博士論文

HPC and feature enhancements of micro- and macroscopic traffic simulators for disaster management applications

(災害対応におけるHPC活用のための微視的・巨視的な交通シミュレータの機能向上)

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF CIVIL ENGINEERING
OF THE GRADUATE SCHOOL OF ENGINEERING,
UNIVERSITY OF TOKYO
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Petprakob Wasuwat
(ペットプラッコ・ワスワット)
September 2018

Acknowledgements

First of all, I would like to sincerely thank Dr. Lalith Wijerathne and Dr. Muneo Hori. They always supply me with logical, useful, sharp, and clear advice. In addition, I would like to thank my doctorate committee member, Dr. Tsutoshi Ichimura, Dr. Eiji Hato, and Dr.Miho Iryo-Asano, for their expert advice and suggestions. Their comments crucially improve my thesis.

I would like to thank my supervisor, Dr. Lalith Wijerathne, for his hard work. I really appreciate the way he taught me to solve many problems during my study. I am very lucky to learn C++ language and Message Passing Interface (MPI) from an expert like him. Learning from his code has been significantly improved my programming skill and logical thinking. I am always surprised every time when he provides me with a solution to the problem which I think it is impossible to solve. This thesis's existence is not possible without his relentless help.

I would like to thank the Ministry of Education, Culture, Sports, Science, and Technology of Japan (MEXT) for giving me a chance to study in one of the best university such as the University of Tokyo. I will apply all the knowledge that I have got from this university to help to develop my country.

Special thanks to my master degree supervisor, Dr. Pruettha Nanakorn; I could not have a chance to apply for the University of Tokyo without his advice and support. He always provides me with kind support even though I always make mistakes.

I would like to extend the gratitude to my lab mates: Ms. Nao Tamechika, Dr. Supun Chamara, Dr. Muhammad Rizwan Riaz, and Dr. Sumet Supprasert. Without the listed lab mates, I think I could not survive the life in Japan during my first year. They provide me with sincere support and always cheers me up especially Dr. Sumet Supprasert. I would like to thank Mr. Takuma Yamaguchi, Mr. Keisuke Katsushima, Mr. Atsushi Yoshiyuki, and Mr. Inawashiro Hiromichi for their kind help with translating Japanese to English and taking random Japanese calls. Also, I would like to thank all lab fellows for their kindness even though we did not have a lot of chances to talk with each other.

I would like to thank Ms. Yuki Takeda who inspires me to come and study in Japan. I have a lot of good memory with her during staying in Hokkaido. We could not have a chance to meet each other since the distance between Tokyo and Sapporo is very far. I wish I could have more chances to visit Sapporo.

Finally, I would like to thank my family; my parents and my sister. They are ready to sacrifice anything for me. We know that we deeply love each other without verbal explanation. I could not achieve anything in life without their existence.

Abstract

In earthquake disaster mitigation, traffic flow simulations have two major applications of significance. The first is to find near-optimal traffic assignment such that the functioning portion of a road network, which is damaged by a major earthquake, can be optimally utilized to minimize economic losses and even plan the recovery sequence of the damaged road segments. The second is high fidelity traffic simulations for virtually mimicking mass evacuation scenarios with the aim of quantitatively evaluating various strategies to accelerate the evacuation processes.

Though both these applications are of significant importance, there lacks HPC capable software to solve real-life problems involving large regions. According to my literature survey, one iteration of optimal traffic assignment such as day-to-day traffic assignment took 1 hour and 30 minutes for New York's road network. At this speed, it could take months to find an acceptable traffic assignment of a large-scale road network, since the optimal traffic assignment may take several thousand iterations to converge to an acceptable traffic assignment. In earthquake disaster mitigation, traffic assignment plans have to be updated at least weekly to best utilize the latest repaired roads. This large gap between the required and the available times-to-solutions emphasizes the need of the development HPC enhanced software to find near-optimal solutions within at least several days period. When it comes to microscopic modeling of emergency evacuations, there are a number of commercial products targetting small-scale applications like fire evacuation of buildings, etc. These cannot be utilized to simulate emergency tsunami evacuations which involve hundreds of thousands of people, on foot and vehicles, over several hundreds of square kilometers. Though a Multi-Agent Simulator (MAS) developed by a group at ERI is capable of such large-scale simulations utilizing HPC, its traffic simulation module lack of essential features like controlling vehicles on multi-lanes and at junctions, and requires further HPC enhancements to scale to larger regions like those affected by Tokai, Tonankai, and Nanaki mega-thrust earthquakes.

The objectives of this research are to develop HPC enhanced systems to address the lack of traffic simulators for the above-mentioned applications by:

1. developing a parallel computing extension for a macroscopic traffic simulator with the aim of finding near-optimal traffic assignments within a reasonably short time
2. implementing lightweight junction model and enhancing the performance of a microscopic agent-based simulator for applications in mass evacuations.

For the first objective, a parallel computing extension was implemented for a macroscopic traffic simulator called FastDUE, which was a software developed by Prof. Iryo (Kobe University) for finding near-optimal traffic assignment based on dynamic user equilibrium [14].

A simple link transmission model [38], which uses asynchronous clocks for vehicles, is used in FastDUE for simulating traffic flow. The developed distributed memory computing extension, based on MPI (Message Passing Interface), consists of five main stages; partitioning the network so that nearly equal workload is assigned to each CPU, calculating shortest time paths for a selected set of vehicles, vehicle loading, forward relay of vehicles, and finally backward relay to update upstream according to traffic congestion. In order to attain high parallel performance, we strived to modify the serial algorithms to minimize the required number of communications and hide communications. The asynchronous clocks in link transmission model make it impossible to predict the amount of computations in the next several time steps. This makes it impossible to assign equal workload to CPUs and is the major barrier to attain high scalability. In order to improve the scalability, it is recommended to replace the current link transmission model with a suitable traffic flow simulator. Since the current link transmission model is a core part of FastDUE, no attempt was made to replace it with a suitable alternative.

Three main improvements are implemented to reduce the computation time of FastDUE. The Distributed memory pathfinding algorithm in the original code was a major bottleneck since path-finding is an inherently serial algorithm. Replacing it with an embarrassingly parallel model, near perfect scalability could be attained for pathfinding, and reduce the computation time for pathfinding by 200 folds. An additional advantage of this embarrassingly parallel pathfinding eliminated the time consuming parallel vehicle loading step. The time asynchronous nature and presence of links without any traffic found to waste CPU cycles in fetching unnecessary link data from main memory. Executing only the active links, not only this large waste of CPU cycles was eliminated but also improved load balance among CPUs. Compared to the small amount of time required to prepare the list of active links, this was found to reduce the time for link transmission model by 20 times. The third improvement is the development of an algorithm for finding a weak ordering for execution to nearly maximizing the vehicle flow rate of the links in each partition, at a given iteration step. Finding optimal execution order of links to maximize vehicle flow rate is an NP-hard problem and impossible for very large networks. However, it is found that certain ordering of links in each partition significantly increases the flow rate, reducing the total execution time up to 40%. According to our literature survey, this is a novel contribution which is applicable to distributed parallel computation of link transmission models in traffic engineering, etc.

In case of microscopic traffic simulator, we have used the time-step driven agent-based system which is developed by a team at ERI, for simulating emergency evacuation. Though it has a reasonable model of pedestrians with high parallel scalability, it has a weak model of vehicles. The main reason for this weak model of cars is its poor model of the environment, which is modeled as a hybrid of a grid and a topological graph. A number of shortcomings in the topological graph introduced several problems like difficulties in define logic to use multilane roads and control vehicles at junctions, errors in pathfinding, etc. In order to address these problems, I developed an automated pre-processor which is capable of generating high-quality grid and graph, and even including other information like contours, parks, water bodies, inundated area, etc. Making use of efficient data structures, the developed pre-processor is fast and robust. The new pre-processor could generate an environment of 600 km^2 region within 1$\sim$2 minutes, while the former pre-processor required hours and was unreliable.

With the improved environment, I implemented logic to mimic vehicle flow at an arbitrary junction, with or without traffic lights. Comparing with real observations from

literature, it is demonstrated that free flow vehicle agents can mimic speed and acceleration profiles observed at a real junction fairly well. In order to resolve conflicts of multiple cars on different trajectories at a junction, a simple and lightweight algorithm based on fear of collision was implemented. The results of the fear-based algorithm for several conflict scenarios are compared with literature. It is shown that the new junction model can mimic drivers' behavior to a reasonable extent, and significantly improved the flow rate compared to the former simple junction model. In addition, the junction model is enhanced to control the traffic with traffic lights.

Further, the parallel computing extension of the MAS is improved by enabling to increase or decrease the number of CPUs in real time. This complex process eliminated the need of aborting the simulations due to a concentration of agents. In order to reduce execution time and improve the accuracy of agents' vision, I implemented a real-time ray tracing algorithm. On Intel CPUs, this reduces the agents' vision by around 40%, which brings it a step closer to simulate evacuate in real time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Major earthquakes, like the impending Tokai, Tonankai and Nankai earthquakes, are predicted to inflict serious damages to the infrastructures of the industrial heartland of Japan and inundate the coasts with destructive tsunamis. Severe damages to road networks will disrupt the industrial supply chain networks and other economic activities triggering a secondary disaster. According to historic earthquakes, the repair time for road networks after major earthquakes can be more than a year; repair time after the 1995 Kobe earthquakes was 21 months ([7]). By optimally utilizing the functioning portions of the road network to meet the traffic demand during this long period, the degree of secondary economic disasters can be minimized or even eliminated. Finding optimal allocation of traffic is a challenging problem, which is well known to be an NP-hard (i.e. algorithms involve non-deterministic polynomial hardness).

Though optimal solution cannot be found with existing algorithms, there exists a number of methods to find near-optimal traffic assignment [29, 6, 3, 28]. However, there are no records in the literature on their applications to a large network, like the Kanto network in Japan. A major challenge in applying those algorithms to large networks is the rapid increase of computational demand with the size of the network. This emphasizes the need of utilizing High-Performance Computing (HPC) in optimal traffic assignment of large networks. The use of HPC is especially necessary after a major earthquake disaster since the traffic allocation has to be updated, at least weekly, to best utilize the latest repaired roads.

Although tsunamis also destroy infrastructures, the large losses of lives are the major concern for coastal cities located in close proximity to subduction zones. Tsunami is expected to arrive at some coastal cities within a few tens of minutes, which is too short, especially for elderly coastal communities, to evacuate to a safe high ground. Allowing to use cars for evacuation is one of the cost-effective means of accelerating evacuation. According to the lessons from the past major tsunamis, the unconstrained use of cars can produce worse outcomes. Extensive studies are necessary to answer questions like, what percentage of people can be allowed to use cars, what are the safe time windows for each region to safely use cars, how to cope with unexpected road damages, how stable is car usage, etc. Such studies will contribute to finding safe strategies to make the evacuation process faster, saving many lives. Considering the number of lives involved and the presence of an unusually large number of pedestrians, which is the recommended mode of evacuation, a mass evacuation simulator with microscopic models of vehicles, pedestrians, and their interactions are required to find answers to questions like the above. Most of the large-scale traffic simulators does not include microscopic models of vehicles and pedestrians, while the existing large-scale agent-based microscopic simulators do not provide junction and lane changing models with accurate speed profiles to model interactions among pedestrians and vehicles.

1.1 Objectives

Motivated by the above-mentioned needs of traffic assignment in disaster recovery, and emergency mass evacuations, this research has two major objectives. The first is to develop a scalable HPC system for finding near-optimal traffic assignment for the large network within a reasonably short time, targeting applications in disaster recovery. The second objective is to enhance an agent-based microscopic traffic simulator for simulating emergency mass evacuation by improving its environment, implementing a junction model capable of reproducing vehicles' speed profiles and autonomously resolving conflicts.

1.2 Contributions

The contributions of this thesis can be summarized as

- HPC enhancement of FastDUE, which is a program for finding near-optimal traffic assignment.
  - Implementing a distributed memory parallel computing extension for FastDUE.
  - Improving pathfinding to reduce computation time, and to reach near ideal scalability.
  - Updating algorithms of FastDUE so that most of the communications can be overlapped with computations, thereby improving the scalability.
  - Enhancements of serial algorithms to significantly reduce computation time
    - Active link partitioning scheme made the traffic simulator run 20 times faster.
    - Re-ordering execution of links to reduce the required number of iterations, leading to 40% reduction of runtime.
- Enhancements of agent-based microscopic traffic simulator
  - Fast and robust pre-processor for generating high-fidelity models of the environment
  - Enhanced behaviors and collision avoidance at intersections
  - 40% reduction of the runtime of agents' vision by replacing a memory-bound algorithm with a computation-bound ray tracing algorithm.
  - Parallel and serial performance improvements

1.3 Thesis structure

The rest of the thesis is organized as follows. The second chapter explains about equilibrium states. Chapter 3 provides details on the first objective including a literature review, details of developed HPC enhanced system for near-optimal traffic assignment, techniques for reducing time-to-solution and improve parallel scalability. The fourth chapter presents details of an agent-based model for mass evacuation simulation, improvements to its environment, the new junction model with accurate speed profiles and conflict resolving, validation of the speed profiles, and parallel computing enhancements. Some concluding remarks are given in the last chapter.

Chapter 2

Equilibrium states of traffic flow

In this chapter, we aim to present the notation and problem definition of user equilibriums.

2.1 Wardrop's Equilibriums

Finding a distribution of route choice in static case is firstly discussed in Wardrop (1952) [36]. To evaluate the performance of future improvement of the road, firstly, we have to do an origin and destination (O-D) survey. Every vehicle from the survey is assumed to only choose the quickest route. Our problem is to find how the vehicles distribute itself to new alternative routes which will be added according to the future improvement plan. Therefore, we have to find a way to find the most efficient distribution of traffic flow so that no single vehicle can unilaterally change its route to reduce the travel cost. This mentioned well-distributed traffic flow is called an equilibrium state.

There are two assumptions which can be used to find Wardrop equilibrium state including:

1. Every vehicle has the same travel time on all selected routes and this common travel time is less than the travel time of any vehicle on the unused route. The equilibrium state under this assumption is called user equilibrium (UE).

2. The average travel time of every vehicle is minimum. The equilibrium state under this assumption is called system optimal (SO).

Wardrop [35] stated that the first assumption is more practical since people always seek for the shortest path. In this case, we will discuss only the first assumption. Suppose the total flow of traffic, Q, is composed of D number of alternative route choices, q_i. The flow, q_i, is a function of travel time , t_i, along the route i. Then:

$$q_i(t_i) \geq 0, i = 1, 2, 3, ..., D$$

and

$$\sum_{i=1}^{D} q_i(t_i) = Q.$$

According to the Wardrop's first principle, $t_1, t_2, ..., t_D$ must be the same value (T). Therefore, the problem is to find a common travel time T. The problem is defined as follows:

$$\sum_{i=1}^{D} q_i(T) = Q.$$

This problem is technically an optimization problem and the solution can be found by using many methods. Obviously, the common travel time cannot be found explicitly. Therefore, an acceptable travel time which produces the smallest error must be found instead. This common travel time (T) can be used to evaluate the performance of the road network. For example, after constructing new roads, the common travel time must be reduced because there are new quicker alternative paths.

2.2 Dynamic user equilibrium

There are many researches that try to formulate dynamic user equilibrium (DUE). Friesz et al. [14] is one of first research that formulates the DUE. According to them, the definition of DUE is the state that, in every instance of time, the travel time on a used path which connects an origin-destination (O-D) pair must be equal and minimal.

The dynamic user equilibrium (DUE) is formulated as follows.

$$f_{k,p}(t)\big[h_{k,p}(f,t) - m_k(t)\big] = 0, \forall k \in O, p \in P_k, t \in [0,T] \tag{2.1}$$

$$m_k(t) = \underset{\mathbf{p} \,\in\, P_k}{min} \big\{h_{k,p}(f,t)\big\}, \forall k \in O, t \in [0,T]$$

The above equations must satisfy following feasibility conditions as follows.

$$\sum f_{k,p}(t) = D_k(t), \forall k \in O, t \in [0,T]$$

$$f_{k,p}(t) \geq 0, \forall k \in O, p \in P_k, t \in [0,T]$$

$f_{k,p}(t)$ is the flow of traffic on path p which connects an O-D pair k at time t. O denotes all O-D pairs. P_k denotes all paths in the domain. t denotes the specific time in time spane $[0,T]$. $h_{k,p}(f,t)$ is the travel time of path p, which connects O-D pair k and is affected by the path flow rates f at time t. $m_k(t)$ denotes the minimum travel time of path p. $D_k(t)$ denotes the demand of O-D pair k at the time t.

The path flow rates f is composed of several flow rate on time span $[0,T]$. It can be written as follows:

$$f = \begin{bmatrix} f_{k_1,p_1}(0) & f_{k_1,p_1}(t_1) & f_{k_1,p1}(t_2) & \dots & f_{k_1,p1}(T)) \\ f_{k_2,p_1}(0) & f_{k_2,p_1}(t_1) & f_{k_2,p1}(t_2) & \dots & f_{k_2,p1}(T)) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f_{k_N,p_1}(0) & f_{k_N,p_1}(t_1) & f_{k_N,p_1}(t_2) & \dots & f_{k_N,p_1}(T)) \\ f_{k_1,p_2}(0) & f_{k_1,p_2}(t_1) & f_{k_1,p2}(t_2) & \dots & f_{k_1,p2}(T)) \\ f_{k_2,p_2}(0) & f_{k_2,p_2}(t_1) & f_{k_2,p2}(t_2) & \dots & f_{k_2,p2}(T)) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f_{k_N,p_2}(0) & f_{k_N,p_2}(t_1) & f_{k_N,p_2}(t_2) & \dots & f_{k_N,p_2}(T)) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f_{k_N,p_N}(0) & f_{k_N,p_N}(t_1) & f_{k_N,p_N}(t_2) & \dots & f_{k_N,p_N}(T)) \end{bmatrix}$$

The physical meaning of feasibility conditions is that the summation of the flow in the whole domain must not exceed the demand and there are only forward flows. According to Eq. 2.1, it can be concluded that there will be no traffic flow on those path p which connect O-D pair k or the travel time in that path is minimum. To find the dynamic user equilibrium state, we must find the path flow rate (f) which satisfy Eq. 2.1 and its feasibility conditions. This is an NP-hard problem. The search space is enormous. Also, the equation 2.1 is not an optimization problem. Therefore, heuristic rules must be set to optimize the path flow rates. In this study, we aim to use a day-to-day traffic assignment to find the near-optimal path flow rates or near-optimal traffic assignment.

Chapter 3

Macroscopic traffic simulator

Damages to the lifeline networks during a major earthquake can bring long lasting disruptions to manufacturing and other economic activities leading to a secondary disaster. Especially when a commercial center like Tokyo is affected by a major earthquake, the resulting secondary disaster can bring serious risk to the nation's economy and even send ripples in the global economy. It is hard to say which lifeline network plays the most critical role since the industries and other economic activities depend on these in a rather complicated manner. Road network is one of the vital element with a significant influence on the economic activities.

Depending on the severity of damages, recovery of road network after a major earthquake can take several months to years of time. As an example, it has taken 21 months to fully recover the road network after the 1995 Kobe earthquake [7]. It is vital to find the means to optimally utilize the functioning portion of a damaged network to meet the traffic demand with minimum traffic delays so that degree of secondary economic disasters is minimized. Such post-disaster traffic assignment plans should be continuously updated according to the progress of recovery of damaged segments.

It is well known that optimal traffic assignment problem is NP-hard problem, hence it is essential to choose a suitable algorithm and develop efficient high-performance computing extensions to find near-optimal solutions for this post-disaster traffic assignment problem. There exists a number of methods to find near-optimal traffic assignment [29, 6, 3, 28]. However, according to our literature survey, none of these methods have been applied to solve large-scale problems like Tokyo probably due to the extensive computational demand. This emphasizes the need of choosing a computationally light algorithm and high-performance computing.

Out of the many methods for near-optimal traffic assignment, in this study, we use the method of day-to-day traffic assignment, which mimics how people find shorter routes by changing their routes according to experiences on previous days. As one would easily guess, this method replans the route of a random subset of vehicles according to travel time information of the previous simulation, and estimate the resulting travel time by simulating the traffic after the route replanning. The above process is repeated until the total delay time reaches a certain convergence criterion. The advantage of this method is that the total computation time can be reduced by using a lightweight macroscopic traffic simulator.

