computing*, vol. 10, no. 2, pp. 160–166, Oct. 1990.
- [57] L. Oliker and R. Biswas, "PLUM : Parallel Load Balancing for Adaptive Unstructured Meshes," *Journal of Parallel and Distributed Computing*, vol. 52, no. 2, pp. 150–177, Aug. 1998.
- [58] Y. Hu and R. Blake, "An optimal dynamic load balancing algorithm," Ph.D. dissertation, Daresbury Laboratory, Warrington, UK, 1995.

- [59] C. Walshaw, M. Cross, and M. Everett, "Parallel dynamic graph partitioning for adaptive unstructured meshes," *Journal of Parallel and Distributed Computing*, vol. 47, no. 2, pp. 102–108, Dec. 1997.
- [60] H. Meyerhenke, B. Monien, and T. Sauerwald, "A new diffusion-based multilevel algorithm for computing graph partitions," *Journal of Parallel and Distributed Computing*, vol. 69, no. 9, pp. 750–761, Sep. 2009.
- [61] M. H. Willebeek-LeMair and A. P. Reeves, "Strategies for dynamic load balancing on highly parallel computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 9, pp. 979–993, 1993.
- [62] D.-H. Lee, "A framework for parallel traffic simulation using multiple instancing of a simulation program," *Journal of Intelligent Transportation Systems*, vol. 7, no. 3, pp. 279–294, Jul. 2002.
- [63] M. Berger and S. Bokhari, "A partitioning strategy for nonuniform problems on multiprocessors," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 570–580, 1987.
- [64] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel Computing*, vol. 26, no. 12, pp. 1519–1534, Nov. 2000.
- [65] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell system technical journal*, vol. 49, no. 1, pp. 291–307, 1970.
- [66] P. Sanders and C. Schulz, "Distributed evolutionary graph partitioning," in *Proceedings of the Meeting on Algorithm Engineering & Experiments, ALENEX '12*, Kyoto, Japan, January 16, 2011, pp. 16–29.
- [67] D. Zhang, C. Jiang, and S. Li, "Research on dynamic load balancing algorithms for parallel transportation simulations," in *Proceedings of the 7th International Conference on Advanced Parallel Processing Technologies*. Guangzhou, China: Springer Berlin, Heidelberg, November 22–23, 2007, pp. 560–568.

- [68] X. Sun, F. Chen, X. Li, and X. Wang, “An improved dynamic load balancing algorithm for parallel microscopic traffic simulation,” in *Proceedings of the 2012 International Conference on Measurement, Information and Control (MIC)*. Harbin, China: IEEE, May 18-20, 2012, pp. 600–604.
- [69] S. Jafer, Q. Liu, and G. Wainer, “Synchronization methods in parallel and distributed discrete-event simulation,” *Simulation Modelling Practice and Theory*, vol. 30, pp. 54–73, 2013.
- [70] K. Chandy and J. Misra, “Distributed simulation: a case study in design and verification of distributed programs,” *IEEE Transactions on Software Engineering*, vol. SE-5, no. 5, pp. 440–452, 1979.
- [71] D. R. Jefferson, “Virtual time,” *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, pp. 404–425, Jul. 1985.
- [72] B. D. Lubachevsky, “Efficient distributed event-driven simulations of multiple-loop networks,” *Communications of the ACM*, vol. 32, no. 1, pp. 111–123, 1989.
- [73] D. Nicol, C. Micheal, and P. Inouye, “Efficient Aggregation Of Multiple LPs In Distributed Memory Parallel Simulations,” in *Proceedings of the 1989 Winter Simulation Conference*, Washington, DC, USA, December 4-6, 1989, pp. 680–685.
- [74] J. S. Steinman, “SPEEDES: Synchronous parallel environment for emulation and discrete event simulation,” in *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, Anaheim, CA, USA, January 23-25, 1991, pp. 95–103.
- [75] R. E. Bryant, “Simulation of packet communication architecture computer systems,” in *Technical Report*. MIT Laboratory for Computer Science, 1977.
- [76] W.-K. Su and C. L. Seitz, “Variants of the Chandy-Misra-Bryant distributed discrete-event simulation algorithm,” in *Proceedings of the SCS Multiconference on Distributed Simulation*, Tampa, Florida, March 28-31, 1989, pp. 38–43.