According to our literature survey, there are reported cases of parallel implementations of dynamic traffic assignment [6, 3]. However, these implementations do not scale well (i.e.

reduce the computation time linearly with respect to the number of CPUs used) to solve large-scale problems in a shorter time by utilizing high-performance computer clusters or supercomputers. The current best implementation scales up to 128 CPUs and takes around 1 hour and 30 minutes [31] for a single day iteration. According to our estimations, it would take more than a month to find a solution for New York network. Though several months period is acceptable for regular traffic assignment problems, it is too long for post-disaster recovery problem. For practical applications in post-disaster recovery, the total computation time has to be at least reduce to a week so that traffic assignment plans can be regularly updated with the progress of the recovery of damages. Hence, a better scalable HPC enhanced day-to-day traffic assignment must be implemented for solving post-disaster traffic assignment problems.

In the rest of this chapter is organized as follows. Firstly, we briefly introduce day-to-day traffic assignment. Secondly, we discuss the parallelization of the day-to-day traffic assignment. In addition, we discuss strategies which are used to enhance the parallel algorithms used in our implementation. The next section shows the numerical experiments which are used to demonstrate the efficiency of each strategy. We finally discuss the possible future improvements in the last section.

3.1 Day-to-day traffic assignment

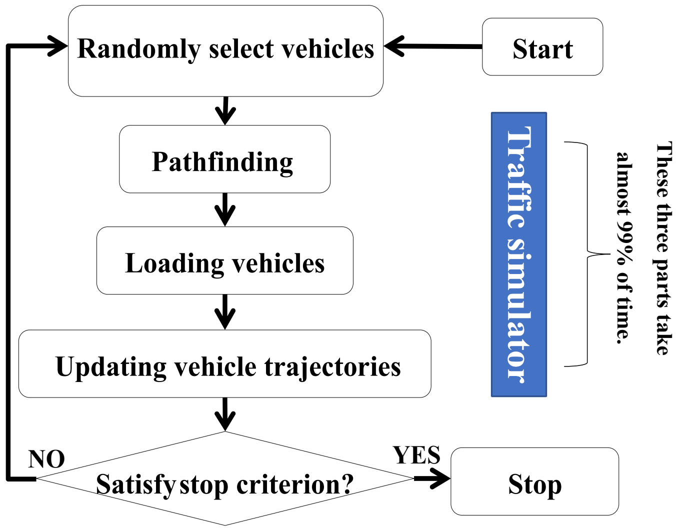


Figure 3.1: An overview of serial day-to-day traffic assignment.

For the sake of completeness, the overview of the day-to-day traffic assignment is briefly explained in this section. Figure 3.1 shows a flowchart of the day-to-day traffic assignment algorithm. It is a daily experience that drivers change their paths according to their previous travel experiences in order to reduce their travel time ([19, 30]). The day-to-day algorithm mimics this drivers' behavior to find a near-optimal traffic assignment. Numerical instabilities of iterative oscillation may be observed.

As above described, day-to-day traffic assignment involves traffic flow simulations, and it usually requires a large number of traffic flow simulations to obtain a converged solution by mimicking drivers' behavior. Therefore, a lightweight stochastic cell transmission traffic simulator is used in this study as a strategy to lower the total computation time. In stochastic cell transmission simulators, each road segment stores information of every passing through vehicles. Each element in the links' vehicle list stores information pertaining to a vehicle such as a vehicle id, entry time to the link and departure time from the link. The main 5 steps of day-to-day traffic assignment are briefly explained below.

Randomly select a set of vehicles: If the all the vehicles are not yet included, introduce a certain percentage of vehicles from the input origin-destination pairs (OD pairs). Also, choose a random subset of the vehicles already introduced to the system.

Pathfinding: Find the paths with minimum travel time, based on the previous travel time estimated with step 4 of previous iterations, for the set of vehicles selected in the above step.

Loading vehicles: The randomly selected vehicles in step 1 are assigned to the vehicle lists of the links along the corresponding paths. The entry time and the departure time of newly assigned vehicles are set to infinity.

Updating vehicle trajectory: Simulate traffic flow using a suitable link delay model for handling congestions. Store the travel time information of each link at suitable time interval so that travel time can be estimated in step 2 of next day iteration.

Stop criterion: Evaluate the total delay, and decide to terminate iterations or return to step 1 and continue iterations based on a suitable criterion.

The day-to-day traffic assignment's result depends on random parameters like in which order the vehicles are introduced, etc. Depending on these random parameters we may arrive at different solutions. It is best to conduct many simulations with different random parameters and choose a suitable stable solution. The stability of the solutions can be tested in several ways. One prominent way is to give a random variation, within a practical time range, to the departure time of the vehicles and estimate the resulting traffic condition by simulating traffic flow. For this simulation for stability testing, it is better to use a microscopic traffic simulator so that most of the affecting factors can be taken into account. A criterion like a percentage increase of delay time can be used to test whether the traffic condition is not significantly affected by the random variations.

In this study, we used the FastDUE code, which was developed by Prof Takamasa Iryo (Kobe University), as the base day-to-day traffic simulator. All our HPC enhancements are implemented in FastDUE code. In this study, we did not explore the possibilities to enhance the performance of the serial FastDUE code, though some potential advantages improvements, like modifications to data structures, were identified.

3.2 HPC enhancement

A short introduction to the main steps of the developed distributed memory parallel day-to-day algorithm with link transmission model (LTM) [38]. is presented in this section. The workload is distributed among CPU cores of a cluster of computers connected via a dedicated high-speed network. The necessary information to make the CPUs to collectively solve the problem are exchanged among the CPU cores using Message Passing Interface

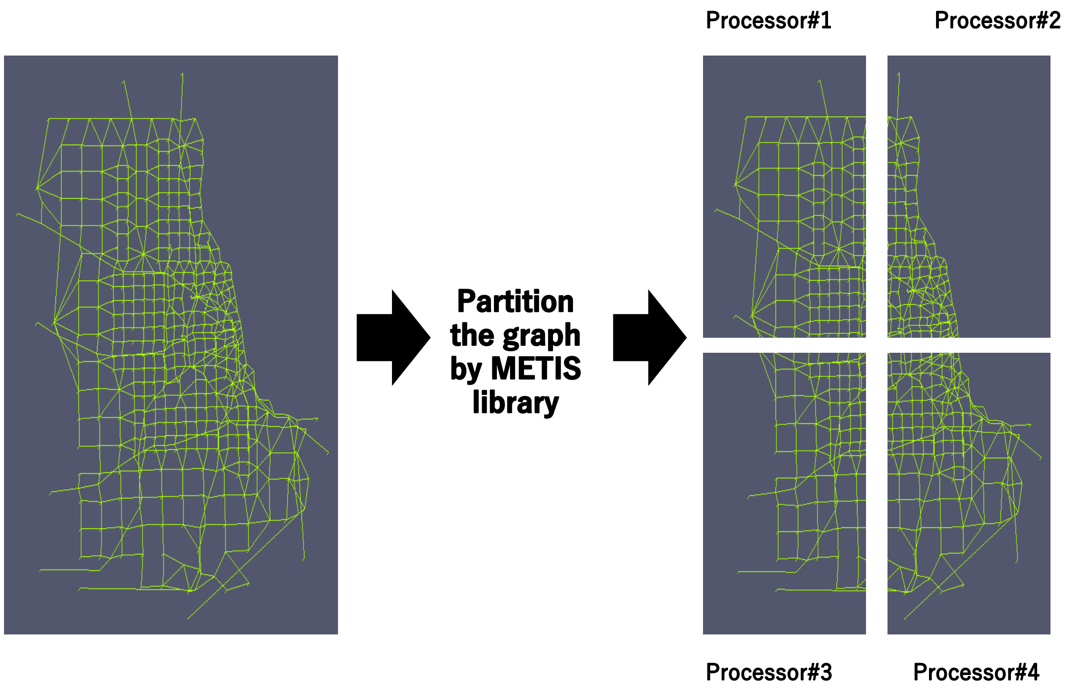


Figure 3.2: An example of decomposed Chicago road network.

(MPI). The basic processing unit of our current implementation is a CPU core since we use only MPI. The terms CPU core and MPI process are used interchangeably in the rest of the thesis.

3.3 Domain decomposition

In order to distribute the computational workload of Day-To-Day Traffic Assignment (D2DTA), the bi-directional graph of the traffic network is decomposed into non-overlapping and continuous sub-networks using the graph partitioning software METIS ([21]) as shown in Fig.

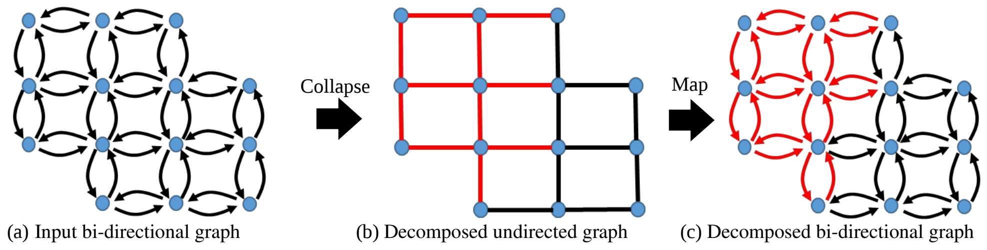

Collapse
Map
(a) Input bi-directional graph
(b) Decomposed undirected graph
(c) Decomposed bi-directional graph

Figure 3.3: Main steps of generating non-overlapped partitions of a bidirected graph.

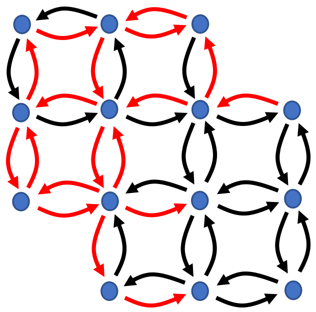


Figure 3.4: Overlapped partitions

3.2. The time asynchronous nature of LTM (i.e. clocks of vehicles are not synchronized) makes it difficult to make a reasonable estimation of the amount of computations in each partition, within next several iterations. Devoid of better solutions, We use link-based partitioning scheme with a number of passing vehicles as weight.

Each of the partitions is assigned to a separate MPI process (i.e. CPU core) so that computation power of multiple MPI processes can be utilized to solve the problem in a shorter time. In this particular MPI-only settings we use, even if some of the CPU cores are located on the same motherboard sharing all the resources, each CPU core behaves as an independent MPI process with its own private memory space and resources. Therefore, to maintain the continuity of the original problem, each MPI process should exchange state of the upstream links located along partition boundary, which deliver vehicles to neighbor partitions, sending explicit messages to the owner MPI processes of its neighbor partitions. This extra overhead of communication between partitions, which does not exist in serial version, significantly increases with the number of MPI processes involved, and hence has to be minimized to efficiently utilize a larger number of CPUs to solve the problem in shorter time. Communication hiding (i.e. doing some useful computation while communication is progressing) is the main strategy we utilize to reduce this extra communication overhead.

Figure 3.3 illustrates the main steps involved in partitioning the bidirected graph. It is highly desirable to use non-overlapped partitions since it enables independent traffic flow simulations within each MPI process, simple and efficient management of message passing among MPI processes, and attaining high parallel scalability by communication hiding. In order to obtained non-overlapped partitions, first the bidirected graph is collapsed into an undirected graph; the sum of the number of vehicles in each pair of up and down links are assigned to the corresponding link of the undirected graph as the estimator of the amount of computations. Partitioning the undirected graph with METIS and mapping the results back to the original bi-directional graph, we can obtain desired non-overlapped partitions (see Fig. 3.3(b) and Fig. 3.3(c)).

Direct partitioning the bidirected graph with METIS produces overlapped partitions (i.e. up and down links between some pairs of nodes belong to different MPI processes) as shown in Fig. 3.4. Such overlapping introduces significantly a large number of in/out flow of traffic scattered all over the domain of the partition, making the traffic flow simulations within a partition to be heavily dependent on the neighbor partitions. On the other hand, in the non-overlapping case, in/out flow of traffic to/from a partition happens only through the links along the partition boundary, making traffic flow simulations within the partition

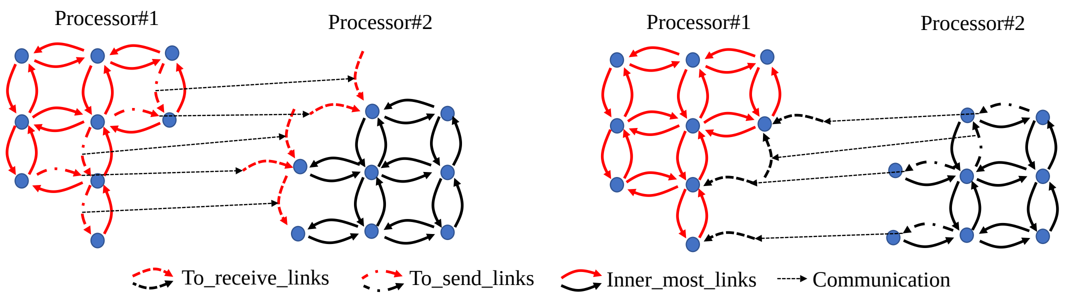


Figure 3.5: Grouping of links for communication hiding and efficient communication management. At each iteration, the state of ghost copies, shown in dashed lines, are updated using MPI according to the state of the corresponding links in the neighbor partitions.

to be independent of neighbor partitions. This greater independence from the neighbor partitions provides better opportunities to attain high scalability and manage communications efficiently. In a nutshell, overlapped partitions significantly lower both the computational efficiency and parallel scalability, and complicates the management of MPI communications.

As above mentioned, to maintain the continuity, each MPI process should exchange state of the upstream links located along partition boundary, which deliver vehicles to neighbor partitions, to the owner MPI processes of its neighbor partitions. In order to efficiently manage communication of the state of those upstream links, we include their dummy copies along the boundary of the neighbor partitions according to the corresponding downstream links. In each partition, the dummy upstream links are grouped into a set called *to_recv_links*. Further, the upstream links along partition boundaries, whose states are to be sent to the neighbor partitions, are grouped to a set called *to_send_links*. The set *innermost_links* is formed by subtracting the *to_send_links* from the links of a partition. Fig. 3.5 illustrates these three subsets of links in a partition.

Overlapping the communications with the computations is one of the widely used strategy in attaining higher scalability in distributed parallel computing. Doing some useful computations while communications are progressing, the time spent on communications can be reduced or even completely eliminated. This often produces higher parallel scalability (i.e. two fold increase of the number of MPI processes exactly or nearly halves the computation time). The above sub-grouping of links in a partition makes it possible to efficiently manage not only the communications, but also overlapping of communications with computations. In order to hide communications of the traffic flow computations, first the *to_send_links* are executed, and the messages carrying their updated states to the corresponding neighbor partitions are posted using the non-blocking MPI functions *MPI_Isend*() and *MPI_Irecv*(). Next, the traffic flow of the *innermost_links* is executed, and finally the above-posted messages are finalized with *MPI_Waitall*() or *MPI_Testall*(). Provided that a number of links in subset *innermost_links* is large, this overlapping of computations and communications can significantly reduce the communication overhead, thereby increasing the parallel scalability.

Though not essential, contiguous partitions are desired since it contributes to increasing the effectiveness of communication hiding. Compared to non-contiguous partitions, contiguous partitions increases the ratio between the number of links inside a partition to the number of links along the partition boundary, thereby increasing the number of calculations to be performed relative to the amount of data to be communicated, thus providing long enough time to complete overlapped communications.

3.4 Enhancements of parallel performance

The computation time of a perfect scalable parallel program halves when the number of CPUs doubles. Attaining high scalability is a challenging task and requires significant modifications to the data structures and serial algorithms, or it may even require the development of completely new algorithms. This section presents three strategies, apart from above presented communication hiding, to improve the computational efficiency of distributed D2DTA.

3.4.1 Pathfinding

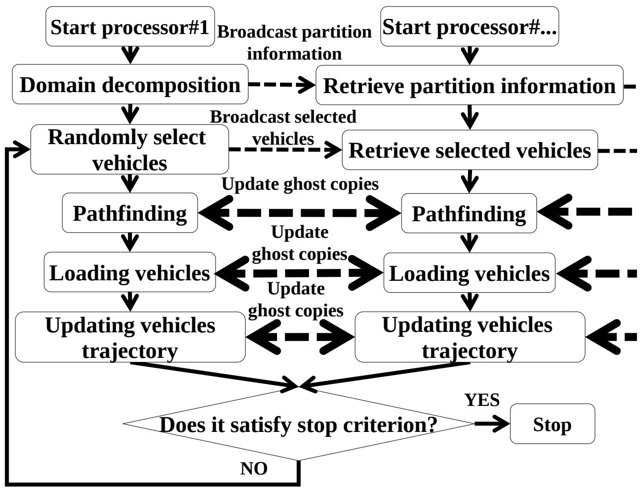
Pathfinding is one of the most time-consuming components of D2DTA. FastDUE uses the Dijkstra's algorithm ([11]), which is widely used for pathfinding. Due to its inherent serial nature, it is difficult to implement scalable distributed or shared memory parallel versions of Dijkstra's algorithm. A shared-memory parallel implementation by Jasika et al. ([20]) has attained only 10% speed up, which is fairly low. None of the distributed-memory parallel implementations ([17];[18];[6]) have reported reasonable scalability. The main difficulty in distributed parallel implementation is involvement of a large number of messages and difficulties in assigning balanced workloads to MPI processes.

We developed two versions of parallel D2DTA; one with distributed memory parallel Dijkstra's algorithm similar to Haribar et al.'s work ([17]) as shown in Fig. 3.7, and the other with the serial Dijkstra's algorithm. Figure 3.6 illustrates details of the MPI communications involved in both the implementations.

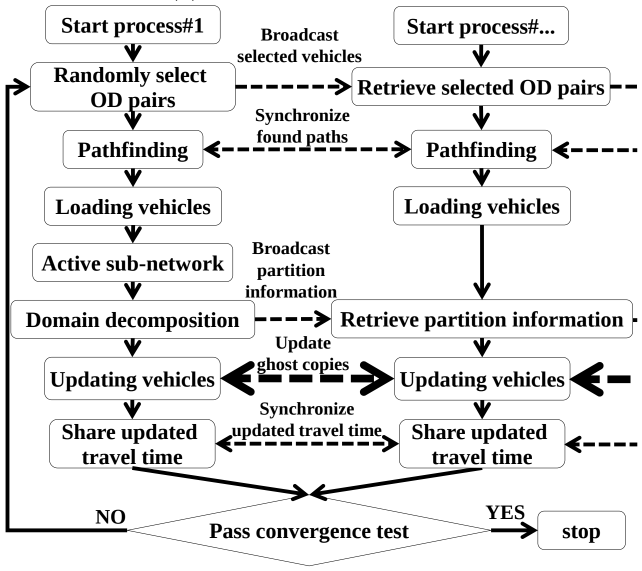
In the parallel pathfinding, each processor finds the path for the assigned vehicle until vehicles reach its boundary then these vehicles will be passed to neighbor processors as shown in Fig. 3.7. The advantage of this method is each processor does not have to keep the whole network. Therefore, the code is not memory intensive so that it can handle large scale problem without exceeding memory of a node in a computer cluster. The disadvantage is it requires a lot of communication. The approximated number of communication messages could be estimated as *number of MPI processes* $\times$ *number of nodes along partition boundaries* $\times$ *number of vehicles*, which can be quite large.

In the serial implementation of pathfinding, each MPI process keeps a copy of the whole network and travel time information of each link and calculates paths for a given subset of vehicles as shown in Fig. 3.8. We assigned an equal number of vehicles to each MPI process so that each MPI process will have nearly the same computational workload. In the example (Fig. 3.8), we assign 1 vehicle to each processor. Instead of assigning the equal number of vehicles, using the spatial distance between the origin and destination of each vehicle, or previous measurements of pathfinding time will lead to better scalability. This serial implementation of pathfinding belongs to an embarrassingly parallel category, since finding a path for a vehicle does not require any MPI communications.

The main disadvantage of the embarrassingly parallel approach is the need to keep of copy of complete network in each MPI process. This is not a major problem since modern computing nodes have a large memory. Though the serial implementation requires each MPI process to exchange the travel time data of the previous iterations and the paths found


Start processor#1
Broadcast partition information
Start processor#...
Domain decomposition
Retrieve partition information
Broadcast selected vehicles
Randomly select vehicles
Retrieve selected vehicles
Update ghost copies
Pathfinding
Pathfinding
Update ghost copies
Loading vehicles
Loading vehicles
Update ghost copies
Updating vehicles trajectory
Updating vehicles trajectory
YES
Does it satisfy stop criterion?
Stop
NO

(a) With parallel path finding.



(b) With embarrassingly parallel path finding and active sub-network.

Figure 3.6: Flowcharts of two implementations of parallel day-to-day traffic assignment. Dashed lines represent MPI communications; thickness and arrows indicate volume and direction of data flow, respectively.