- [77] W. Cai and S. J. Turner, “An algorithm for distributed discrete-event simulation - The carrier null message approach,” in *Proceedings of the SCS Multiconference on Distributed Simulation*, San Diego, CA, USA, January 17-19, 1990, pp. 3–8.
- [78] D. M. Nicol, “Parallel discrete-event simulation of fcfs stochastic queueing networks,” in *Proceedings of the ACM/SIGPLAN Conference on Parallel Programming: Experience with Applications, Languages and Systems*. New Haven, CT, USA: ACM, July 19-21, 1988, pp. 124–137.
- [79] Y.-H. Low, C.-c. Lim, W. Cai, W. J. Hsu, B.-p. Gan, S. Jain, S. Y. Huang, and S. J. Turner, “Conservative parallel simulation for manufacturing systems,” in *Proceedings of the 8th International Parallel Computing Workshop (PCW’98)*, Singapore, September 07-08, 1998, pp. 293–300.
- [80] D. Jefferson, “Virtual Time II: Storage Management in Conservative and Optimistic Systems,” in *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing*, Quebec City, Quebec, Canada, August 22-24, 1990, pp. 75–89.
- [81] C. D. Carothers, D. Bauer, and S. Pearce, “ROSS: a High Performance Modular Time Warp System,” in *Proceedings of the 14th Workshop on Parallel and Distributed Simulation*. Bologna, Italy: IEEE, May 28-31, 2000, pp. 53–60.
- [82] K. S. Perumalla and S. K. Seal, “Discrete event modeling and massively parallel execution of epidemic outbreak phenomena,” *Simulation*, vol. 88, no. 7, pp. 768–783, 2012.
- [83] A. Gafni, “Rollback mechanisms for optimistic distributed simulation systems,” in *Proceedings of the SCS Multiconference on Distributed Simulation*, San Diego, CA, USA, February 3-5, 1988, pp. 61–67.
- [84] H. Bauer and C. Sporrer, “Reducing rollback overhead in time-warp based distributed simulation with optimized incremental state saving,” in *Proceedings of*

the 26th Annual Simulation Symposium, Arlington, VA, USA, March 29-April 1, 1993, pp. 12–20.

- [85] T. Potuzak, “Distributed traffic simulation and the reduction of inter-process communication using traffic flow characteristics transfer,” in *Proceedings of the 10th International Conference on Computer Modelling and Simulation, EUROSIM/UKSim2008*, 2008, pp. 525–530.
- [86] S. B. Yoginath and K. S. Perumalla, “Parallel vehicular traffic simulation using reverse computation-based optimistic execution,” in *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation (PADS)*, Roma, Italy, June 3-6, 2008, pp. 33–42.
- [87] M. Hanai, T. Suzumura, G. Theodoropoulos, and K. S. Perumalla, “Exact-differential large-scale traffic simulation,” in *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, London, UK, June 10-12 2015, pp. 271–280.
- [88] J. Helton, “Uncertainty and sensitivity analysis in the presence of stochastic and subjective uncertainty,” *Journal of Statistical Computation and Simulation*, vol. 57, no. 1-4, pp. 3–76, Apr. 1997.
- [89] W. L. Oberkampf, S. M. DeLand, B. M. Rutherford, K. V. Diegert, and K. F. Alvin, “Error and uncertainty in modeling and simulation,” *Reliability Engineering & System Safety*, vol. 75, no. 3, pp. 333–357, 2002.
- [90] G. de Jong, A. Daly, M. Pieters, S. Miller, R. Plasmeijer, and F. Hofman, “Uncertainty in traffic forecasts: literature review and new results for The Netherlands,” *Transportation*, vol. 34, no. 4, pp. 375–395, Dec. 2006.
- [91] R. F. Benekohal and G. Abu-Lebdeh, “Variability analysis of traffic simulation outputs: Practical approach for TRAF-NETSIM,” *Transportation Research Record*, vol. 1457, pp. 198–207, 1994.

- [92] M. Bayarri, J. Berger, G. Molina, N. M. Rouphail, and J. Sacks, "Assessing uncertainties in traffic simulation: a key component in model calibration and validation," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1876, no. 1, pp. 32–40, 2004.
- [93] R. M. Fujimoto, "Exploiting temporal uncertainty in parallel and distributed simulations," in *Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS'99)*. Atlanta, GA, USA: IEEE, May 01-04, 1999, pp. 46–53.
- [94] D. M. Nicol, "Exploiting uncertainty and error to accelerate simulations." in *Keynote Addresses of International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2012)*, 2012.
- [95] M. L. Loper and R. M. Fujimoto, "A case study in exploiting temporal uncertainty in parallel simulations," in *Proceedings of the 33rd International Conference on Parallel Processing*. Montreal Quebec, Canada: IEEE, 2004, pp. 161–168.
- [96] H. Park and P. a. Fishwick, "An analysis of queuing network simulation using GPU-based hardware acceleration," *ACM Transactions on Modeling and Computer Simulation*, vol. 21, no. 3, pp. 1–22, 2011.
- [97] O. Rihawi, Y. Secq, and P. Mathieu, "Relaxing synchronization constraints in distributed agent-based simulations," *Jurnal Teknologi*, vol. 63, no. 3, pp. 65–76, 2013.
- [98] C. Ding and Y. He, "A ghost cell expansion method for reducing communications in solving PDE problems," in *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*. Denver, CO, USA: ACM, November 10-16, 2001, pp. 1–12.
- [99] J. Demmel, "Avoiding communication in sparse matrix computations," in *Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008)*. Miami, FL, USA: IEEE, April 14-18, 2008, pp. 1–12.
- [100] B. G. Aaby, K. S. Perumalla, and S. K. Seal, "Efficient simulation of agent-based models on multi-GPU and multi-core clusters," in *Proceedings of the 3rd Interna-*