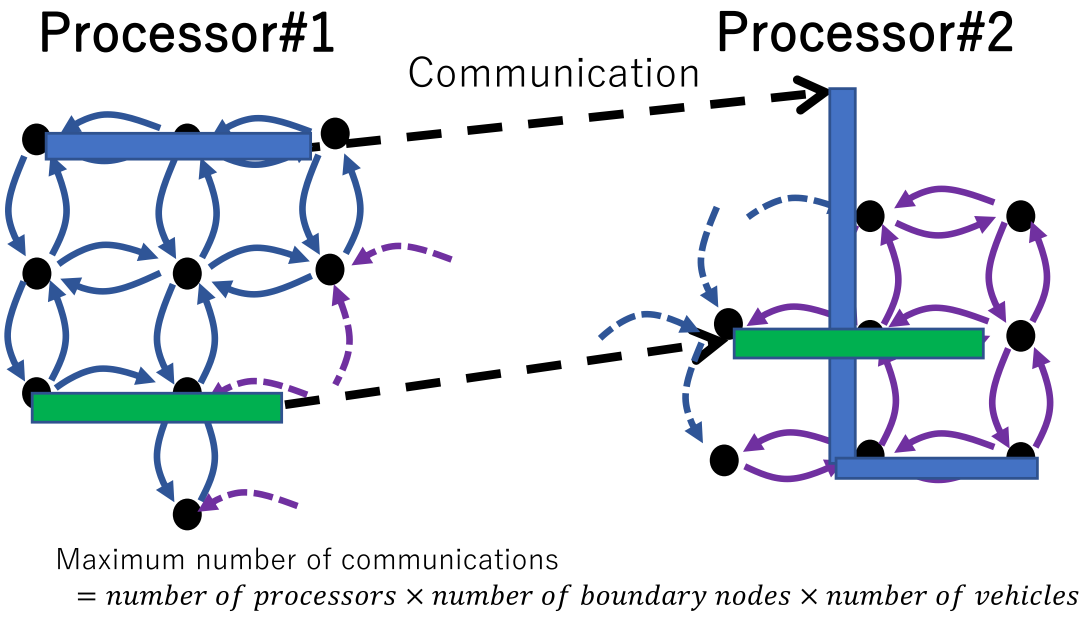

Processor#1
Processor#2
Communication
Maximum number of communications
= number of processors × number of boundary nodes × number of vehicles

Figure 3.7: Distributed parallel pathfinding.

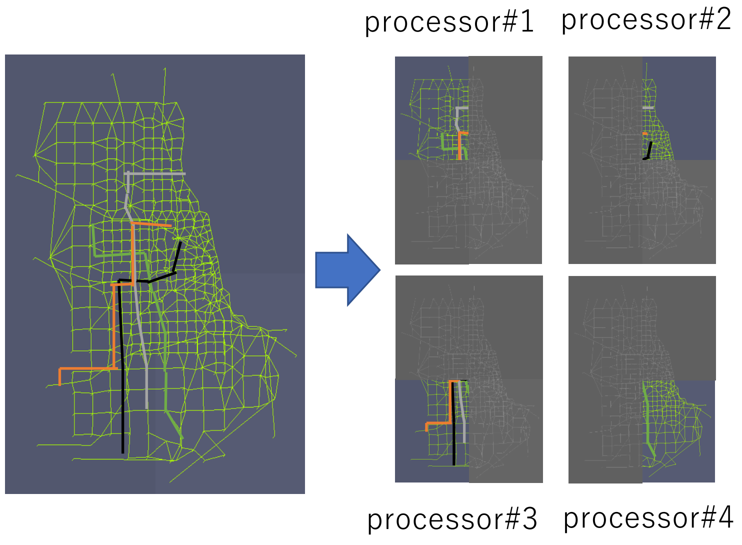
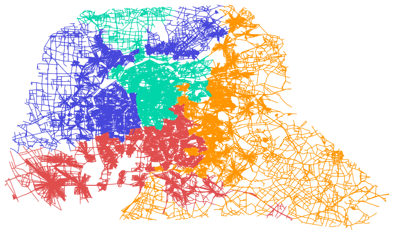


Figure 3.8: Serial pathfinding.


(a) Complete traffic network.

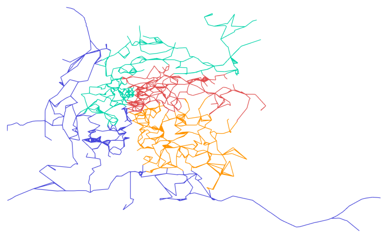

(b) The active links during the early iterations.

Figure 3.9: Decomposed Nagoya network for 4 MPI processors (Colors indicate partitions)

with the rest of MPI Processes, this communication overhead is orders of magnitudes smaller than that is required by the distributed memory parallel pathfinding. Further, synchronizing the paths among all the MPI processes make it possible to completely eliminate the large number of communications involved in vehicle loading step of parallel pathfinding. This large reduction of communication overhead and better load balance make the embarrassingly parallel approach to drastically improve the scalability of pathfinding and vehicle loading steps, compared to those with parallel pathfinding.

3.4.2 Processing only the active sub-network

Significantly small number of links are active (i.e. vehicles passes through) at the early stages of D2DTA since D2DTA algorithms gradually introduce vehicles to the simulation. For the sake of brevity, let's call this small subset of links *active links.* Figure 3.9(a) and Fig. 3.9(b) show the full Nagoya network and active links in an early iteration of D2DTA. Instead of simulating the whole network, simulating only the active network provides two advantages. First is the large reduction of total computation time by eliminating the number of unproductive access to the main memory. Even though the inactive links do not incur any computational overhead, processing the whole network requires data of each link to be brought to CPU from main memory, which is a significant time-consuming operation. The second is poor quality partitions. As mentioned in section 3.3, we use the number of vehicles in each link as the weight in partitioning. The presence of links with zero vehicles leads to seriously low-quality partitions with high workload imbalance.

The active links can be easily identified right after the vehicle loading stage. The active sub-network is created by picking only active links from a full network. Instead of simulating the full network, the active sub-network is partitioned and used in updating vehicle algorithms. According to Fig. 3.6(b), after domain decomposition scheme, only updating vehicle trajectory algorithm uses the partition information. Hence, the link-based partition is used to equally distribute the workload for accelerating the updating vehicle trajectory algorithm. The result of active sub-network domain decomposition is illustrated in Fig. 3.9(b).

If only the active sub-network is simulated, it will not only eliminate time wasted on unproductive loading data of inactive links from main memory but also generate well load-balanced partitions leading to higher parallel efficiency. As it will be demonstrated in the next section, there is a considerable gain in simulating only the active sub-network. Figure 3.6(b) illustrates the flowchart of the developed parallel D2DTA algorithm with active sub-network.

3.4.3 Processing links in the direction of vehicle flow

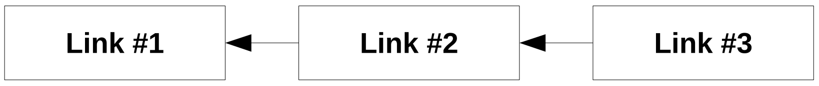


Figure 3.10: A simple 1D flow of vehicles

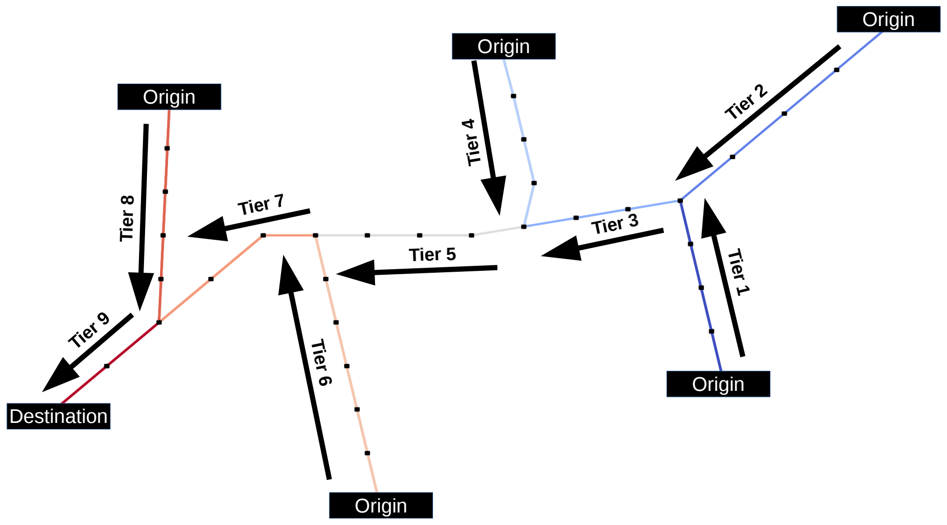


Figure 3.11: The multiple source one destination problem

The number of iterations requires for LTM based traffic simulator depends on the execution order of the links, though ideally, the total number of updates of variables remains roughly the same. As an example, consider a unidirectional flow of n number of vehicles starting from the origin end of a single straight road consisting of N number of links (see Fig. 3.10). LTM based traffic simulator involves only one iteration if the links are processed from origin to destination. However, if the links are processed in the destination to origin direction, it requires N number of iterations. Though some of these iterations in processing destination to origin direction does not involve any vehicle transmission among links, it still incurs a non-negligible overhead wasting CPU cycles in loading links' data from main memory, going through all the vehicles entries in a link to identify no vehicles are there to be transmitted, message passing, etc. On the other hand, processing in the origin to destination direction does not involve such overhead due to useless iterations and transmits the vehicles.

To further demonstrate the advantages of processing links in flow direction, consider the multiple source single destination problem shown in Fig. 3.11 with 250,000 vehicles from each origin. We compared execution of links in a random order with the tier orders of execution shown in Fig. 3.10. The time asynchronous link transmission model we use produces a perfect vehicle transmission among links, delivering all the vehicles to their destinations in a single iteration, when links are executed in the tier order shown in Fig. 3.10. Compared to random order of execution of links, this is almost 10 times faster.

Though it is straightforward to identify in the above two simple examples, no such ideal execution order which complete traffic flow simulation in one iteration exists for real traffic networks. Given a link A, executing all the upstream links which deliver vehicles to link A, is one strategy to find an optimal links execution order. Repeatedly apply the above simple rule, it may be possible to find an optimal execution order. However, on real networks with loops and complicated traffic flow, this develops into a challenging graph optimization problem.

Instead of solving an optimization problem, we opt for finding a weak link execution order with higher vehicle transfer rate per iteration. Though not optimal, a link execution order which substantially increases the vehicle transfer rate at each iteration of LTM can also significantly reduce the number of iterations. In each partition assigned to an MPI process, the major vehicle flow is from *to_recv* links to *to_send* links via the *inner_most* links, which makes *to_recv* and *to_send* links virtual origins and destinations in each partition. Based on this observation, we propose a weak link execution order which significantly increases the vehicle transfer rate per iteration. The pseudo-code of the algorithm to find this weak link execution order is given in Algorithm 6.1.

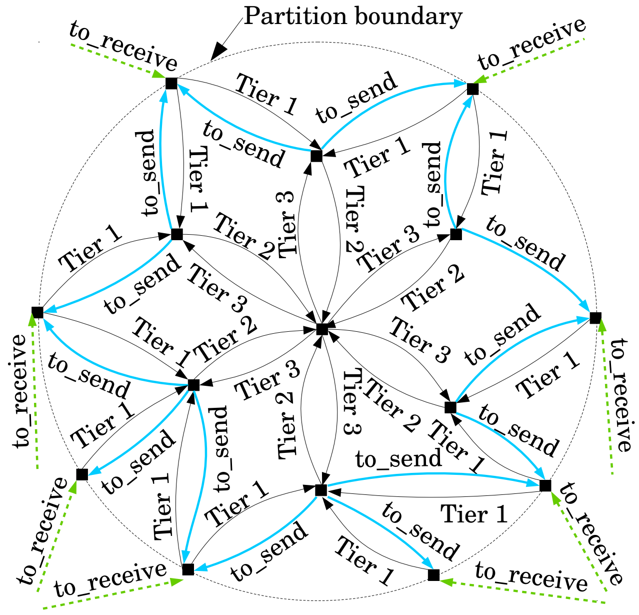


Figure 3.12: Tier ordering of links according to vehicle flow directions.

In the Algorithm 6.1, starting from the *to_receive_links*, we traverse the network in a breadth-first order, tagging immediate downstream links of *to_receive_links* as *tier-1*, their immediate downstream as *tier-2* and so on until *to_send_links* are reached (see Fig. 3.12). Each link is traversed only once during this process. While this tagging process takes a short time, processing the links in the created tier-lists (i.e. *to_receive_links*, *tier-1*, *tier-2*, . . ., *to_send_links*) significantly reduces the required number of LTM iterations. As demonstrated

Algorithm 3.1: Reordering execution order of innermost links

```
input  : Partition information (indexs of to-receive links, to-send links, and
         innermost links) and road network
output: Execution order array
1  create vector $EOA$;                    /* vector of execution order array */
2  create set $TMPS$;                             /* set of temporary links */
3  create set $LSL$;                             /* set of last set of links */
4  foreach link L in to-receive links do
5  |   LSL.insert($L$);
6  end
7  initialize int $LSize$ = -1;                      /* size of the last set */
8  while sizeof(EOS) ¡ sizeof(innermost links) do
9  |   foreach link L in LSL do
10 |   |   foreach link OL in outgoing links of L do
11 |   |   |   if $OL$ is not a bidirectional link of $L$ then
12 |   |   |   |   if $OL$ belongs to the partition then
13 |   |   |   |   |   if $OL$ is not in to-send links group then
14 |   |   |   |   |   |   $TMPS$.insert($OL$);
15 |   |   |   |   |   end
16 |   |   |   |   end
17 |   |   |   end
18 |   |   end
19 |   end
20 |   $LSL = TMPS$;
21 |   $EOA$.append($TMPS$);
22 |   $TMPS$.clear();
23 |   if sizeof(EOA) != LSize then
24 |   |   $LSize$ = sizeof($EOA$);
25 |   else
26 |   |   break;
27 |   end
28 end
29 if sizeof(EOA) != sizeof(innermost links) then
30 |   create set $SDiff$;     /* difference of sets innermost links and $EOA$ */
31 |   $SDiff$ = Innermost links$\setminus EOA$; /* $\{ x \in$ Innermost links | $x \notin EOA \}$ */
32 |   $EOA$.append(SetDiff);
33 end
34 return $EOA$
```

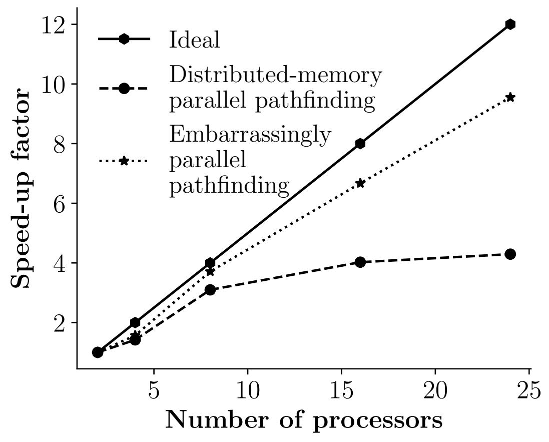


Figure 3.13: Scalability of the two pathfinding algorithms.

in the next section, it can produce around 30% reduction of runtime compared to the execution of links in a random order (e.g. in the ascending order of link ID's).

3.5 Numerical experiments

In this section, we demonstrate the effectiveness of the parallel performance improving strategies presented in the previous section. We used the Nagoya network consisting of 152,464 links and 37,511 nodes as shown in Fig. 3.9(a), for the simulations presented here.

3.5.1 Distributed-memory parallel pathfinding versus embarrassingly parallel pathfinding

Two experiments are conducted to evaluate advantages of the embarrassingly parallel pathfinding, which is explained in section 3.4.1, comparing with the distributed-memory parallel Dijkstra's algorithm by [17].

For the sake of completeness, first the distributed-memory parallel Dijkstra's algorithm9) is briefly summarized as follows:

1. Place all vehicles' source in local subnetwork into local queue
2. Find the shortest path for all source in local queue
3. Send updated boundary node labels to neighbors
4. Receive boundary node labels from neighbors and place in queue
5. Perform global operation for termination detection. If the algorithm is not terminated, then repeat step 2 to step 4

In the first experiment, 5,000 vehicles with random origin-destination pairs (OD pairs) are simulated in a workstation consisting of two Intel Xeon CPUs (E5-2697 v2 @ 2.70 GHz)

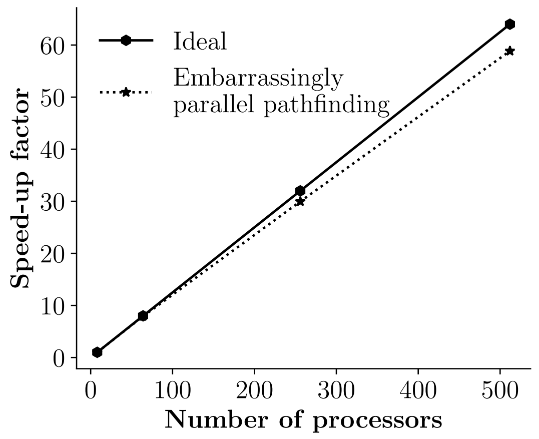


Figure 3.14: Scalability of embarrassingly parallel pathfinding; a large-scale problem.

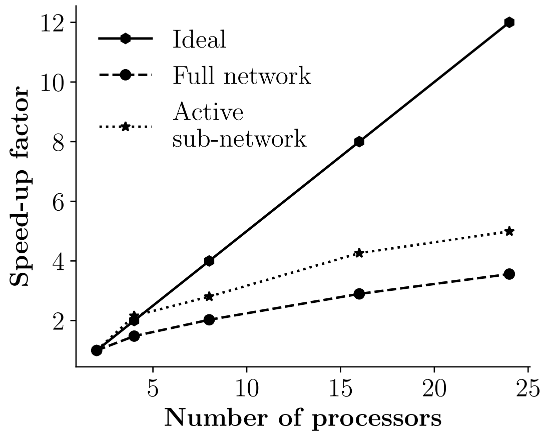


Figure 3.15: Scalability improvement brought by processing active sub-network.

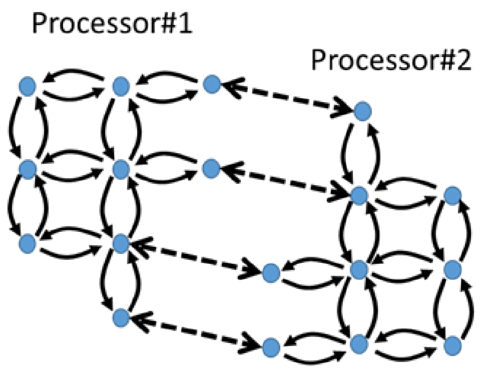


Figure 3.16: Communication pattern in a distributed-memory parallel pathfinding.

and 256 GB memory. As shown in Fig. 3.13, the embarrassingly parallel algorithm produces significant improvement in scalability. Most importantly, as shown in Table 3.1, there is nearly two orders of magnitudes reduction of pathfinding time. In the second experiment, scalability of the embarrassingly parallel pathfinding is tested with 500,000 vehicles in the K-computer (AICS, Kobe, Japan). According to Fig. 3.14, near ideal scalability can be attained with this embarrassingly parallel approach. These tests demonstrate that the embarrassingly parallel pathfinding does not only have higher scalability but also requires significantly low computation time.

The reason for the poor scalability of a distributed-memory parallel path finding is the involvement of a large number of communication with an unpredictable pattern of destinations. The example of communication pattern of the distributed-memory parallel path finding is illustrated in Fig. **??**. The bidirectional dotted-arrow represent the communication messages. Each partition can exchange the updated node labels with their neighbors in each iteration. In some iteration, it is possible that there are inactive partitions. Therefore, the communication pattern is not predictable. To explain why it involves an unpredictable number of communications, refer Fig. **??**. Here, each partition is represented by different colors, and we consider only one vehicle with the path shown in red color. In this particular case, it needs at least 5 iterations to finish, and the exact number of iterations may be larger. In the practical problem, there are more partitions and much more vehicles leading to an unpredictable number of communication.

3.5.2 Processing the full network versus the active sub-network

In this section, the effectiveness of processing only the active sub-network is compared with the full network. Randomly selected 200,000 vehicles are used. The full road network (Fig. 3.9(a)) consists of 152,464 links and 37,511 nodes, while the active sub-network (Fig. 3.9(b)) consists of 3,943 links and 2,020 nodes.

According to Fig. 3.15, processing only the active sub-network produces only a slight scalability improvement. However, Table. 3.2 shows that simulating traffic flow on the active sub-network is around 30 times faster than that with the full network. This indicates that traversing through inactive links, CPUs waste a significant amount of time unproductively accessing main memory.