- tional ICST Conference on Simulation Tools and Techniques (SIMUTools 2010)*. Torremolinos, Malaga, Spain: ICST, March 15-19, 2010, pp. 1–10.
- [101] T. Zou, G. Wang, and M. Salles, “Making time-stepped applications tick in the cloud,” in *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC 2011)*. Cascais, Portugal: ACM, October 26-28, 2011, pp. 1–14.
 - [102] R. Zunino, “Trading computation time for synchronization time in spatial distributed simulation,” in *Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS)*. Nice, France: IEEE, June 14-17, 2011, pp. 1–8.
 - [103] Y. Xu, H. Aydt, and M. Lees, “SEMSim: a distributed architecture for multi-scale traffic simulation,” in *Proceedings of the 26th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS)*. Zhangjiajie, China: IEEE, July 15-19, 2012, pp. 178–180.
 - [104] A. Ferscha and S. Tripathi, “Parallel and distributed simulation of discrete event systems,” Univ. of Maryland Institute for Advanced Computer Studies, Tech. Rep., 1995.
 - [105] M. Lees, B. Logan, R. Minson, T. Oguara, and G. Theodoropoulos, “Modelling environments for distributed simulation,” in *Environments for Multi-Agent Systems*, ser. Lecture Notes in Computer Science, D. Weyns, H. Van Dyke Parunak, and F. Michel, Eds. Springer Berlin Heidelberg, 2005, vol. 3374, pp. 150–167.
 - [106] G. Karypis and V. Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs,” *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, Jan. 1998.
 - [107] A. Pothen, H. Simon, and K. Liou, “Partitioning sparse matrices with eigenvectors of graphs,” *SIAM Journal on Matrix Analysis and Applications*, vol. 11, no. 2, pp. 430–452, 1990.

- [108] D. Helbing, M. Treiber, A. Kesting, and M. Schönhof, “Theoretical vs. empirical classification and prediction of congested traffic states,” *The European Physical Journal B*, vol. 69, no. 4, pp. 583–598, 2009.
- [109] M. Treiber and A. Kesting, “Stability analysis,” in *Traffic Flow Dynamics*. Springer Berlin Heidelberg, 2013, pp. 157–180.
- [110] —, “Elementary car-following models,” in *Traffic Flow Dynamics*. Springer Berlin Heidelberg, 2013, pp. 257–301.
- [111] —, “Lane-changing and other discrete-choice situations,” in *Traffic Flow Dynamics*. Springer Berlin Heidelberg, 2013, pp. 239–255.
- [112] A. Kesting and M. Treiber, “How reaction time, update time, and adaptation time influence the stability of traffic flow,” *Computer-Aided Civil and Infrastructure Engineering*, vol. 23, no. 2, pp. 125–137, 2008.
- [113] B. Park and H. Qi, “Development and evaluation of a procedure for the calibration of simulation models,” *Transportation Research Record: Journal of the Transportation Research Board*, no. 1934, pp. 208–217, 2005.
- [114] K. Basak, S. N. Hetu, C. L. Azevedo, H. Loganathan, T. Toledo, and M. Ben-Akiva, “Modeling reaction time within a traffic simulation model,” in *Proceedings of the 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. The Hague, Netherlands: IEEE, October 6-9 2013, pp. 302–309.
- [115] B. Efron, “Bootstrap methods: another look at the jackknife,” *The Annals of Statistics*, vol. 7, no. 1, pp. 1–26, 1979.
- [116] K. Al-Tawil and C. A. Moritz, “Performance Modeling and Evaluation of MPI,” *Journal of Parallel and Distributed Computing*, vol. 61, no. 2, pp. 202–223, 2001.
- [117] T. Hoefler, W. Gropp, R. Thakur, and J. L. Träff, “Toward performance models of MPI implementations for understanding application scaling issues,” in *Proceedings of the 17th European MPI users’ group meeting conference on Recent*

Advances in the message passing interface, Stuttgart, Germany, September 12-15, 2010, pp. 21–30.

- [118] A. Kesting, M. Treiber, and D. Helbing, “Agents for traffic simulation,” *Multi-Agent Systems Simulation and Applications*, pp. 325–356, 2008.

Appendix A

Threshold Estimation for Initiating Load-balancing Operations

This appendix shows the *estimation* of the threshold value for invoking load-balancing operations. We first analyse the benefit and the overhead of a load-balancing operation. Supposing a load-balancing operation is performed at time t , the reduction of execution time for one update interval at time t is

$$(Imb^t - Imb_{bal}^t) \cdot Ta \quad (A.1)$$

where Imb^t and Imb_{bal}^t are the load imbalances at time t before and after the load-balancing operation respectively, in terms of number of agents, and Ta is the execution time of the models in an agent in one update interval. Supposing the period for evaluating workload imbalance in terms of the number of update intervals is N_{dlb} , let $benefit_dlb$ be the benefit of a load-balancing operation for N_{dlb} update intervals, i.e., reduction of the execution time resulting from the load-balancing operation. Then, we have

$$benefit_dlb = \sum_{x=1}^{N_{dlb}} (Imb^{t+x \cdot \delta} - Imb_{bal}^{t+x \cdot \delta}) \cdot Ta \quad (A.2)$$

Equation A.2 does not include the influence of possible change of synchronisation overhead resulting from the load-balancing operation, because it is not straightforward to quantify the change of synchronisation overhead. The overhead of a load-balancing operation is the sum of the time for adjusting the partitions of the road network, i.e., repartitioning, and the time spent on re-distributing agents to the new partitions. The time overheads can be obtained by running the partitioning algorithm once and profiling the cost of sending messages and potential re-distribution of agents. Apparently, the net profit of performing dynamic load-balancing, i.e., the different between the benefit and the overhead depends on the load imbalance before and after the load-balancing operation, the time for re-partitioning, and the time for re-distributing agents. The threshold value should ensure that the enefit of dynamic load-balancing exceeds the overhead.

$Imb^{t+x\cdot\delta}$ and $Imb_{bal}^{t+x\cdot\delta}$ ($x \in \mathbb{Z}, 1 \leq x \leq N_{dlb}$) in Equation A.2 are unknown at simulation time t , because they are only available in the future simulation time. Therefore, the value of *benefit_dlb* can only be estimated using some heuristics. In order to represent Equation A.2 using the available information at simulation time t , we need to make some assumptions. Supposing the workload of LPs are perfectly balanced after the load-balancing operation, we obtain $Imb^t - Imb_{bal}^t = Imb^t$. We also suppose that $Imb^{t+x\cdot\delta} - Imb_{bal}^{t+x\cdot\delta}$ remains as a constant Imb^t in the simulation period $[t, t + N_{dlb} \cdot \delta]$. Thus, Equation A.2 can be derived as follows:

$$benefit_dlb = N_{dlb} \cdot Imb^t \cdot Ta \quad (A.3)$$

To make the benefit larger than the overhead, suppose that the overhead has an upper bound *overhead_dlb*, then the following inequality needs to be satisfied:

$$N_{dlb} \cdot Imb^t \cdot Ta > overhead_dlb \quad (A.4)$$

The upper bound can be estimated by profiling the time spent on partitioning and re-distributing agents. Divide the both sides of Formula A.4 by $N_{dlb} \cdot Ta$, we can obtain

$$Imb^t > \frac{overhead_dlb}{N_{dlb} \cdot Ta} \quad (A.5)$$

Therefore, the threshold of initiating a load-balancing operation should be

$$\frac{overhead_dlb}{N_{dlb} \cdot Ta} \quad (A.6)$$

This threshold ensures that the estimated benefit is always greater than the overhead introduced. Note that this threshold is in terms of *the number of agents*. A threshold in terms of computation time can be obtained by multiplying expression A.6 by Ta .

Appendix B

Lookahead Determination of MA synchronisation Protocol

B.1 Lookahead based on Traffic Conditions

The lookahead in MA synchronisation protocol is *directed*, which means that the lookahead values of an LPs towards different neighbouring LPs at a certain simulation time may be different. The lookahead $l_{i,j}^t$ is the minimum time that any agent in G_i or any agent to be created in G_i requires to travel into buffer region $B_{j,i}$. Specifically, there are three cases that need to be considered for different traffic conditions.

The first case is that no agents exist in local space G_i . This usually happens when the simulation just starts. The minimum time for any newly created agent to travel into $B_{j,i}$ is denoted as $\Delta t_{new_{i,j}}$. Then, $l_{i,j}^t = \Delta t_{new_{i,j}}$. $\Delta t_{new_{i,j}}$ can be estimated as follows: let the set of outgoing boundary links from G_i towards G_j at time t be $O_{i,j}^t$, and the minimum travelling time from the start of a link l ($l \in O_{i,j}^t$) to the buffer region of that link is Δt_{ol} , then $\Delta t_{new_{i,j}} = \min_{l \in O_{i,j}^t} (\Delta t_{ol})$. We assume that agents are only created at the beginning of links, namely, on nodes. The travelling time here is estimated from speed limit. The actual travelling time is usually longer, since the actual velocities of agents are usually smaller than the speed limit, in other words, Δt_{ol} and $\Delta t_{new_{i,j}}$ are conservative estimations.

Figure B.1: Scenarios of lookahead calculation using MA synchronisation protocol. (a) No local agents exist. A new agent is being created in a network node. (b) Buffer regions are empty and there are no migrating agents. (c) A new agent is being created in the scenario of (b). (d) Buffer regions are not empty.

An example of Δt_{o_l} is shown in Figure B.1(a).

The second case is that local agents exist, but there are no agents inside buffer regions or ready for migration before the next update interval, as depicted in Figures B.1(b) and (c). The lookahead is the minimum time for the existing agents and new agents to travel to a buffer region. Migration of local agents only happens on outgoing links, therefore, the agents whose routes pass through outgoing boundary links should be examined. Note that new agents may also be created (see Figure B.1(c)). Suppose the minimum time for any existing agent to travel to buffer region $B_{j,i}$ is $\Delta t_{exist_{i,j}}$, then $l_{i,j}^t = \min(\Delta t_{exist_{i,j}}, \Delta t_{new_{i,j}})$. Since agent states are only updated with an interval δ , $l_{i,j}^t$ is set to δ if $l_{i,j}^t < \delta$.

The third case is that there are agents inside buffer regions or ready for migration in the current update interval. Lookahead can only be as large as one update interval (i.e., δ), because shared states must be updated and agents ready for migration must be transferred before agents take actions for the next update interval. The case is illustrated in Figure B.1(d). There is an agent inside the buffer region of an incoming boundary link 1 of G_i and another inside the buffer region of an outgoing boundary link 2.

B.2 Lookahead of Relaxed Synchronisation By Increasing Update Intervals

If there is any agent inside $B_{j,i}$ or ready for migration from LP_i to LP_j , the lookahead is set to the relaxed update interval $\delta_{i,j}^t$. $l_{i,j}^t$ will always be greater than or equal to $\delta_{i,j}^t$. The calculation of $l_{i,j}^t$ by LP_i is shown in Algorithm 8.