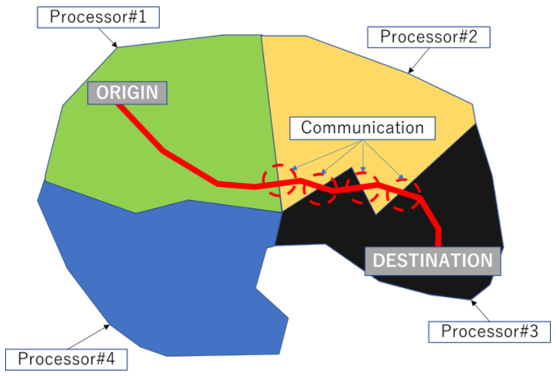


Figure 3.17: Unpredictable number of communications in distributed-memory parallel pathfinding.

3.5.3 Processing links in the direction of vehicle flow

In order to study the effectiveness of processing links in traffic flow direction, which was presented in section 3.4.3, two sets of traffic flow simulations were conducted with randomly selected 1,000,000 vehicles. The links were updated in the ascending order of their ID's (i.e. a random order) in the first set of simulations, while, in the second set, the links were updated according to the tier-lists presented in section 3.4.3. Only the active links of the network are considered in these simulations, and time to complete traffic flow with LTM is measured. According to Fig. 3.18, processing the links in the flow direction based tier list and random order have identical poor scalability.

Though tier-ordered list produces no improvement in scalability, it significantly reduces the number of iterations required for simulating the traffic flow. As shown in Fig. 3.19, the flow rate (i.e. the number of updated vehicles during an iteration) is significantly high in the tier-ordered processing of links. This high vehicle flow rate reduces the required number of iterations for traffic flow simulation to 28, while the unordered processing of links requires 70 iterations. Further, Table 3.4 shows that tier-ordered processing of links brings about 30% reduction of time required for simulating the traffic flow with LTM.

It is quite mysterious that the computational time is reduced even though there is no improvement of scalability. According to 3.19, we can observe that the area under two curves is almost the same. The flow rate in Fig. 3.19 is the global summation of each processor's flow rate. Therefore, it would be clearer to plot the flow rate of each processor (Fig. 3.20). In parallel computing, the computational time is mainly wasted for communications between processors. All processors except one processor that has the highest workload (flow rate) have to wait for synchronization of data at the end of each iteration. For example, the red lines in Fig. 3.20 represent each processor's flow rate of traffic simulator with unordered execution order. It can be observed that there is a big gap between each processor. It implies that processors waste a lot of CPU cycles for idling. Since the workload of link transmission model based traffic simulator is not predictable. Therefore, we could not easily find the good partitioning scheme to reduce this gap if the traffic simulator is not redesigned. Since we find the simple and effective order of execution to reduces the number of iterations which is needed to converge the macroscopic traffic simulator then the total wasted time can be

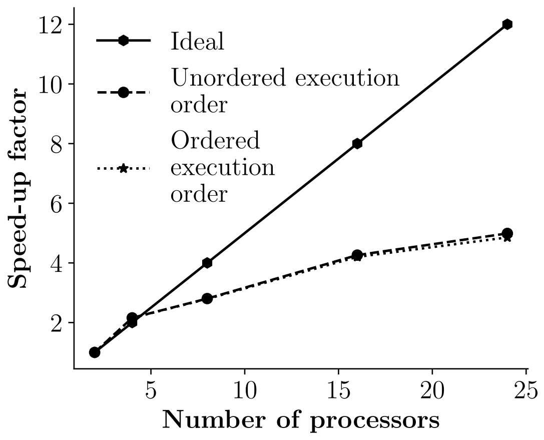


Figure 3.18: Scalability of traffic flow simulations with tier-ordered, and unordered processing of links.

eliminated. According to Fig. 3.20, the big gap still appear in case of ordered execution (black lines). However, the number of iterations is reduced significantly (40 iterations). This smaller number of iterations leads to the reduction of the runtime.

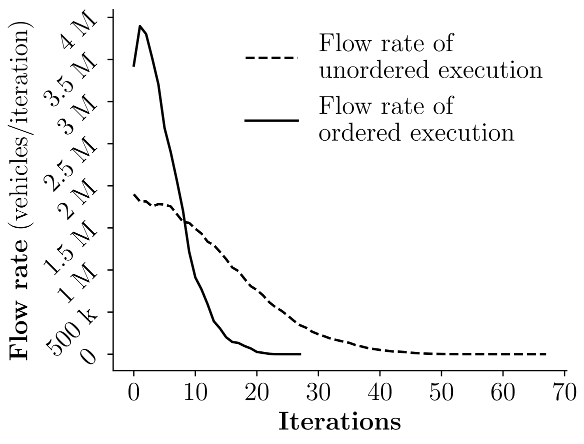


Figure 3.19: Vehicle flow rates with tier-ordered, and unordered processing of links.

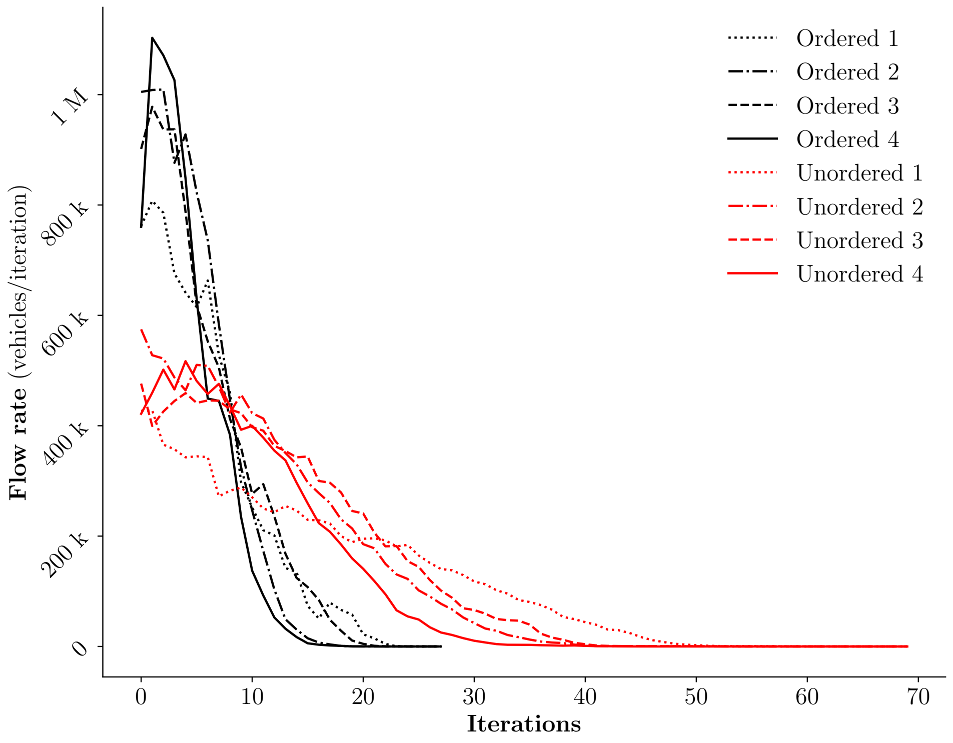


Figure 3.20: Vehicle flow rates of each MPI processors

MPI processors	T_1 (s)	T_2 (s)	T_1/T_2
4	6,429.58	25.24	254.764
8	2,949.18	10.68	276.14
16	2,269.82	5.94312	381.924
24	2,127.69	4.15	512.969

Table 3.1: Execution time of distributed-memory (T_1), and embarrassingly: parallel (T_2) pathfinding.

MPI processors	T_1 (s)	T_2 (s)	T_1/T_2
4	61,225.2	2,171.71	28.1921
8	44,702.7	1,671.18	26.75
16	31299.2	1,099.71	28.46
24	25452.2	939.583	27.09

Table 3.2: Execution time of updating vehicle trajectory with full network, T_1, and active sub-network, T_2.

MPI processors	T_1 (s)
8	9,135.34
64	6,429.58
256	2,949.18
512	2,269.82

Table 3.3: Execution time of embarrassingly parallel pathfinding for large scale problem, T_1.

MPI processors	T_1 (s)	T_2 (s)	$((T_1 - T_2) \times 100)/T_1$
2	4,687.23	2,769.81	40.9%
4	2,171.71	1,422.4	34.5%
8	1,671.18	1,175.33	29.7%
16	1,099.71	903.52	17.8%
24	939.58	682.5	27.3%

Table 3.4: Time required for traffic flow of one million vehicles with unordered processing of links, T_1, and tier-ordered processing of links, T_2.

Chapter 4

Issues for future improvements in macroscopic traffic simulator

In this chapter, the issues which can help to further improve the macroscopic traffic simulators are discussed.

4.1 Dynamic load balancing

In parallel computing, one of the most difficult tasks is to distribute the workload equally to each processor. To distribute the workload which does not change their position in the domain such as distributing the FEM element to multiple processors is much easier compared to distributing the workload of traffic simulation which vehicles almost move throughout the domain (road networks). Even though, on the first iteration, the workload of traffic simulator is equally distributed to each processor. However, after few iterations have passed. The execution time will be not well distributed.

Consider Fig. 4.1, we can observe that the workload is equally assigned to each processor at the first few iterations (around 100 iterations). However, after that, the workload different is huge. In parallel computing, all MPI based software needs to do a synchronization process so that we can ensure that the necessary data which is used to maintain the continuity of the problem is successfully exchanged. While doing the synchronization process, the faster vehicle (Processor 2) in the Fig. 4.1 has to wait for the slower vehicle (Processor 1) to finish the computation so that they can start to synchronize the necessary data. Hence, the CPU cycles are wasted during the synchronization process.

The dynamic load balancing technique could be used to redistribute the workload when the big difference of computational time is observed. However, to redistribute the workload, we need to carefully set the frequency of the repartition. If the frequency is too high, this technique might not be useful as we taught. The computational time of repartition is one of the most time-consuming part of any parallel code. There are two way to repartition the domain: the first one is to do it on the master processor then broadcast the partition information to slave processors and the second method is to use parallel such as ParMETIS [22] which can partition the domain by using many processors. Neither of the methods consumes significant computational time. Therefore, we must make sure that the runtime difference as shown in the Fig. 4.1 is longer than the runtime of repartition process. Oth-

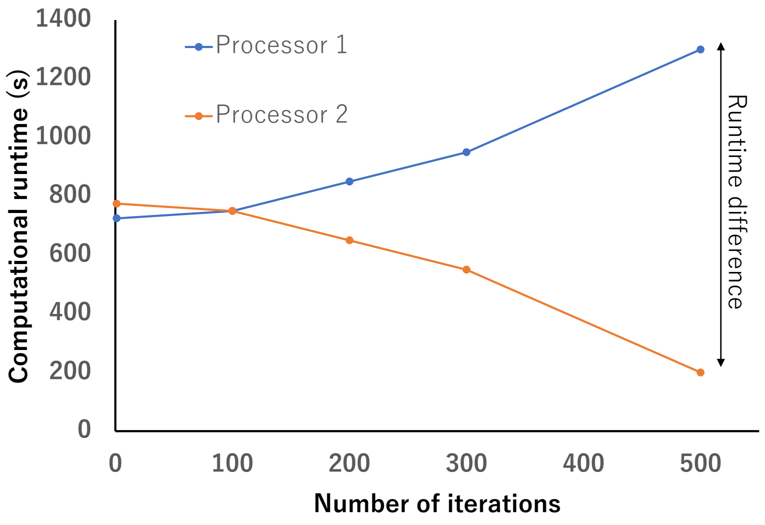


Figure 4.1: An example of a variation of computational runtime after several iterations.

erwise, we can reduce the computational time different but the total computational time is longer than the code without the dynamic load balancing technique. In our case, we could not apply this technique now because of the unpredictable runtime which will be explained in the next subsection.

4.2 The unpredictable execution of link transmission model

According to Thulasidasan et al. [32], they successfully implement the dynamic load balancing for their macroscopic traffic simulator which is capable for simulating the within-day traffic flow of New York road network within 1 hour and 30 minutes. They have shown that the execution time is well distributed among 256 processors. However, we could not just adopt their methodology to our code. The reason is that we cannot predict our runtime different as shown in a sample (Fig. 4.1).

As we observed from Fig. 4.2, we have a severe imbalance in cumulative computational time. Therefore, it would be logical to apply the dynamic load balancing technique so that we could minimize the wasted CPUs cycles. Unfortunately, we observe the unpredictable the different of the runtime between each iteration of our traffic simulator as shown in Fig. 4.3. Therefore, our traffic simulator must be redesigned before we can utilize the load balancing technique.

4.3 Possible further improvement of the embarrassingly parallel pathfinding's scalability

It has been mentioned that the scalability of embarrassingly parallel pathfinding could be further improved by distributing the vehicles to each processor by considering the spatial

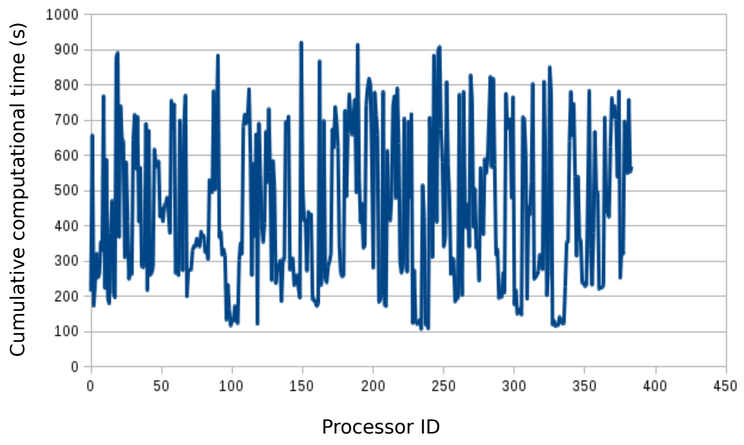


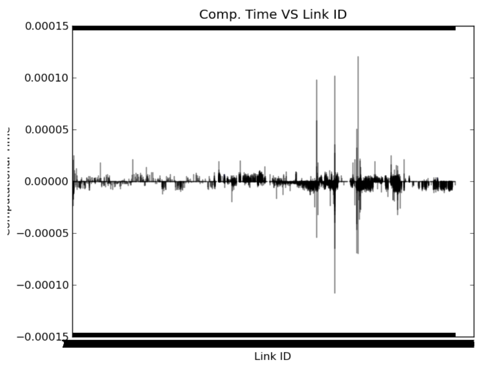
Figure 4.2: A big imbalance in computational time of each processor.

distance from an origin to destination (O-D) pair of each vehicle as a weight in section 3.4.1. Actually, the calculation time of Dijkstra's algorithm is significantly affected by the spatial distance of O-D pair because the Dijkstra's algorithm will stop the process after the destination node is found (Best-First Search). Therefore, at the first iteration, there is no single vehicle in the road network. We conduct the experiment to capture the calculation of time of all vehicle from O-D pairs of Nagoya network (3,000,000 vehicles). The result is shown in Fig. 4.4(a), Fig. 4.4(b), and Fig. 4.4(c). According to the result, we observed the imbalance of runtime between each vehicle. This imbalance is the main reason why the scalability of the embarrassingly parallel pathfinding cannot reach the ideal case.

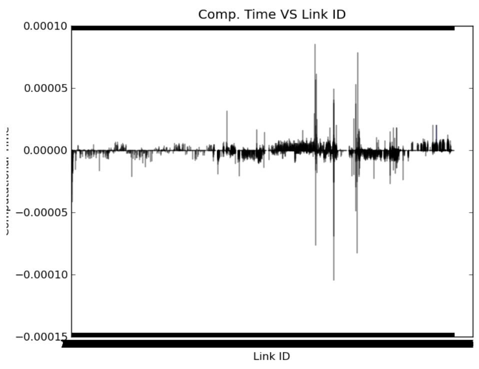
Instead of assigning an equal number of vehicles to each processor, it is more logical to assign the subset of vehicles which has an equal summation of the corresponding execution time of their pathfinding. However, after the network is fully loaded, the execution times of Dijkstra's algorithm which is shown in Fig .4.4 might not be accurate anymore. We could try to update these execution times of every N number of iteration or update the execution time when the repartition scheme is activated.

4.4 Possible further improvement of link transmission model based traffic simulator by using a communication hiding technique

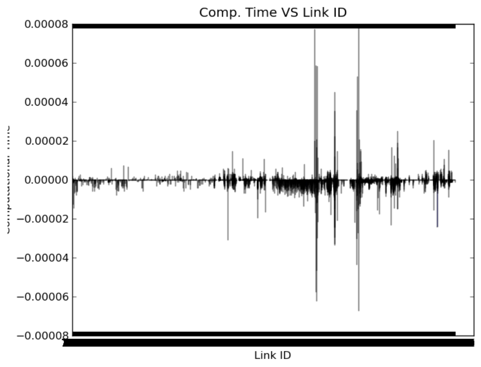
The basic information of the link transmission model is discussed in [38]. In this section, I would like to explain how exactly working in parallel. Figure 4.10 illustrates the steps involved in the preliminary updating vehicle trajectories algorithm. As discussed earlier, the communication hiding technique is to overlap the communication time and the computational time. In Fig. 4.10, we allocate the memory while posting the packed wave propagation data so that we can overlap communication time and computational time. We further improve the traffic simulator by overlap the execution time of intra links and the execution time of inter links as shown in Fig. 4.11.



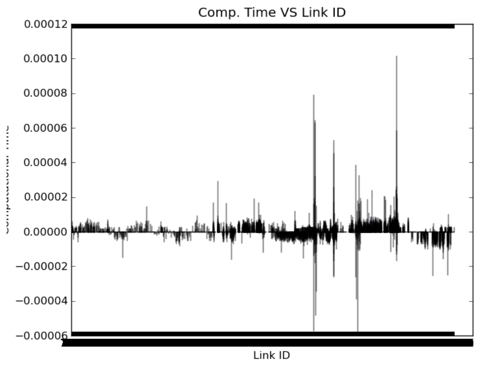
(a) Difference of runtime between iteration 1 and iteration 2.



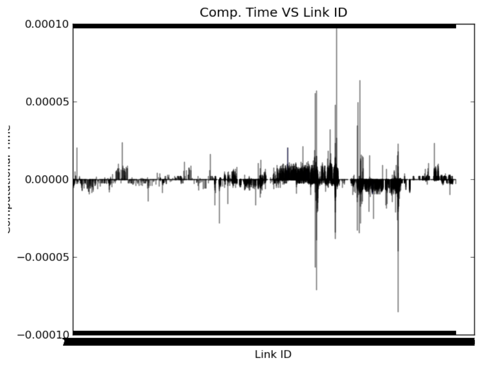
(b) Difference of runtime between iteration 2 and iteration 3.



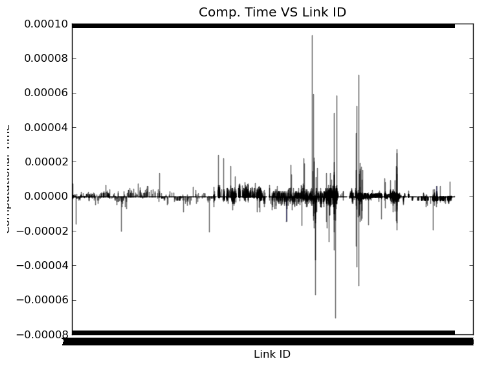
(c) Difference of runtime between iteration 3 and iteration 4.



(d) Difference of runtime between iteration 4 and iteration 5.

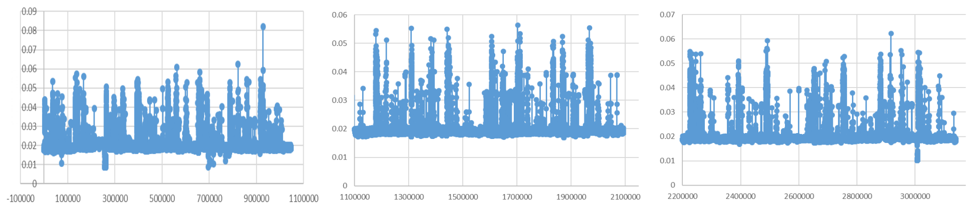


(e) Difference of runtime between iteration 5 and iteration 6.



(f) Difference of runtime between iteration 6 and iteration 7.

Figure 4.3: The different of runtime between each iteration in link transmission model based traffic simulator.



(a) Execution time of Dijkstra's algorithm (Vehicle 0 - 1,000,000).

(b) Execution time of Dijkstra's algorithm (Vehicle 1,000,001 - 2,000,000).

(c) Execution time of Dijkstra's algorithm (Vehicle 2,000,001 - 3,000,000).