Algorithm 8: Lookahead determination in MA relaxed with reduced temporal resolution

Data:

- A_i^t set of agents in LP_i at time t
 - $M_{i,j}^t$ set of agents migrating from LP_i to LP_j
 - $S_{i,j}^t$ set of shared states that should be sent by LP_i to LP_j
 - $\Delta t_{new,i,j}$ minimum time for any new agent to travel to the buffer region $B_{j,i}$
 - $\Delta t_{exist,i,j}$ minimum time for any agent in A_i^t to travel from its current position to buffer region $B_{j,i}$
 - $\delta_{i,j}^t$ relaxed update interval for agents at the boundary between G_i and G_j
- Result:** lookahead from LP_i towards LP_j at time t , $l_{i,j}^t$
-

```

1 initialise  $l_{i,j}^t \leftarrow \text{maximum double}$ ;
2 if  $A_i^t = \emptyset$  then
3   |  $l_{i,j}^t \leftarrow \Delta t_{new,i,j}$ ;
4 else if  $M_{i,j}^t = \emptyset \wedge S_{i,j}^t = \emptyset$  then
5   |  $l_{i,j}^t \leftarrow \min(\Delta t_{exist,i,j}, \Delta t_{new,i,j})$ ;
6 else
7   |  $l_{i,j}^t \leftarrow \delta_{i,j}^t$ ;
8 end
9  $l_{i,j}^t \leftarrow \max(l_{i,j}^t, \delta_{i,j}^t)$  ;

```

Appendix C

Driver Behaviour Models

This appendix describes the three driver behaviour models used in this study.

C.1 Acceleration Model

Intelligent Driver Model (IDM) model is a car-following model that describes the dynamics of the positions and velocities of single vehicles [39]. It is a time-continuous car-following model. For a vehicle, the acceleration is assumed to be a continuous function of its velocity v , the gap s between the vehicle and the vehicle ahead, and the velocity difference Δv to the vehicle ahead:

$$\dot{v} = a \left[1 - \left(\frac{v}{v_0} \right)^4 - \left(\frac{s^*(v, \Delta v)}{s} \right)^2 \right] \quad (\text{C.1})$$

where v_0 is the desired velocity of the vehicle, and a is the maximum acceleration of the vehicle. The expression consists of two terms: a free road acceleration term $\dot{v}_{free}(v) = a[1 - (v/v_0)^4]$, and a braking deceleration term $\dot{v}_{brake}(s, v, \Delta v) = -a[s^*(v, \Delta v)/s]^2$. The deceleration term dominates when the vehicle is too close to the vehicle ahead. It depends on the ratio between the effective desired minimum gap and the actual gap s . The

desired minimum gap is determined by:

$$s^*(v, \Delta v) = s_0 + vT + \frac{v\Delta v}{2\sqrt{ab}} \quad (\text{C.2})$$

where s_0 is a safe distance to the vehicle ahead, T is the minimum safe time gap, and b is the comfortable deceleration of the vehicle (b has a positive value). When there is no vehicle ahead in the same lane, the deceleration term will be excluded from the acceleration model. IDM is a collision-free model due to the deceleration term. When the city traffic is simulated, there are always roads with different speed limits connecting with each other. If it happens that the velocity of a vehicle is greater than the desired velocity which is usually smaller than or equal to the speed limit, the free acceleration is replaced by:

$$\dot{v}_{free}(v) = -b \left[1 - \left(\frac{v_0}{v} \right)^4 \right] \quad (\text{C.3})$$

The velocities of vehicles are obtained by numerical integration of the acceleration over time. The positions of vehicles are calculated with the equations of motion. There is approximation error in the integration while the continuous model IDM is integrated by discrete time-steps (update intervals). The acceleration is commonly assumed to be constant within an update interval. The velocity and position of a vehicle at the end of an update interval are $v(t + \delta) = v(t) + \dot{v}(t) \cdot \delta$ and $x(t + \delta) = x(t) + v(t)\delta + \frac{1}{2}\dot{v}(t)\delta^2$, where $v(t)$ and $x(t)$ are the velocity and position of the vehicle at time t , and δ is the update interval of the vehicle.

The size of update intervals should be as small as possible. However, a smaller update interval is computationally inefficient. The update interval is typically set between 0.1 seconds and 0.2 seconds [118]. When the size of update interval is large (for example, 0.5 seconds or 1 second), the velocity at the end of an update interval can be greater than the speed-limit. The road network used in our experiment consists of a large number of links with various speed limits. Due to the large update interval we use (0.5 seconds), the acceleration output from IDM sometimes results in the velocity of a vehicle being greater

than the speed limit or less than 0. Therefore, the output of IDM is monitored and $v(t + \delta)$ is bounded to be greater than or equal to 0 and less than or equal to the speed limits of roads. Collisions of vehicles may also occur under some traffic conditions. There is also a collision detection mechanism. If two vehicles are detected to be overlapping, the vehicle behind will be forced to come to a halt. Another modification to the original IDM model is the introduction of sensing range. Agents only take actions when neighbouring vehicles are in their sensing ranges, which is not considered in the IDM model. This is necessary for controlling the data required for fulfilling data dependencies of LPs.