Figure 4.4: The execution time of Dijkstra's algorithm of all vehicles in Nagoya network. The x-axis represents the vehicle id and the y-axis represents the execution time of Dijkstra's algorithm.

The idea is simple. We first analyze the inter links and post the packed analyzed information. After posting information to neighbor processors, we analyze the intra links which can be analyzed without any communication. Finally, the posted messaged will definitely arrive their destination processors since the number of intra links is technically much higher than the number of inter links which are at the boundary of the partition. To evaluate this idea, we conduct the experiment with 3 million vehicles in Nagoya road network. The result is shown in Fig. 4.5. The runtime of traffic simulator does not reduce significantly. It can reduce the execution time around 20 seconds when we use 128 processors. In addition, the execution time is longer than the preliminary code. Further, we investigate the scalability to see the parallel efficiency. Both of them has very poor scalability comparing to the ideal case as shown in Fig. 4.6. If we omit the ideal case line, we can see that the scalability is significantly improved as shown in Fig. 4.7. We can further improve both runtime and scalability by finding some useful tasks which do not require the transmission of the information as shown in 4.11. Unfortunately, in the current situation, we could not find the useful tasks to do while waiting for the synchronization of the data. Therefore, the traffic simulator must be redesign so that there are some independent tasks which can be done during the synchronization process.

We also investigate two main parts of the traffic simulator including the forward propagation and the backward propagation. The result in Fig. 4.8 shows that the forward wave propagation is more time consuming than the backward wave propagation. However, Fig. 4.9 illustrates that the backward wave propagation can scale in an acceptable range which almost reaches the ideal case. According to Fig. 4.9, the scalability of forward wave propagation is very poor. This is the source of poor scalability of our traffic simulation. Therefore, we must redesign the forward wave propagation algorithm so that we can enhance the scalability of traffic simulator.

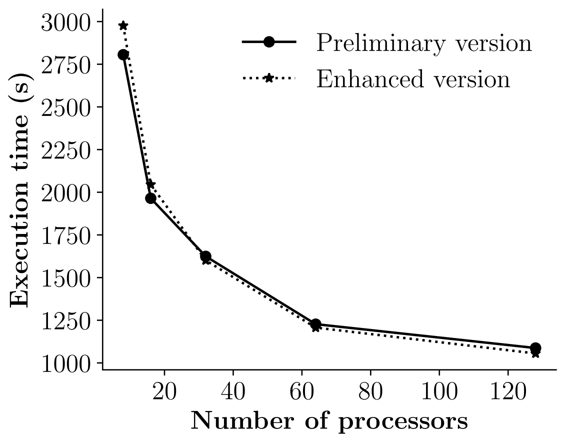


Figure 4.5: The execution time of preliminary and enhanced updating vehicle trajectories algorithm.

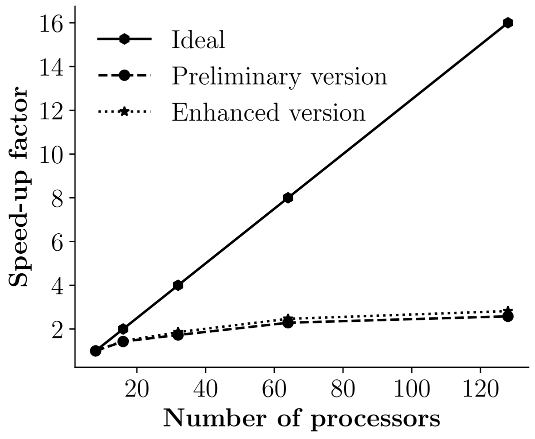


Figure 4.6: Scalability of preliminary and enhanced updating vehicle trajectories algorithm.

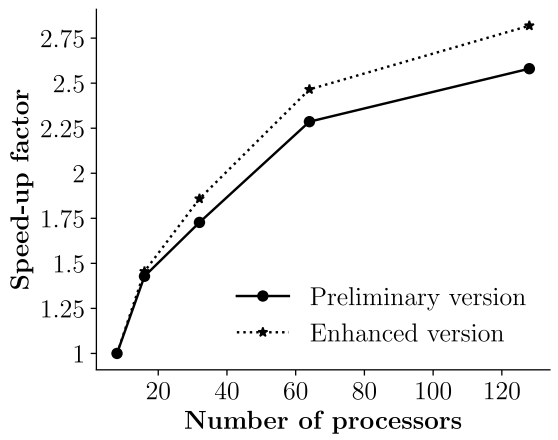


Figure 4.7: Scalability of preliminary and enhanced updating vehicle trajectories algorithm with out ideal case.

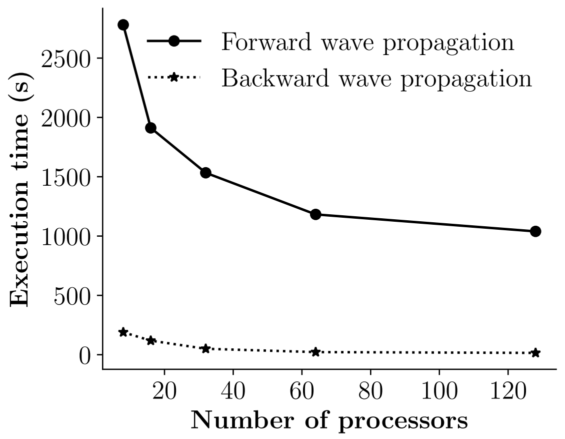


Figure 4.8: The execution time of forward and backward propagation.

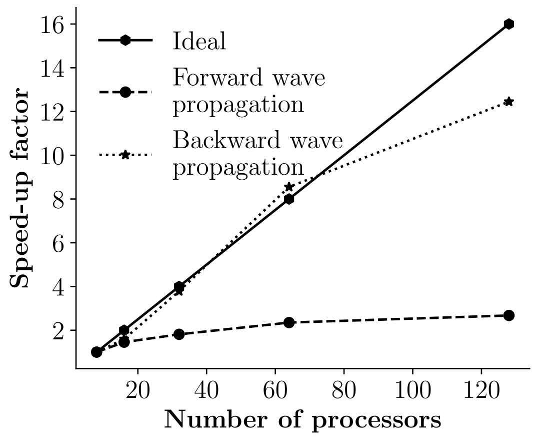


Figure 4.9: Scalability of forward and backward propagation.

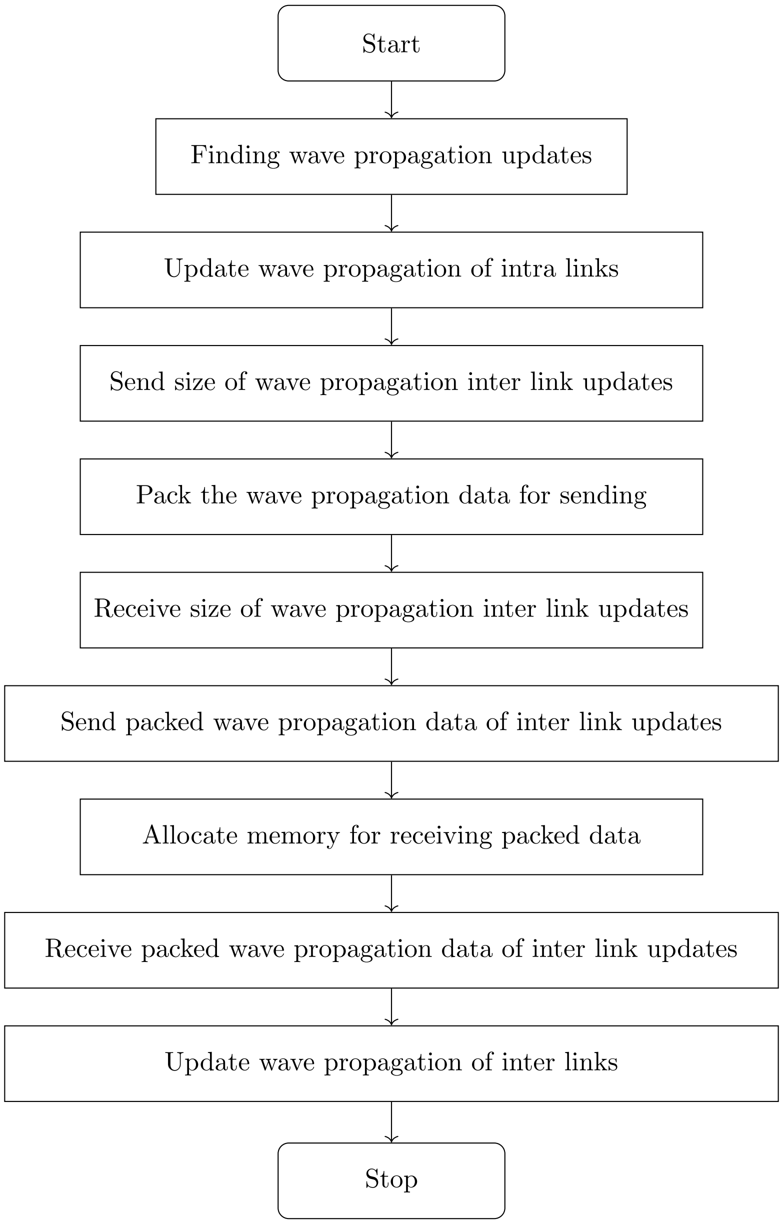

Start
Finding wave propagation updates
Update wave propagation of intra links
Send size of wave propagation inter link updates
Pack the wave propagation data for sending
Receive size of wave propagation inter link updates
Send packed wave propagation data of inter link updates
Allocate memory for receiving packed data
Receive packed wave propagation data of inter link updates
Update wave propagation of inter links
Stop

Figure 4.10: Flowchart of the preliminary parallel updating vehicle trajectories algorithm.

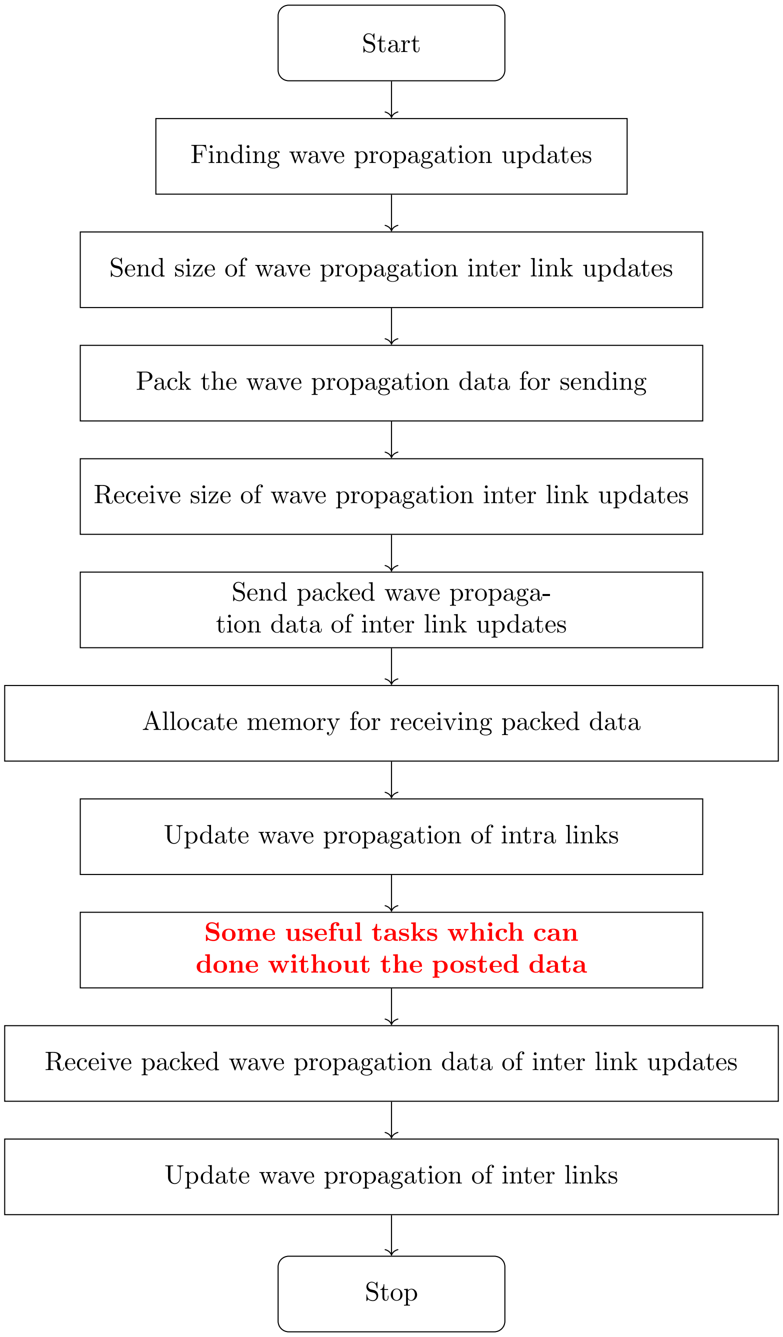

Start
Finding wave propagation updates
Send size of wave propagation inter link updates
Pack the wave propagation data for sending
Receive size of wave propagation inter link updates
Send packed wave propagation data of inter link updates
Allocate memory for receiving packed data
Update wave propagation of intra links
Some useful tasks which can done without the posted data
Receive packed wave propagation data of inter link updates
Update wave propagation of inter links
Stop

Figure 4.11: Flowchart of the enhanced parallel updating vehicle trajectories algorithm.

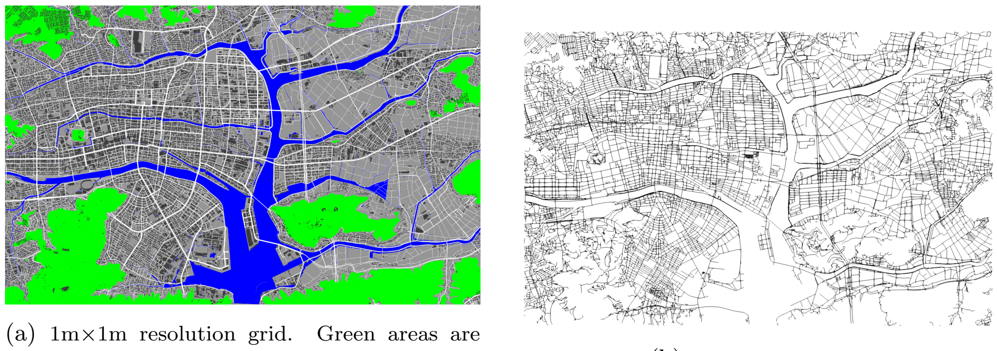
Chapter 5

Microscopic traffic simulator

Anticipating major earthquakes in Tokai, Tonankai and Nankai regions of Japan are predicted to bring major tsunami hazards to the neighboring coastal regions. While short tsunami arrival time and high tsunami heights are major threats in many areas, different regions have various other problems, like a long distance to the evacuation areas, insufficient capacity of evacuation shelters, large elderly population, narrow roads, car usage, etc. Numerical simulations are invaluable to the disaster mitigation authorities in seeking various strategies to address these problems, accelerating evacuation process, identifying unforeseen problems, etc. Motivated by the ease of use, less effort in developing and limited computational resources available, simple queue models with 1D networks are widely used in mass evacuation studies. While such simplified models provide much useful information, they have strong limitations. Related to mass evacuation in large congested urban cities, there are many scenarios which require the use of an accurate high-resolution model of the environment and complex models of individuals: people with different behaviors and responsibilities, multiple evacuation modes and their interaction, dynamic changes like a progressive inundation, visibility, etc. With the aim of simulating such demanding scenarios, we developed an Agent-Based Model (ABM) which includes a high-resolution model of the environment and complex agents which are capable of perceiving it ([25], [37]). In order to meet the high computational demand of complex agents in the high-resolution environment, we implemented a scalable parallel computing extension so that millions of agents in several hundreds of square kilometers can be simulated by utilizing high-performance computing facilities ([36], [1]). In this thesis, we present the details of the implemented agent-based model and its parallel computing extension.

5.1 Agent based model

The developed system is a time step driven agent based model which is consisting of a hybrid environment and cognitive agents. In order to model heterogeneous evacuating crowds, agents are assigned with different physical and behavioral parameters, different roles (i.e., evacuee, volunteer, officials, etc.), different level of access to information. Most systems model mass evacuation as agent flows on 1D networks and set the agents' speeds according to the average density in each link and suitable fundamental diagrams. On the contrary, the parameters of the autonomous agents of the developed system are tuned such that agents'



(a) 1m×1m resolution grid. Green areas are above 30m elevation.

(b) Topological graph

Figure 5.1: The environment is modeled as a hybrid of a 1m×1m resolution grid and a graph. Shown is 8.5km×5.4km area of Kochi city, Japan.

individual actions reproduce characteristics like fundamental diagrams as an emergent behavior.

5.1.1 Hybrid environment

With the aim of modeling evacuation process in microscopic details, we model the environment as a hybrid of a high resolution grid and a topological graph, see Fig. 5.1. The traversable 2D domain is modeled with a 1m×1m resolution grid, and updated at desired time intervals to model dynamic changes like tsunami inundation, fallen debris due to the earthquake, etc. The grid updates are generated based on the physics based simulators or observations. Connectivity between the traversable spaces in the grid is abstracted by a topological graph. The graph contains topological connectivity of an ordinary day (i.e., not updated according to dynamic changes of the grid). The graph serves as the base map of the agents' memory supporting agents' decision making functions. Scanning the grid in high resolution, agents recognize the features of their visible surrounding in the grid and any mismatches between the information in graph and grid. These mismatches, like a blocked road, and other experiences, like getting support from an official, are stored with reference to the graph so that experiences and new information can be easily taken into account in their decision making processes. The graph is equipped with a number of functions to find paths with various requirements (e.g. minimize the use of narrow roads in night time evacuation scenarios). The environment evolves in time, and its state at time t is denoted by E^t.

The grid and graph are automatically generated from GIS data. Traversable spaces can be generated from any of the following three data: center line and width of roads, road boundaries, or outline of the buildings. While generating the topological graph is straightforward with road center lines, we use thinning algorithm ([4]) to generate graph when the given input data is grid environment. The hybrid system is scalable to accommodate several hundreds of square kilometers in $1 \times 1\text{m}^2$ resolution. An accurate model of the environment, including data like buildings, contours, water bodies, etc., can be generated in a short time, when GIS data is available. As an example, the model of 8.5km×5.4km region shown in Fig. 5.1 was generated in 90 seconds.

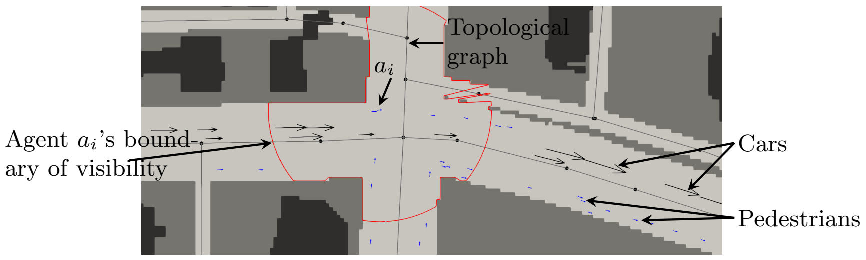


Figure 5.2: Snapshot of agents' movements at a junction. Blue and black arrows indicate instantaneous velocities of pedestrians and cars, respectively. Pedestrian agents walk along the edges, if the road can accommodate vehicles.

5.1.2 Agents

An agent, a_i , which mimics the behavior of a person during an emergency mass evacuation, consists of its state at time t, s_i^t, and a set of constituent functions, g^i, defines its role. s_i^t consists of agent's private data (e.g. destination, sight distance, decision, etc.) and information which are deducible by neighboring agents (e.g. speed, moving direction, etc.). The surrounding, with which an agent interacts at time t, consists of animate and inanimate parts. Inanimate surrounding consists of a_i's visible surrounding, $E_i^{visible,t} \subset E^t$, and any remote sources of information, $E_i^{remote,t} \subset E^t$, like the current travel time of the road segments along its current path. Animate surrounding consists of agents visible to a_i's, $A_i^{visible,t} \subset A$, where the set of agents $A = \{a_i | i = 1, 2, \ldots\}$. Figure 5.2 shows the boundary of visibility of an agent a_i with 30m sight distance; the agents inside this visibility boundary are a_i's $A_i^{visible,t}$.