C.2 Lane-changing Model

The lane-changing model outputs a decision whether to stay in the current lane or change to an alternative lane. We develop a new lane-changing model which is shown in Figure C.1. The lane-changing decision starts with checking whether an alternative lane exists. All the lanes that belong to the route of an agent are assigned weights. If there is a turn or a junction, the lanes that allow turning are assigned greater weights than those that do not allow turn. Therefore, agents should tend to change to lanes with higher weights. When multiple alternative lanes exist, the lane with the higher weight is chosen as the target lane. Similar to other lane-changing models, there are discretionary lane-changing and compulsory lane-changing. If the position of an agent is within a certain distance to a turn or a junction, it makes a compulsory lane-changing decision. The agent changes its lane if the target lane has a higher weight than the current lane and it is safe to perform lane-changing according to a gap-acceptance model (presented in the next section of this appendix). If the agent is outside the distance to a turn or a junction, it makes a discretionary lane-change decision. The decision is based on whether the agent is able to gain an acceleration benefit after a lane change. The weights of the lanes do not play a role here. The agent first checks whether there is a slower preceding vehicle blocking its way in the current lane. If there is none, the agents stay in the current lane. If there is one, the agent checks the target lane to see if it can get an acceleration benefit. If it can obtain a higher acceleration, it will check whether there is enough space for it to change

Figure C.1: Flow diagram of the lane-changing decision of agents.

without causing a collision. If so, the agent changes to the target lane. Otherwise, it remains in the current lane.

C.3 Gap-acceptance Model

The gap-acceptance model here describes the judgement of a driver whether the gap between a lead vehicle and lag vehicle in the target lane allows a safe lane change. It is a new model.

There are a lead gap and a lag gap between the subject vehicle and the lead and lag vehicles, respectively, which are shown in Figure C.2. The lead gap and the lag gap should ensure that no collision happens if the subject vehicle changes its lane.

Figure C.2: Lead gap and lag gap in the target lane

The lead gap is denoted as s_{lead} , the lag gap as s_{lag} , the velocity of the lead vehicle as v_{lead} , the velocity of the lag vehicle as v_{lag} , and the velocity of the subject vehicle as v . To accept the gaps in the target lane, the following two conditions must be satisfied:

$$s_{lead} \geq s_0 + \frac{\delta}{2} \cdot (v + v_{max} - v_{lead}) \quad (C.4)$$

$$s_{lag} \geq s_0 + \frac{\delta}{2} \cdot (v_{lag} + v_{lag_{max}} - v) \quad (C.5)$$

where $v_{lag_{max}}$ and v_{max} are the maximum velocities that the lag vehicle and the subject vehicle can reach at the end of the update interval, respectively. The first condition depicts that the lead gap should be at least as big as the safe gap even when the lead vehicle comes to an emergency stop and the subject vehicle accelerates with the maximum acceleration. The second condition depicts that the lag gap should be at least as big as the safe gap even when the subject vehicle comes to an emergency stop and the lag vehicle accelerates with the maximum acceleration.

Appendix D

Searching Algorithms for Extended Layers

The symbols used in this appendix are shown in Table D.1.

Table D.1: Symbols used in searching algorithms for extended layers

Symbol	Description
$length_l$	length of link l
S_{add}	data structure holding the output (i.e., new segments)
S_{cont}	data structure holding the segments that connect to the next layers, $S_{cont} \subseteq S_{add}$
S_{check}	set of segments that have been checked already
$\hat{v}_l$	speed limit of link l
i	id of the LP which the extended layer to be searched belongs to
j	id of the LP executing the search function ($j \neq i$)
s_{avail}	space available on link l for the new segment
γ_f	front sensing range of agents
γ_b	back sensing range of agents
δ	update interval
ζ	boolean - whether the search should continue

D.1 Function *searchDownstream*

Function *searchDownstream* is shown in Algorithm 9. It takes the following inputs:

- l : the id of the link to be searched
- $start$: the starting displacement of the segment to be searched
- $size$: the size of the segment

The output of the function is the segment starting from $start$ on link l with the size of $size$ in the downstream direction of the link. If the available space is not enough, the size of the segment can be smaller than $size$. The resultant segment is put into S_{add} . If the search continues from the segment to the next layer, it is added to S_{cont} as well. To avoid the area of the segment being searched again, it is also added to S_{check} . If the available space is empty, no segments will be added.

Link l may have three cases. The first case is that link l has already been searched from the other direction (i.e., from the end node towards the upstream direction), as shown in Figure D.1(a). The segment will be restricted to the space available on link l (lines 1 - 8). Let e_{min} be the end displacement of the last segment searched in the upstream direction on link l . Then, the available space will be $e_{min} - start$ (lines 2 and 3). If the available space is larger than 0, the segment will be added to S_{add} and S_{check} (lines 4 - 8). If the available space is even larger than the space required for the layer, it is added to S_{cont} as well (line 5), since search will continue to the next layer in the next iteration.

The second case is that the link is an outgoing boundary link of G_j , as shown in Figure D.1(b). The segment is restricted within the space till the midpoint of the link (lines 9 - 15). If the segment has reached the boundary, the search will be stopped since the link connects to a third partition (line 14). The segments of this layer will not be added to the extended layers.