The discrete time evolution of the agent based system, $A \cup E^t$, is governed by the functions updating the environment, and functions defining individual agent's role. Evolution of environment is modeled by updating with suitable functions, λ_j's , to mimic events like tsunami inundation, damage propagation, etc.; $E^{t+\Delta t} = \lambda_1 \circ \lambda_2 \ldots \circ \lambda_m(E^t)$. Autonomous actions of an agent a_i are defined by f_i, which updates the agent's state as $s_i^{t+\Delta t} = f_i(s_i^t, A_i^{visible,t}, E_i^{visible,t}, E_i^{remote,t})$. f_i is composed of suitable subset of the available set of constituent functions, $G = \{g^j | j = 1, 2, \ldots\}$, to mimic the role of corresponding agent; $f_i = g^{move} \circ g^{\cdots} \circ \ldots \circ g^{see}$.

The system provides a number of predefined constituent functions to perceive the high resolution environment, do complex path planning for different scenarios, identify whether a path is blocked, follow someone, etc. Each agent uses g^{see} function to scan its visible surrounding in high resolution (every 0.5°) like a radar; each agent has its own eyesight distance, which is typically set within 30m to 100m. This high resolution scanning enables agents to detect dynamic changes in the environment, move avoiding collision obeying restrictions like walk along road edges or drive on road lanes.

The constituent function for collision avoidance, g^{coll_av}, is implemented adopting the Optimal Reciprocal Collision Avoidance (ORCA) ([33]) with minor modifications and additional parameters to model interactions among agents ([24], [25]). The parameters of g^{coll_av} were tuned to model collision avoidance among pedestrians and pedestrians, cars and cars, and cars and pedestrians. In network flow based simplified models, within each road link, the rate of evacuees' flow is controlled according to a relevant fundamental diagram (i.e., relation between speed and flow density which is obtained from field observations). In contrast, in the developed system, the agents' speeds are not controlled externally. Instead, each agent autonomously controls its moving speed and direction using g^{coll_av}, which is tuned to reproduce observed fundamental diagrams. Several predefined update functions, f's, which are composed of suitable constituent functions are available to mimic roles of residents, visitors, officials who advise people to evacuate, cars, pedestrians.

5.2 High Performance Computing Extension

In order to meet the high computational demand of the autonomous agents, a scalable High Performance Computing (HPC) extension was developed. Some of the basic constituent functions, like g^{see} and $g^{identify\_road\_blocks}$, are computationally expensive. g^{see}, which is based on a memory intensive ray tracing algorithm, scans the visible neighborhood in high resolution. $g^{identify\_road\_blocks}$ is used to identify road blockages due to fallen debris or inundation and detect whether it is possible to continue avoiding debris. When encountered with road blockages and congestion, agents replan their paths or even start looking for new destinations. The processor intensive and memory intensive nature of these functions demand a significant amount of computational resources. As an example, a simulation with 90,000 agents with the environment shown in Fig. 5.1 requires 33 node hours in K Computer; a computing node of K-computer consists of an 8-core SPARC64 VIIIfx processor with 16GB of RAM. In order to address this high computational demand, a shared and hybrid parallel extension was developed. This section presents some details of the implemented HPC extension.

5.2.1 Domain decomposition

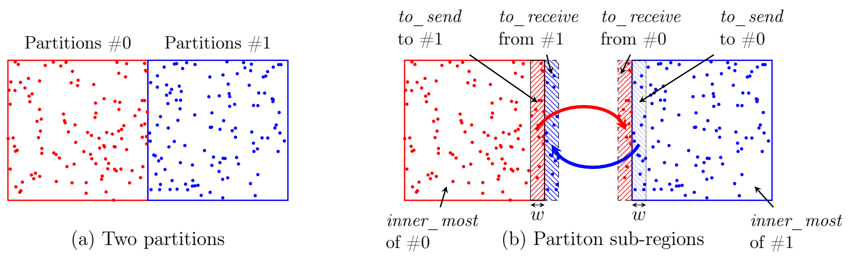


Figure 5.3: Domain decomposed for 2 MPI processes. In order to maintain the continuity, a region of width w from the neighbor partition #0 is included along the boundary of the partition #1

In order to utilize computational power of multiple computing nodes in a computer cluster, the domain of the environment is partitioned such that each computing node is assigned nearly equal workload. Almost all the computational workload is due to agents, and execution time related to the environment is mostly memory bound. Since the execution time of the heterogeneous agents highly depends on their type, visible surrounding, the density of other agents in the neighborhood, etc., we use measured execution time of each agent as a weight in creation partitions. Kd-tree based 2D partitions are generated using agents' execution time as weight, and each partition is assigned to an MPI process. In our case, a shared memory compute node is designated as an MPI process, and computations within a node are accelerated using OpenMP threads. Specifically, we use OpenMP's *task* level parallelism to accelerate computations.

Figure 5.3 shows the domain decomposed for two MPI processes. It is necessary to exchange information among those to ensure the continuity of the problem, since the compute nodes cannot directly access each other's memory. Along the boundary of each partition, we add dummy copies of a w wide region from the neighbor partitions (see Fig. 5.3(b)). In HPC literature, this dummy layers are referred as *halo*, or *ghost*. At the end of each iteration step, the information of the dummy agents in ghost layers are updated according to the latest states of the originals in the corresponding neighbor partitions, using MPI. By using the information of the agents in the ghost regions, when executing the agents of its own partition, MPI process can eliminate the effect of artificial boundaries introduced by partitioning (i.e. maintain the continuity). Since agents' actions are affected by the information within their visible surrounding, the thickness, w, of the ghost region is set to the longest eyesight of the agents.

5.2.1.1 Communication hiding

The inter-node communication, which is required for maintaining the continuity, is an extra overhead, and this overhead increases with the number of MPI processes. Communication hiding (i.e. doing useful work during communication) is a standard technique to minimize the performance degradation due to communication overhead. In order to hide communications, we group the agents in an MPI process into three mutually exclusive sets. *to_send* includes all the agents to be sent to neighbor MPI processes, while *to_recv* group includes all the agents in ghost regions. The rest are included in *inner_most*.

In order to hide the communications, first the agents in the set *to_send* is executed, and non-blocking messages to send the latest state of these agents to the corresponding neighbor MPI processes are initialized using non-blocking *MPI_Isend* and *MPI_Irecv* functions. While the communications proceed, the agents in *inner_most* set are executed. As long as execution of *inner_most* set takes longer than the time to complete the communication, we can effectively eliminate the communication overhead. Once, execution of *inner_most* is completed, the non-blocking messages are finalized, which updates the agents in ghost regions to latest states.

5.2.2 Dynamic load balancing

With time, agents move from the domain of one partition to another, which we call *agent migration* in this thesis. Unlike the above discussed ghost region updates, permanently moving agents to neighbor partitions is expensive. Figure 5.4 shows the time history of run-time with 10 million agents in a 588 km^2 region.

Movement of a large number of agents in or out from a partition leads to imbalances of workloads assigned to MPI processes, lowering the computation efficiency. When the load imbalance reaches a critical state, domain is re-partitioned to re-assign equal workloads. Figure 5.3 (a), (b) and (c) show how the partitions are rearranged according to the agent distribution, at early stages of a simulation with 80,000 agents. The process of this dynamic load balancing is the heaviest in communication and can consume a significant time depending on the total number of agents being simulated.

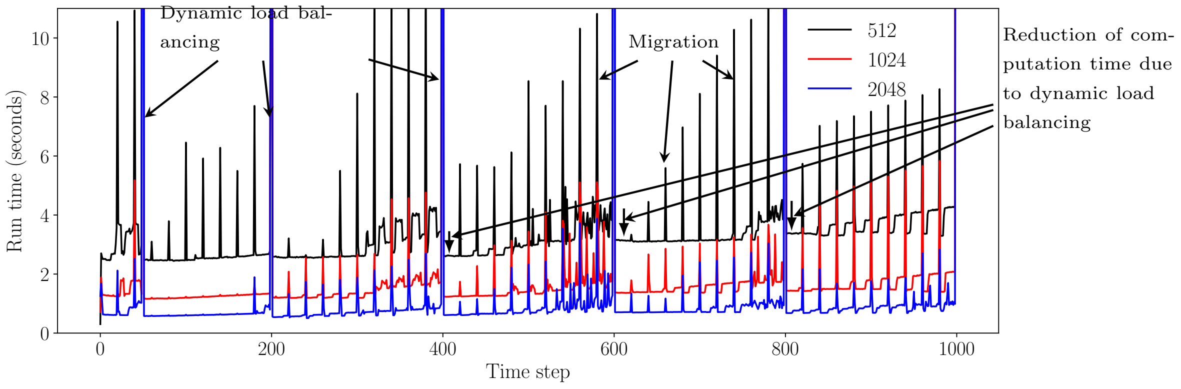


Figure 5.4: Run-time history for 1000 iterations with 10 million pedestrian agents. Three graphs indicate time history with 512, 1024 and 2048 nodes of K computer. Migration and dynamic load balancing are executed on a single thread.

5.3 Dynamically mapping to different number of MPI processes

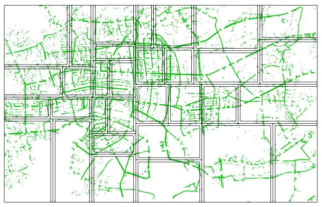
With the progress of time, the agents start to concentrate along roads leading towards evacuation regions and decrease in number since evacuated agents are deactivated. Both these can force to reduce the number of MPI processes. Since the minimum width of the *to_receive* regions, w, is set to the maximum eyesight, the length or width of a partition cannot be smaller than $2w$. The concentration of agents to a few number of streets, as seen in Fig. 5.5 (d) and (e), give rise to partitions with closer to $2w$ side lengths. When that happens, either the program has to be aborted, or the problem should be mapped to a smaller number of MPI processes. Though this mapping is a complex and time consuming process, it is necessary to prevent premature abortions and continue until the end of the desired simulation. Figure 5.5 (d), (e), and (f) show the partitions after mapping to 16 and 8 MPI processes.

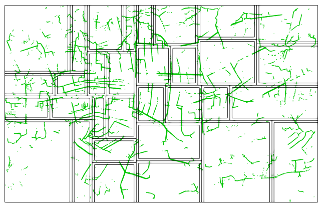
Number of nodes	Execution time (s)	Strong scalability (%)
512	3904.0	-
1024	2067.7	94.4
2048	1258.5	82.1

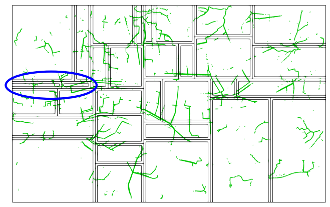
Table 5.1: Runtime and strong scalability with 512, 1024 and 2048 nodes in K computer.

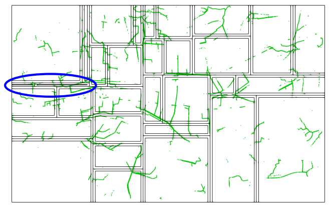
5.3.1 Scalability

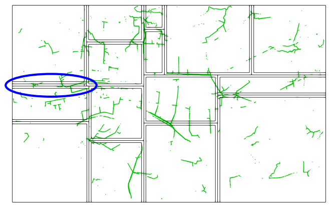
In order to conduct the target large scale evacuation simulations (e.g., tsunami evacuation of a long stretch of coastal region, or evacuation of large metropolis like Tokyo after a major earthquake) utilizing HPC facilities, the developed code must scale to a large number of


(a) Itr. 1. 79972 agents, 32 partitions


(b) Itr. 1000. 71339 agents, 32 partitions.


(c) Itr. 2000. 61079 agents, 32 partitions.


(d) Itr. 2400. 57049 agents, 16 partitions.


(e) Itr. 3600. 46746 agents, 16 partitions.

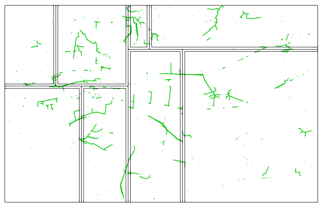

(f) Itr. 5800. 34652 agents, 8 partitions.

Figure 5.5: Partition arrangements at different iteration steps with 80,000 agents in the environment of Fig. 5.1. Partitions are dynamically adjusted according to the distribution of agents and workload; shown in green dots are the agents. When the number of agents is too small and/or concentration of agents leads to partitions with $2w$ or lesser side length, like in (d) and (e), the problem has to be mapped to a smaller number of MPI processes.

MPI processes. Near ideal scale (i.e., halving the execution time when doubling the number of MPI processes) to thousands of compute nodes with the complex heterogeneous agents is a challenging task. According to autohors' literature survey, there are no records of agent based models of similar complexity which scales even to hundred compute nodes.

In order to test the scalability, we conducted several simulations with 10 million pedestrians in a 588 km^2 area in central Tokyo. The pedestrians were set to move to the nearest park, and their walking speeds were set according to [5]. To produce maximum computation load, all the agents were set to start evacuation immediately. 1000 iteration steps were simulated on 512, 1024 and 2048 nodes of K computer.

The Table 5.1 shows the strong scalability which is obtained from the simulations. Strong scalability is defined as $({}^{T_m}\!/_{T_n})/({}^{n}\!/_{m})$, where T_i is the execution time with i number of MPI processes and $n \geq 2m$. These simulations indicate that not only the developed scales well up to 2048 nodes (16384 CPU cores), but also it can handle millions of agents in several hundreds of square kilometers in 1m×1m resolution model of the environment. Figure 5.4 shows the time history of run-times. The tall spikes indicate the dynamic load balancing at 100 iteration steps intervals. The smaller spikes indicate the migration of agents from one partition to another, at 20 iteration intervals. Each of dynamic load balancing took nearly one minute.

The developed system has potential to scale even beyond the results shown in Table 5.1. Currently, only the execution of agents uses shared memory parallelism to best utilize multiple threads in each compute node. The migration and dynamic load balancing functions are executed on single threads. Enhancing these time consuming functions with shared memory parallelism will further improve the scalability.

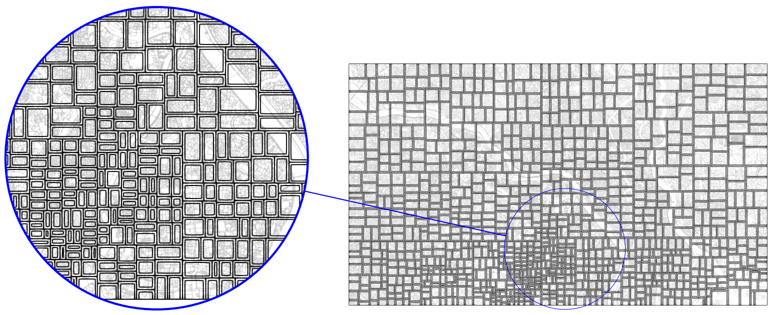


Figure 5.6: 2048 partitions of Tokyo domain. High concentrations of agents in the central Tokyo, encircled with a black ellipse, gives rise to partitions with smaller dimensions and poor aspect ratio. Presence of partitions with side lengths close to $2w$ makes communication hiding ineffective and significantly lower the scalability.

5.3.1.1 Partitions and scalability

In the central Tokyo environment, which is used for the above scalability tests, the city center had a high agent density compared to the outskirts (see Fig. 5.6). This high agent concentration produced smaller partitions, while, with time, agents start to further concentrate along the roads to evacuation regions, as shown in Fig. 5.5. This high concentrations of agents lead to partitions with side dimensions closer to $2w$ and poor aspect ratios, like the one encircled with a blue color ellipse in Fig. 5.5(c). In partitions with dimensions closer to $2w$, most of the agents are located in *to_receive* regions, and have only smaller number of agents in *inner_most* region. This makes the communication hiding ineffective and significantly lowers the scalability. The simulation with 2048 nodes starts to produce partitions with side dimensions closer to $2w$ at iteration 600 and onward, which is why there is 12% reduction in scalability with 2048 nodes. Partitioning scheme with better aspect ratio will enable to maintain the high scalability up to a larger number of MPI processes.

5.4 Enhancement of domain generator

The basic concept of environment in our agent-based simulator is explained in section 5.1.1. The agents make their decision by considering their surrounding environment. The previous domain generator has generated poor topology graph as shown in 5.7. Therefore, it is difficult to formulate decision making of agents. The zigzag patterns are obviously observed. These zigzag patterns introduce large error to the simulation result (traffic flow) because vehicles may need to abruptly change their speed at the shape turn of topology graph. At intersections which are the most significant path of traffic simulation, we observed the malformed intersections and disconnected intersections. These defects in the domain make the agents behave badly at the intersections. Hence, the result of the simulation is not reliable. Also, previous domain generator is computationally demanding. It took more than 1 hour to generate the domain with the size of 100 km^2. We aim to improve the mentioned domain generator.

The reason behind poor quality is that the topology graph is generated by using a thinning algorithm. It is difficult for thinning algorithm to define a centerline from a raster

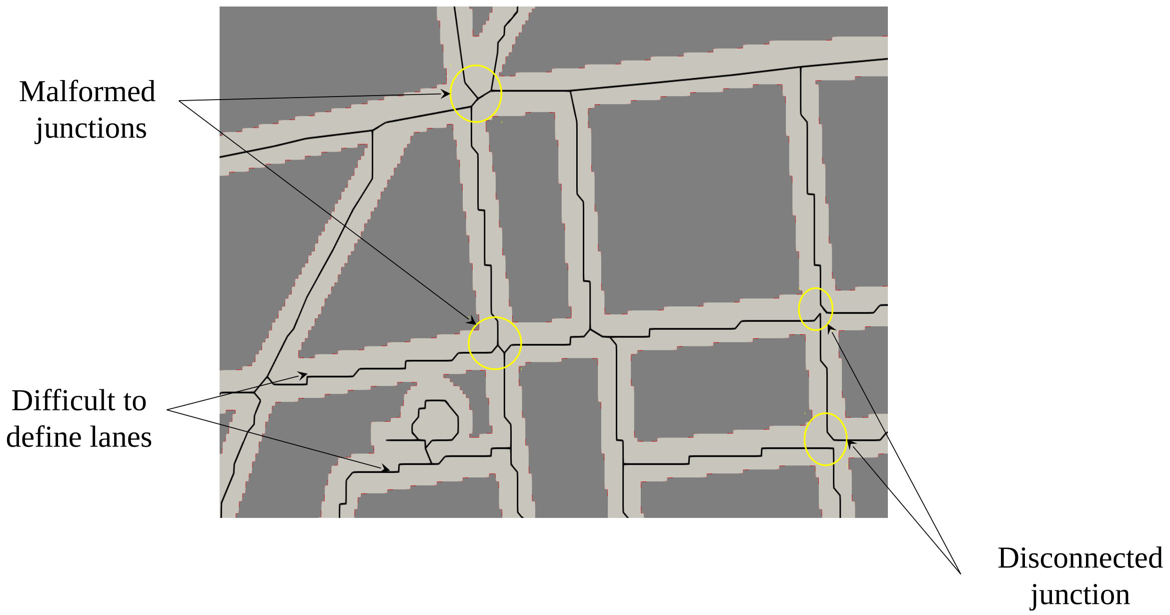


Figure 5.7: The bad quility topological graph from previous domain generator.

grid with a size of $1m \times 1m$. The flow of previous domain generator is illustrated in Fig. 5.8. The road boundaries which are obtained from GIS data are converted to a raster grid. After that, the raster grid will be abstracted to a topological graph.

To fix this problem, instead, we use the road centerlines from GID data to generate the raster grid. The reason that we don't use these road center lines in the first place is that the road centerlines are just available recently. The flow of enhanced domain generator is shown in Fig. 5.9. We created topological graph directly from road center lines. After that, we generate a raster grid by drawing road centerlines to a bitmap file according to road width data from GID data. As a result, all mentioned problems have been solved as shown in Fig. 5.10. The enhanced domain generator uses less than 1 minute to produce the domain with the size of 625 km^2 which are a lot faster than the previous domain generator. At the current stage, multiple lanes can be defined easily. Therefore, a high fidelity model of intersections can be applied conveniently. We also include water bodies, contour lines, building, and park area in the enhanced domain generator.

In addition, the enhanced domain generator also reduces the number of unnecessary nodes and links of a topological graph. There are 351,318 nodes and 717,467 links in Kochi city's domain (Fig. 5.1) when the previous domain generator is used. After the enhancement, we have 79,845 nodes and 170,826 links which are around 4 times lesser compared to the previous domain generator. This reduction leads to faster computation of pathfinding algorithm according to the complexity of Dijkstra's algorithm ($\mathcal{O}(|E| + |V|log|V|)$).