The third case is that the link is not an outgoing boundary link, nor has it been searched in the upstream direction, as shown in Figure D.1(c). The available space is the rest of the link (line 17). If the available space on the link is sufficient, a segment will be added to S_{add} , S_{cont} , and S_{check} (line 19). Otherwise, the search continues on the links connecting to the end node of link l (lines 20 - 30). The segment on link l is added to S_{add} and

Algorithm 9: Function *searchDownstream*(l , $start$, $size$)

Input: l id of the link to be searched $start$ starting displacement of the segment to be searched on link l $size$ required size of the segment**Result:** segment(s) of the extended layer (S_{add} , S_{cont} , and S_{check} will be updated accordingly)// case 1: link l has already been searched from the end node towards the upstream direction

```
1 if  $\exists seg(id,s,e) \in S_{check} : id = l \wedge e < s$  then
2    $\forall seg(id,s,e) \in S_{check}$  where  $id = l$  and  $e < s$ , assign value  $e_{min} \leftarrow \min(e)$ ;
3    $s_{avail} \leftarrow e_{min} - start$ ;
4   if  $s_{avail} > size$  then
5     put  $seg(l, start, start+size)$  into  $S_{add}$ ,  $S_{cont}$ , and  $S_{check}$ ;
6   else if  $s_{avail} > 0$  then
7     put  $seg(l, start, e_{min})$  into  $S_{add}$  and  $S_{check}$ ;
8   end
// case 2: link  $l$  is an outgoing boundary link
9 else if  $l$  is an outgoing boundary link then
10    $s_{avail} \leftarrow \frac{length_l}{2} - start$ ;
    // whether the space is sufficient for the segment plus a buffer region
11   if  $s_{avail} \geq size + \gamma_f$  then
12     put  $seg(l, start, start+size)$  into  $S_{add}$ ,  $S_{cont}$ , and  $S_{check}$ ;
13   else
14      $\zeta \leftarrow false$ ; // reaching a third LP, stop the search and discard the current layer
15   end
// case 3: neither has link  $l$  been searched from the end node towards the upstream
// direction, nor is it an outgoing link
16 else
17    $s_{avail} \leftarrow length_l - start$ ;
18   if  $s_{avail} \geq size$  then
19     put  $seg(l, start, start+size)$  into  $S_{add}$ ,  $S_{cont}$ , and  $S_{check}$ ;
20   else
    // space left is insufficient, continue searching the connecting links
21   if  $s_{avail} > 0$  then
22     put  $seg(l, start, length_l)$  into  $S_{add}$  and  $S_{check}$ ;
23   end
24   foreach link  $l'$  having the end node of  $l$  as its start node do
25      $searchDownstream(l', 0, size - s_{avail})$ ;
26   end
27   foreach link  $l''$  having the same end node as link  $l$  do
28      $searchUpstream(l'', L_{l''}, \hat{v}_{l''} \cdot \delta)$ ; // segment size is  $\hat{v}_{l''} \cdot \delta$ 
29   end
30 end
31 end
```

Figure D.1: Cases in downstream search on link l : (a) Link l has already been searched from the other direction (i.e., from the end node towards the upstream direction). (b) Link l is an outgoing boundary link of G_j . (c) Link l is not a boundary link, and it has not been searched in the other direction.

S_{check} , but not S_{cont} , because it does not connect to the next layer. For the link that has the end node of link l as its start node (e.g., link l' in Figure D.1(c) or link d in Figure 6.4), function *searchDownstream* is used to continue searching for segment(s) for the current layer. For the link that has the end node of link l as its end node (e.g., link l'' in Figure D.1(c) or link c in Figure 6.4), function *searchUpstream* is used.

D.2 Function *searchUpstream*

Function *searchUpstream* is shown in Algorithm 10. It takes the following inputs:

- l : the id of the link to be searched
- $start$: the starting displacement of the segment to be searched
- $size$: the size of the segment

The result is the segment starting from $start$ on link l with a size of $size$ in the upstream direction of the link. Depending on the available space, the segment size can be smaller. The resulting segment will be put into S_{add} , S_{cont} , and S_{check} according to the condition

of the link being searched. Link l is in one of the following three cases similar to those in function *searchDownstream*:

1. it has already been searched in the other direction (i.e., from the start node towards the downstream direction)
2. it is an incoming boundary link of partition G_j
3. it is not an incoming boundary link, nor has it been searched in the downstream direction

The searching rules are similar to the function *searchDownstream*, except for the third case (lines 16 - 35). If the space in link l is insufficient (see Figure D.2), the search continues only on the links whose traffic flows into link l (lines 20 - 35) because only the traffic on those links affects link l (e.g., link l' in Figure D.2 or link a in Figure 6.4). To fill the remaining space for the current extended layer, the segment size required on such a connecting link is set as $size - s_{avail}$ (line 25). This is because agents on link l' further than a distance of $\hat{v}_{l'} \cdot \delta$ away from link l will not affect link l . If link l' has not yet been searched in the upstream direction (line 26 and Figure D.2(a)), or the space that has been searched in the upstream direction on link l' does not cover the required remaining space (lines 29 - 32 and Figure D.2(b)), upstream search will continue on link l' .