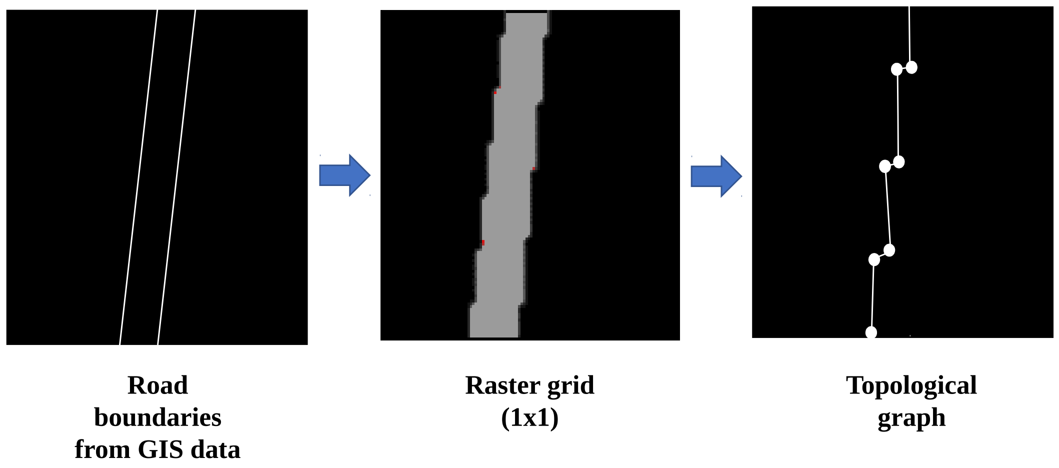


Figure 5.8: Flow of previous domain generator.

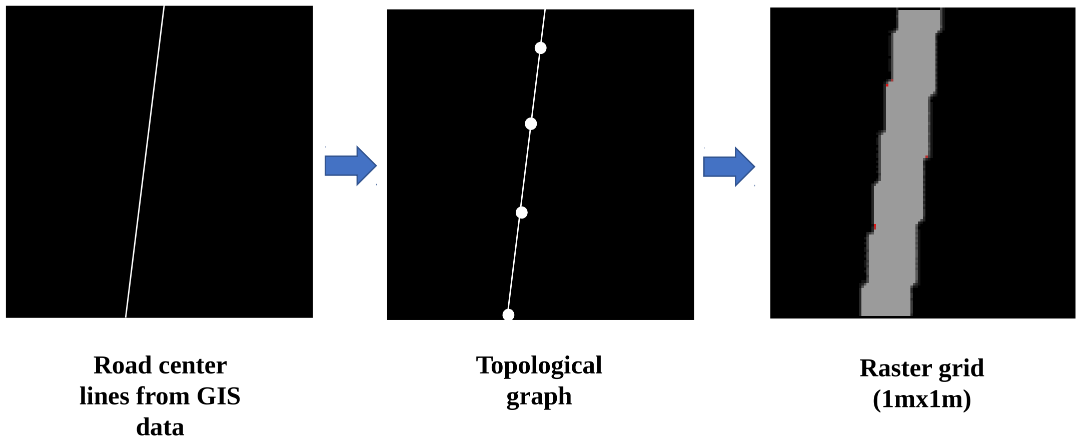


Figure 5.9: Flow of enhanced domain generator.

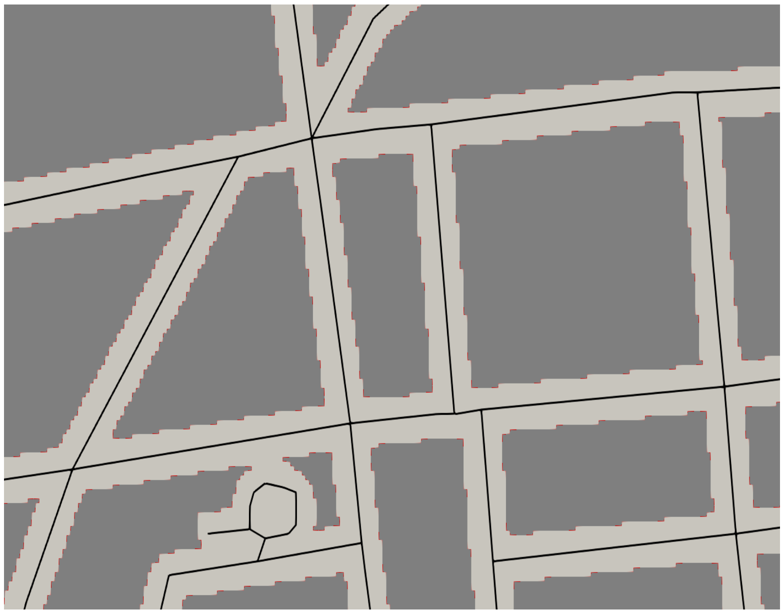


Figure 5.10: The enhanced topological graph from new domain generator.

Chapter 6

Agent-based model for car-car and pedestrian-car interactions at unsignalized junctions

In countries like Japan, where most streets are narrow and recommended to evacuate on foot, tsunami triggered evacuation can produce heavy interactions among cars and an unusual number of pedestrians, especially at junctions. When addressing the problems like how many vehicles can be allowed, what are the allowable time windows for car usage, etc., it is vital to reasonably mimic these unusual pedestrian-vehicle interactions at unsignalized junctions; during a disaster, signal lights may not be functioning or even not respected. Especially, the car agents must be capable of reproducing realistic trajectories and speed profiles within a junction, since cars can be more than twenty times faster compared to pedestrians. While it is rare to find pedestrian-car interaction models at unsignalized junctions, even existing car-car interaction models, which are mostly based on collision avoidance algorithms [34, 9, 12, 13, 15], scheduling scheme [8], and game theory [23], etc., do not reproduce realistic trajectories and speed profiles.

In this chapter, we propose a lightweight agent-based model to accurately reproduce trajectories and speed profiles of vehicles at junctions and reasonably reproduce vehicle-vehicle and vehicle-pedestrian interactions at unsignalized junctions. These features are implemented in an existing agent-based framework [26] for large-scale evacuation simulation; the agents are a cognitive type, environment is modeled as a hybrid of 1m$\times$1m resolution grid and a topological graph of road center lines. It is demonstrated that vehicle trajectories at junctions can be easily approximated with B-splines, while free flow speed profiles can be approximated with cubic polynomials. Vehicle-vehicle interaction is modeled based on simple physics-based approximations, and the uncertainty and fear of collision in close encounters. In addition, pedestrians are given priority when modeling vehicle-pedestrian interactions. With several numerical examples, it is demonstrated that reasonable interaction behaviors can be reproduced.

The rest of this chapter is organized as follows. Section 6.1 presents the approximations for vehicle trajectories and free flow speed profiles at junctions. Vehicle-vehicle interactions and vehicle-pedestrian interactions at unsignalized junctions are presented in section 6.2 and 6.3, respectively. Section 6.4 presents several numerical examples to demonstrate the

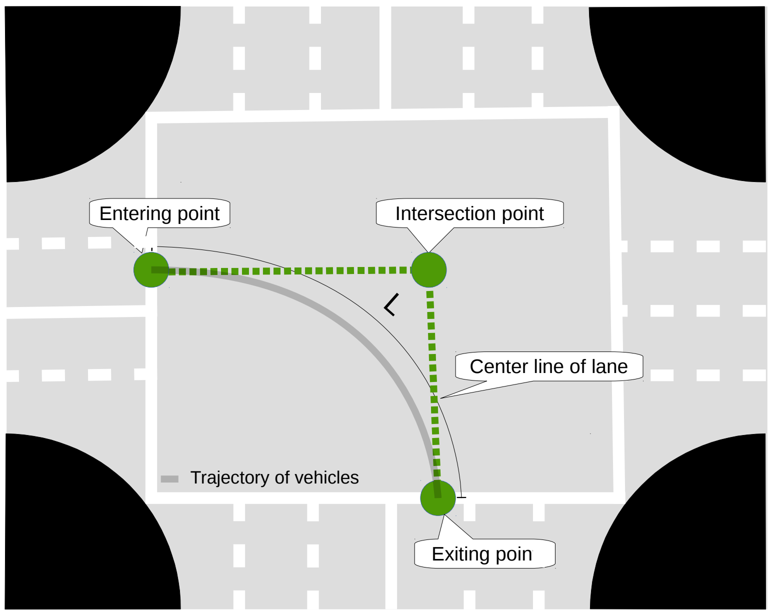


Figure 6.1: Vehicle trajectories at intersections are approximated with third-order B-spline curves.

accuracy of approximations and the interactions can be reasonably reproduced.

6.1 Approximation of trajectories and free flow speed at intersections

6.1.1 Vehicle trajectories at intersections

We sought to find fast-to-evaluate parametric approximations which are easy to define and can be used for any intersection geometry. Though there are approximations combining Euler spirals and circles, which satisfies traffic engineering requirements [2], those are complicated to define and evaluate. Analyzing vehicle trajectory observations by Alhajyaseen et. al. [2], we found that vehicle trajectories can be easily approximated with B-splines. Trajectories at most of intersection geometries, except u-turns, can be approximated using a B-spline curve with knot vector $[0, 0, 0, 1, 1, 1]$, and lane centre at the entry point, the intersection point between centre lines of the incoming and outgoing lanes, and lane center at the exiting point (see Fig. 6.1) as control points. While the required three control points are known, the above B-spline knot vector defines a simple third-order Bézier basis function, making it effortless to define the vehicle trajectory and simple to evaluate points on the trajectory. Comparing with the approximation proposed by Alhajyaseen et. al.[2], we found that our B-spline approximation deviates only a few centimeters, which is negligible for this particular application or even compared to vehicle dimensions. Also, we found that a very accurate approximation can be made with NURBS, but this requires observations and solving a linear set of equations.

In order to reduce the amount of computations in updating agents, we define the vehicle

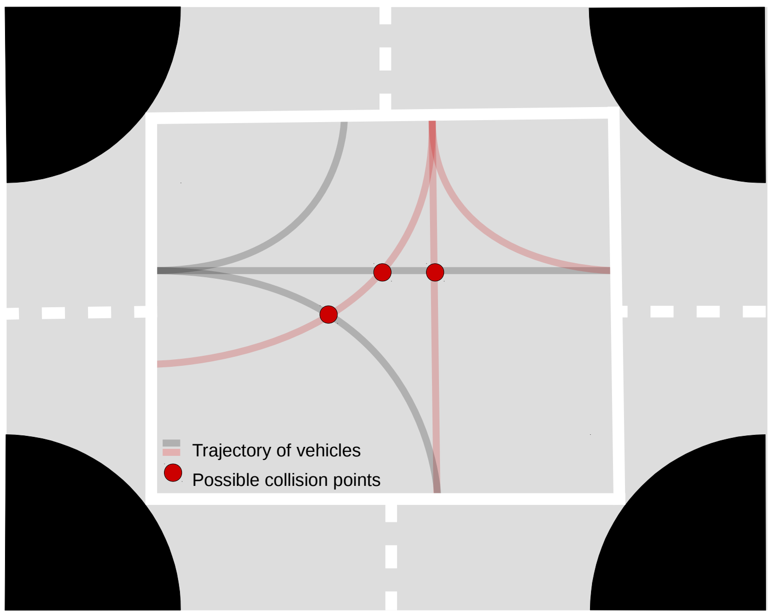


Figure 6.2: Intersection points of multiple trajectories.

agents' trajectories using a few numbers of points (e.g. 5 to 10 points) on the B-spline curve, and the intersection points of multiple trajectories are pre-calculated as shown in Fig. 6.2. Potential collision zones of vehicles on different trajectories are defined based on these intersecting points.

6.1.2 Speed profile of a vehicle at intersections

In modeling vehicle-pedestrian interactions at junctions, it is essential for car agents to have realistic free-flow speed profiles. Since vehicle speed can be one order of magnitude larger than that of a pedestrian, unrealistic or sudden changes of vehicle speed can induce abnormal pedestrian behavior. According to Charitha et al. [10], speed profiles of vehicles can be approximated with higher order polynomial curves. While their approximation requires field observations, we use the following third order polynomial approximation which can be defined with three known parameters and one constraint.

$$\begin{aligned} v(x) \quad = \quad & (-4V_{approach} + 4V_{depart}) \left(\frac{x}{L}\right)^3 + (8V_{approach} - 4V_{min} - 4V_{depart}) \left(\frac{x}{L}\right)^2 \\ & + (-5V_{approach} + 4V_{min} + V_{depart}) \left(\frac{x}{L}\right) + V_{approach}, \end{aligned} \tag{6.1}$$

where L is the length of trajectory and $x \in [0, L]$. We assumed that at $x = {}^{L}/_{2}$ the acceleration is zero (i.e. ${}^{dv}/_{dx}|_{x=\frac{L}{2}} = 0$) and the vehicle reaches the minimum speed of V_{min}. V_{min} can be considered as the maximum allowable speed to prevent accidents due to centripetal force; given the mass of a vehicle, maximum curvature of the trajectory, road surface condition, etc., V_{min} can be defined. $V_{approach}$ is the vehicle's approaching speed to the junction (i.e. $v(0)$) and V_{depart} is the desired departing speed $v(L)$.

The above speed profile defines only an upper bound for the speed of a given car agent, at a length x along its trajectory. Let the speed of i^{th} car agent be $u_i(x)$ and the corresponding upper bound defined by Eq. 6.1 be $v_i(x)$. If $u_i(x) \geq v_i(x)$, i^{th} car agent decelerate itself to follow the speeds of Eq. 6.1, and otherwise continue at $u_i(x)$ or any desired speed below $v_i(x)$. The details of deceleration and acceleration are described in the next section.

6.2 Car-car interactions at unsignalized interactions

During a tsunami triggered mass evacuation, it is highly probable that the preceding earthquake to render the signals at intersections dysfunctional, hence the need of simulating interactions at unsignalized intersections. While we use the trajectories and speed profiles presented in the previous section for modeling free flow traffic, a simple and lightweight vehicle-vehicle interaction model to resolve conflicts and possible collisions at close encounters is presented in this section.

6.2.1 Assumptions and observations

We make the following two assumptions in order to devise a simple, yet realistic, vehicle-vehicle interaction model.

Assumption 1 *Vehicles will not change their lanes while traveling through an intersection.*

Assumption 2 *Higher priority is given to the vehicle that can first enter the region of a potential collision.*

Both these assumptions are not far from the reality. It is a daily experience that almost all the drivers stick to standard trajectories at intersections, and as a matter of fact, changing lanes at an intersection is one of the major sources of accidents [**?**]. Also, many observations support the second assumption[16]. In addition, we use the following observed behaviors of a rational driver in implementing the interaction algorithm.

1. When approaching an intersection, reduces the speed to a comfortable range.
2. Observes neighbor vehicles' position and their turn signals, and estimates their relative speeds.
3. Estimates, with some safety factors, whether his vehicle is going to collide with any neighbor vehicle.
4. If potential collision is identified, avoids it by applying comfortable deceleration to maintain a safe distance in between.

A simple mistake or misjudgment in close encounters at intersections, where vehicles are moving fast at few seconds time gap between each other, can lead to fatal accidents. In fear of collision at such close proximities, drivers usually include a safety factor in their decision making in order to cope with their imperfect mental judgments, driver's reaction time, unpredictable actions of any neighboring driver, mistakes made by neighboring drivers, etc. The most common safety factor is including an extra buffer to mental estimations, in terms of distance or time, instead of mathematically perfect estimations. We included this sense of fear in the decision making of car agents by making them to maintain a gap between neighboring vehicles or pedestrians.

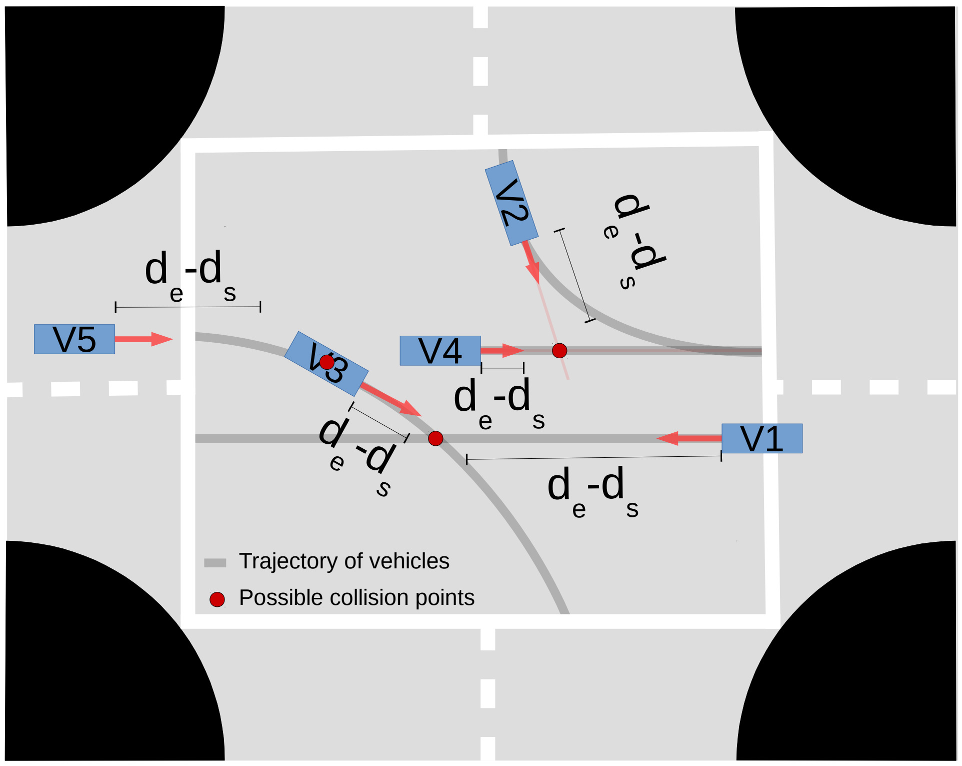


Figure 6.3: Examples of the three different types of collisions considered: V5-V3 fast moving rear vehicle and slow-moving front vehicle; V1-V3 intersection of different trajectories; V2-V4 extrapolated current moving directions on merging trajectories.

6.2.2 Potential unsafe regions

In vehicle-vehicle interactions, we consider three types of potential unsafe zones, each around the following three potential points of collisions (see Fig. 6.3).

1. The collision point of fast-moving rear vehicle and slow-moving front vehicle on the same trajectory.
2. Intersections of two independent trajectories.
3. Intersection of linearly extrapolated current moving directions of two vehicles on merging trajectories.

The third type of point of collision, which considered extrapolated current moving directions, is included to take any mistakes by a driver on merging trajectories into account. The intersecting points which fall under the second type are pre-calculated as explained in the latter part of section 6.1.1 to reduce computational overhead. List of trajectories falls under the third item are also pre-identified to reduce conditional branching in computer implementations.

6.2.3 Avoiding vehicle-vehicle collision within an intersection

As explained in the beginning of this section, we assume that vehicle agents do not change their lane or direction to avoid the collision, within an intersection. Instead, they reduce their speed to avoid any potential collisions. Each vehicle agent identifies all the neighboring vehicles leading to any of the above given three types of collisions, estimates whether itself and any of those vehicles would come closer than a safe distance d_s, and decelerate to maintain or stop before reaching d_s distance to the agent with the shortest time to the collision. While the details of agents' deceleration process are given in the rest of this section, the basic steps involved in vehicle-vehicle collision avoidance are given in Algorithm 6.1.

In estimating time to reach the potential point of collision with neighboring vehicle agent a_j, a vehicle agent a_i assumes that neighbor a_j will continue driving at the current observed speed up to the point of potential collision. However, the agent a_i makes an accurate estimation of its own travel time to the point of collision based on its expected future speed changes. As an example, if a_i is on a curved trajectory, it estimates the travel time based on its expected speed profile bounded by the Eq. 6.1. a_i's accurate estimation of time is not unrealistic since drivers are capable of making fairly accurate estimations of short distance travel times based on their past experiences. However, uncertain of what the neighboring drivers would do, accurate estimation of their travel times is not possible.

Algorithm 6.1: Pseudo code for vehicle-vehicle collision avoidance at intersections.

```
input  : One considered vehicle (cveh),Neighbor vehicle information
output: Collision free speed of considered vehicle
1  initialize list $OVList$ ;                                  /* the observed vehicle list */
2  initialize float $OptimalSpeed$ = FreeFlowSpeed ; /* the collision free speed */
3  initialize vector2d $collpoint$ ;                      /* the possible collision point */
4  Observing neighbor vehicles and keep them in $OVList$
5  while $OVList$ is not empty do
6      $sveh = OVLIST$.pop() ;                                          /* selected vehicle */
7      if $sveh$ is possible to collide with the considered vehicles then
8          Determine $collpoint$ between $cveh$ and $sveh$;
9          if $cveh$ will not reach the collpoint before $sveh$ then
10             Calculate the collision free speed ($collfreespeed$) for $cveh$
11             if $collfreespeed < OptimalSpeed$ then
12                 $OptimalSpeed = collfreespeed$
13             end
14         end
15     end
16 end
17 return $OptimalSpeed$
```

6.2.4 Deceleration to avoid collisions

We assume that the safe distance d_s, which is preferred by the vehicle agent a_i as a safety factor against misjudgments and uncertainties, is $d_s = \tau_i v_i$, where τ_i and v_i are reaction time and current speed of a_i, respectively. The effective distance, d_e between the considered agent a_i and its neighbor a_j is defined as

$$d_e = d - r_i - r_j, \tag{6.2}$$

where d denotes the distance between centers of the two agents, and r_i and r_j denote their radii. To simplify the computations, we represent vehicle agents with a circle of radius r.