Figure D.2: Case 3 in the upstream search on link l , where the required size covers link l' whose end node is the start of link l . (a) Link l' has not yet been searched previously in the upstream direction. (b) Link l' has already been searched in the upstream direction and the space that has been searched on link l' does not cover the required remaining space (e_{min} is the smallest end displacement of the segments on link l' in the upstream direction in the previous search).

Algorithm 10: Function *searchUpstream*(l , $start$, $size$)

Input: l id of the link to be searched $start$ starting displacement of the segment to be searched on link l $size$ required size of the segment**Result:** segment(s) of the extended layer (S_{add} , S_{cont} , and S_{check} are updated accordingly)// case 1: link l has been searched from the start node towards the downstream direction

```
1 if  $\exists seg(id, s, e) \in S_{check} : id = l \wedge s < e$  then
2    $\forall seg(id, s, e) \in S_{check}$  where  $id = l$  and  $s < e$ , assign value  $e_{max} \leftarrow \max(e)$ ;
3    $s_{avail} \leftarrow start - e_{max}$ ;
4   if  $s_{avail} > size$  then
5     put  $seg(l, start, start - size)$  into  $S_{add}$ ,  $S_{cont}$ , and  $S_{check}$ ;
6   else if  $s_{avail} > 0$  then
7     put  $seg(l, start, e_{max})$  into  $S_{add}$  and  $S_{check}$ ;
8   end
// case 2: link  $l$  is an incoming boundary link
9 else if  $l$  is an incoming boundary link then
10    $s_{avail} \leftarrow start - \frac{length_l}{2}$ ;
11   // whether the space is sufficient for the segment plus a buffer region
12   if  $s_{avail} \geq size + \gamma_b$  then
13     put  $seg(l, start, start - size)$  into  $S_{add}$ ,  $S_{cont}$ , and  $S_{check}$ ;
14   else
15      $\zeta \leftarrow false$ ; // reaching a third LP, stop the search and discard the current layer
16   end
// case 3: neither has link  $l$  been searched from the start node towards the downstream
// direction, nor is it an incoming link
17 else
18    $s_{avail} \leftarrow start$ ;
19   if  $s_{avail} \geq size$  then
20     put  $seg(l, start, start - size)$  into  $S_{add}$ ,  $S_{cont}$ , and  $S_{check}$ ;
21   else
22     // space left is insufficient, continue searching the connecting links
23     if  $s_{avail} > 0$  then
24       put  $seg(l, start, 0)$  into  $S_{add}$  and  $S_{check}$ ;
25     end
26     // only the links whose traffic flows to link  $l$  are searched further
27     foreach link  $l'$  having the start node of  $l$  as its end node do
28       // decide the space required on link  $l'$ 
29        $size \leftarrow size - s_{avail}$ ;
30       // check whether link  $l'$  has already been searched in the upstream direction
31       if  $\nexists seg(id, s, e) \in S_{check} : id = l' \wedge e < s$  then
32          $searchUpstream(l', L_{l'}, size)$ ;
33       else
34          $\forall seg(id, s, e) \in S_{check}$  where  $id = l'$  and  $e < s$ , assign value  $e_{min} \leftarrow \min(e)$ ;
35         // do nothing if the space required on link  $l'$  has been searched before
36         if  $L_{l'} - e_{min} < size$  then
37            $searchUpstream(l', e_{min}, size - (L_{l'} - e_{min}))$ ;
38         end
39       end
40     end
41   end
42 end
```

Appendix E

Publications

The publications from this research are listed below.

- Yadong Xu, Wentong Cai, David Eckhoff, Suraj Nair, Alois Knoll, *A Graph Partitioning Algorithm for Parallel Agent-Based Road Traffic Simulation*, In Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, Singapore, 2017.
- Yadong Xu, Vaisagh Viswanathan, Wentong Cai, *Reducing Synchronization Overhead with Computation Replication in Parallel Agent-Based Road Traffic Simulation*, submitted to IEEE Transactions on Parallel and Distributed Systems, 2017.
- Yadong Xu, Wentong Cai, Heiko Aydt, Michael Lees, Daniel Zehe, *Relaxing Synchronization in Parallel Agent-based Road Traffic Simulation*, In ACM Transactions on Modeling and Computer Simulation, 2017.
- Yadong Xu, Wentong Cai, Heiko Aydt, Michael Lees, Daniel Zehe, *An Asynchronous Synchronisation Strategy for Parallel Large-scale Agent-based Traffic Simulations*, In Proceedings of the 2015 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, London, UK, 2015.
- Yadong Xu, Heiko Aydt, Wentong Cai, Michael Lees, *Efficient Graph-based Dynamic*

Load-balancing for Parallel Large-scale Agent-based Traffic Simulation, In Proceedings of the 2014 Winter Simulation Conference, Savannah, US, 2014.

- Heiko Aydt, Yadong Xu, Michael Lees, Alois Knoll, *A Multi-Threaded Execution Model for the Agent-based SEMSim Traffic Simulation*, In Proceedings of the 13th International Conference on Systems Simulation (AsiaSim), Singapore, 2013.
- Yadong Xu, Heiko Aydt, Michael Lees, *SEMSim: A Distributed Architecture for Multi-scale Traffic Simulation*, In Proceedings of the 26th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation, Zhangjiajie, China, 2012.