Consider a vehicle agent a_i with speed $v_i(t)$, and agent a_{front} with speed $v_{front}(t)$ to be moving in front of a_i on the same trajectory. If $v_i > v_{front}$, to avoid the collision with the slow moving front agent a_{front}, the fast moving rear agent a_i decelerates to match their speeds (i.e. $v_i - v_{front} = 0$), before reaching a safe distance of d_s to a_{front} (see Fig. 6.3). Based on simple physics, we can define the equation for updating a_i speeds at Δt time intervals as

$$v_i(t + \Delta t) = v_i(t) + \frac{(v_{front}(t) - v_i(t))^2}{2(d_e - d_s)} \Delta t. \tag{6.3}$$

In all the other potential collision situations, except the above, all the vehicles later reaching the point of potential collision decelerate to completely stop, allowing the agent first reaching the point of collision to continue undisturbed. To completely stop before reaching a safe distance of d_s to the point of potential collision, a vehicle a_i which later reach the point of collision updates its speed according to

$$v_i(t + \Delta t) = v_i(t) - \frac{v_i(t)^2 \Delta t}{2(d_e - d_s)}. \tag{6.4}$$

6.3 Car-pedestrian interactions at unsignalized interactions

As mentioned in the introduction, in countries like Japan, the main mode of tsunami evacuation is walking, and the roads are narrow. Under these circumstances, pedestrians may occupy the narrow roads creating unusual interactions among a large number of pedestrians and a small number of vehicles. One objective of this study is to model such vehicle-pedestrian interactions at unsignalized junctions.

In addition to the two main assumptions related to vehicle-vehicle interactions (see section 6.2.1), we assume that vehicles give priority to pedestrians. Considering the fact that pedestrians have a long distance to walk to reach a safe high ground, this is not an unreasonable assumption.

When interacting with pedestrians, a car agent a_i identifies all the visible pedestrian agents, finds which pedestrians are going to walk across its fixed trajectory and estimates the time to reach a_i's trajectory assuming each pedestrian will continue walking in their current direction as illustrated in Fig. 6.4. Just as in vehicle-vehicle interaction case, vehicle agent a_i makes an accurate estimation of time to collision according to the expected speed profile (see section 6.2.3), and identifies the first pedestrian agent to collide on. If the pedestrian agent can reach closer than the safe distance d_s, it decelerates according to Eq.

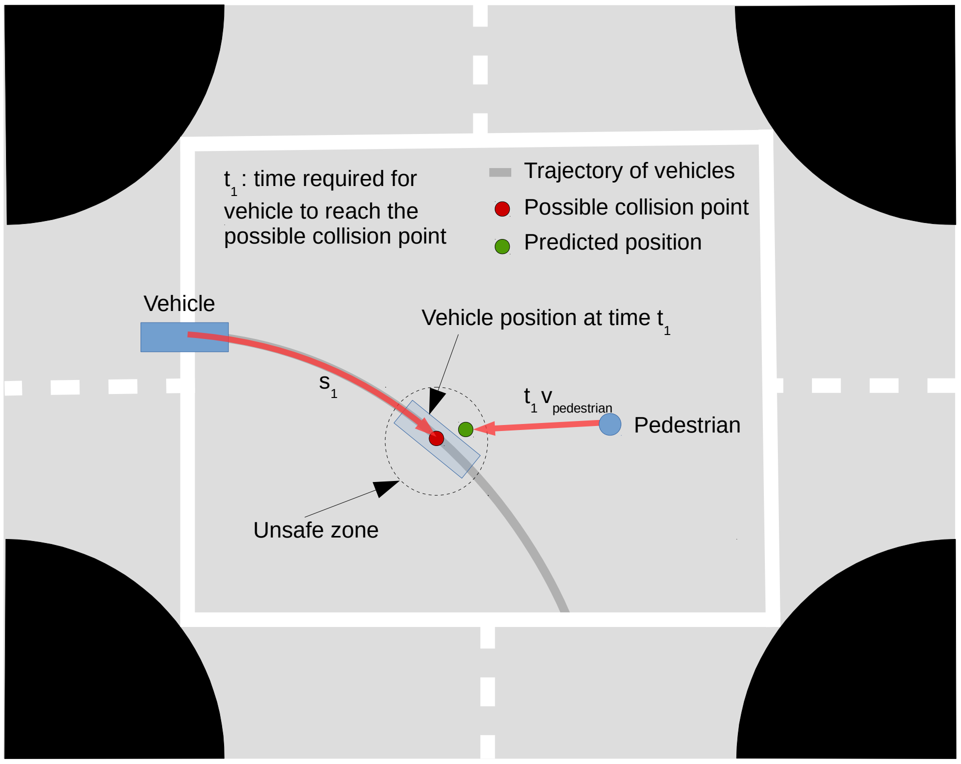


Figure 6.4: Interaction between vehicle agents and pedestrian agents

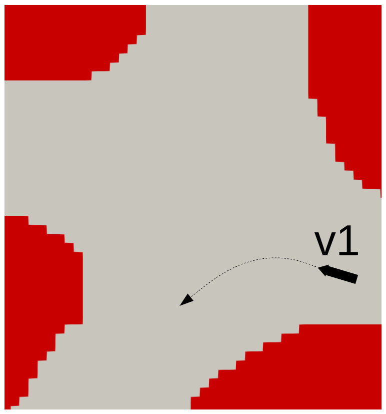


Figure 6.5: Problem settings for experiment 1.

6.4, such that it will stop before entering the unsafe zone (see Fig. 6.4) around the potential point of collision.

In contrast, pedestrian agents avoid collisions by adjusting both their speeds and directions, based on a slightly modified version of Orthogonal Reciprocal Collision Avoidance (ORCA) algorithm [26, 34]. Just as real pedestrians, the pedestrian agents prefer to maintain their preferred walking speed, if possible, by slightly changing their walking directions to avoid potential collisions. Since their speeds are low, pedestrian agents are allowed to stop or change their directions abruptly.

6.4 Validation of free flow speed profile and demonstrative examples

In this section, we present validation of the cubic polynomial approximations for vehicle speed profiles, and numerical experiments to qualitatively demonstrate the implemented agent system for simulating vehicle-vehicle and vehicle-pedestrian interactions at unsignalized junctions.

In all of the numerical examples, reaction time (i.e. τ) and preferred speed of all the vehicle agents are set to 0.67 second [27] and 45 km/h, respectively. To reduce the computations, instead of evaluating B-splines at every iteration step, vehicles were bound to their trajectories by making them pass through 6 equally spaced points on the corresponding B-splines. We set $V_{approach} = 30$ km/h, $V_{min} = 20$ km/h, and $V_{depart} = 30$ km/h in all experiments, except the simulation with a single vehicle agent.

6.4.1 Vehicle-vehicle interaction

In order to demonstrate the vehicle-vehicle interaction model presented in section 3, we conducted four numerical experiments, as shown in Fig. 6.9, 6.10, 6.11, 6.12. The first case is to validate the free-flow speed profile, while the rest are to demonstrate interactions among different number of vehicle agents.

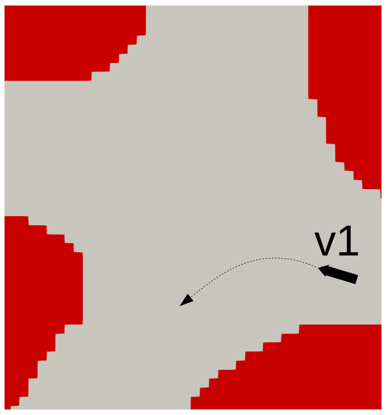


Figure 6.6: Problem settings for experiment 2.

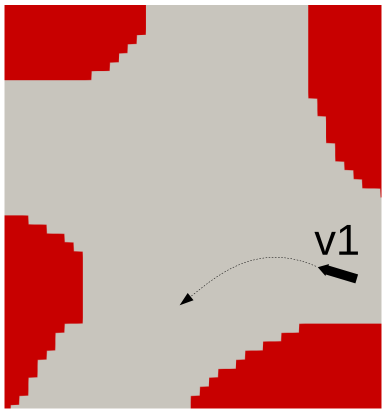


Figure 6.7: Problem settings for experiment 3.

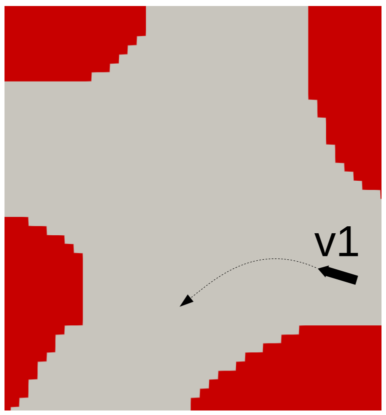


Figure 6.8: Problem settings for experiment 4.

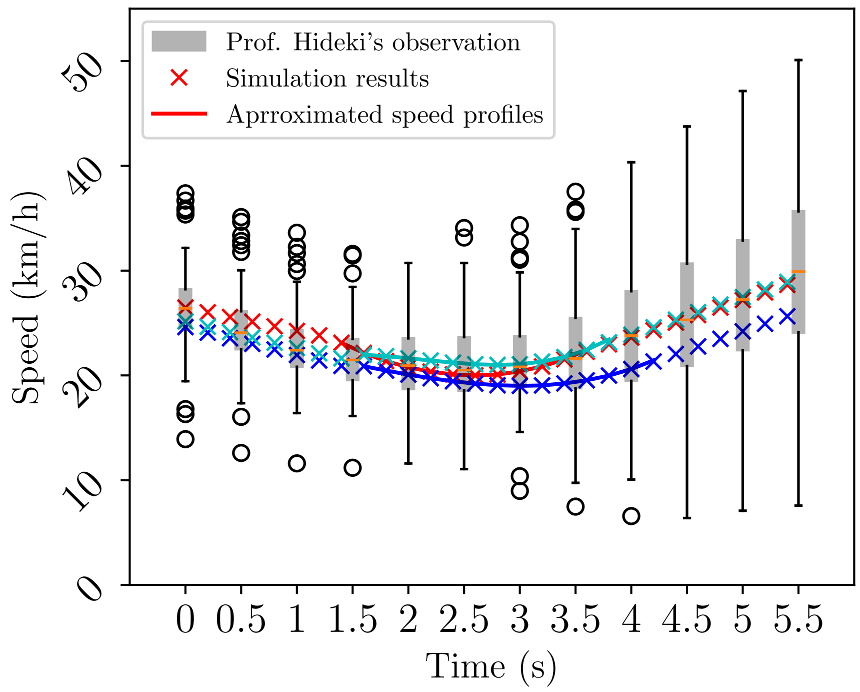


Figure 6.9: Speed profile of experiment 1, and field observations by Prof. Hideki Nakamura.

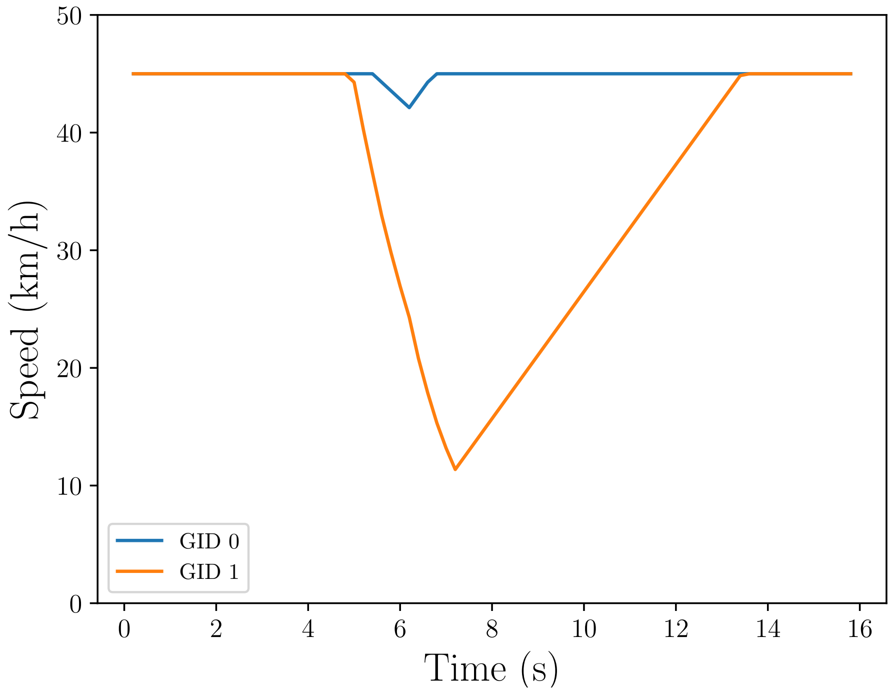


Figure 6.10: Speed profiles of experiment 2

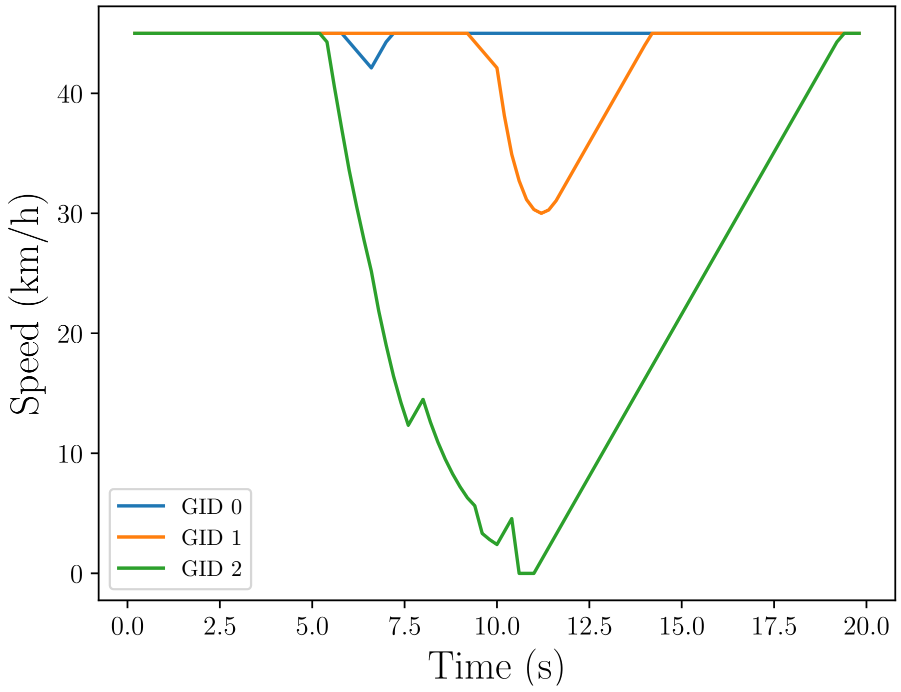


Figure 6.11: Speed profiles of experiment 3

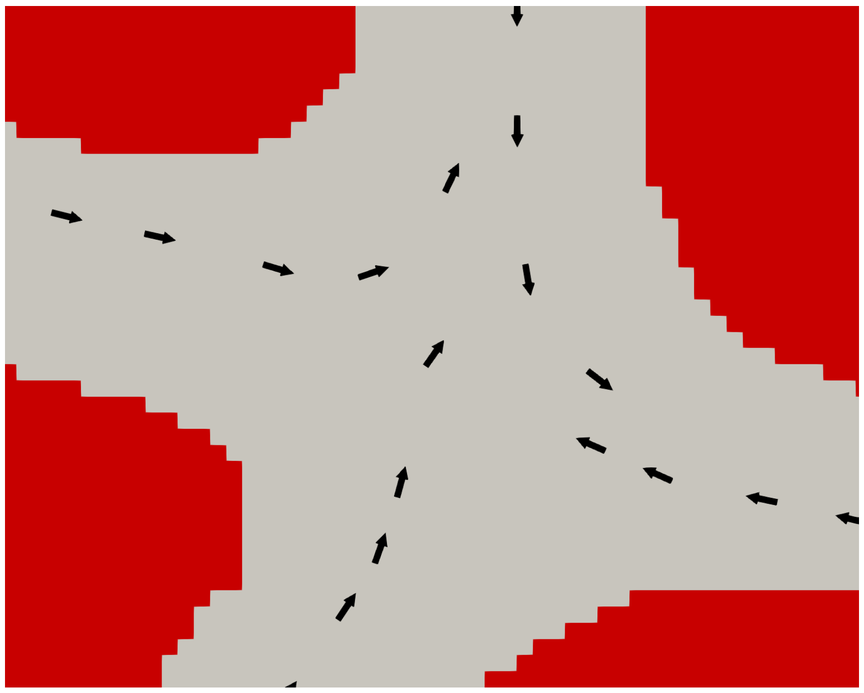


Figure 6.12: Screenshot of the simulation multiple vehicle agents

Case number	$V_{approach}$	V_{min}	V_{depart}
1	24	20	25
2	21	19	23
3	22	21	26

Table 6.1: $V_{approach}$, V_{min} and V_{depart} used for the experiment with the single vehicle agent. Units are in km/h.

6.4.1.1 Free flow of a single vehicle agent

The objective of this set of simulation is to validate our approximation for the free flow speed profile of a vehicle agent moving along a curved trajectory (see section 6.1.2), by comparing with observations by Prof. Hideki Nakamura. Three free flow experiments were conducted using three different sets of values for $V_{approach}$, V_{min} and V_{depart} given in Table 1.

Figure 6.9 shows the corresponding polynomial approximations for speed profiles, and the agent's speed from the developed agent-based model. The plot box-and-whisker diagrams in Fig. 6.9 are generated using the observations of left-turning vehicles at Suemori junction, Nagoya. This dataset provided by Prof. Hideki Nakamura, at Nagoya University, consists of observations of more than 100 left turning vehicles. We observed that the simulated result (Fig. 6.9) and observation data [10] are in good agreement, indicating our third order polynomial is a reasonable approximation for speed profile along curved trajectories. Further comparisons are necessary to validate our cubic polynomial approximation for u-turns or junctions with small acute angles.

6.4.1.2 Interactions of multiple vehicle agents

In the case of multiple interacting vehicles, we could not find field observations to compare with. The most related works are found in [23]. In their work, Mandiau et al. [23] have simulated two and three vehicles on conflicting trajectories at junction using a game theoretic model. Though most of the settings are similar to our problem setting shown in Fig. **??** and Fig. **??**, they have not provided necessary initial conditions to make a detailed comparison.

As seen in the Fig. 6.10, the two vehicles agents resolve the conflicts by giving priority to a first arriving agent with ID 1, and slightly decelerating the agent with ID 0. Qualitatively, these speed profiles are in good agreement with that of Mandiau et al. [23].

Figure 6.11 shows the speed profiles for the case with three vehicles. In this case, the conflicts are solved by giving priority to the agent with ID 0 which shows a slight deceleration. A soon as vehicle 0 moves out from the conflicting zone, vehicle 1 accelerates since its receives the priority; by the time vehicle 0 moves away, the speed of vehicle 1 is much higher than that of vehicle 2. This simulation shows that the giving priority to the first to arrive at the point of collision can resolve the conflicts at junctions. In case if several vehicles can simultaneously arrive at the collision point, those are assigned priorities according to a random process. When compared with corresponding Mandiau et al. results, we observe that both the simulations have similar patterns. However, their game theoretic based model makes all the three agents stop, and take a longer time for the three cars to clear the junctions.

The many vehicle agent case shown in Fig. **??** tests a more realistic scenario with many conflicts. In this case, the right turning vehicles had an uninterrupted flow just as in a real junction, while other three queues cleared the junction one after another. Though it could resolve all the conflicts and made all the agents clear the junction in a shorter time, giving priority to all the vehicles in a single queue is unfair and unrealistic. When there are long queues at a junction, a suitable probabilistic estimation to switch the priorities among the queues in a fair manner.

6.4.2 Vehicle-pedestrian interaction

The objective of this numerical experiment is to demonstrate that the developed agent-based model can resolve the conflicts in the presence of number of pedestrians. We used only 8 pedestrians, arranged in a simple geometric setting, so that one can easily assess the quality of the outcome based on his own expectations. Three vehicles spaced at distances to produce free-flow speeds are set to enter from the lower limb of the junction as shown in Fig. 6.13. To mimic an emergency evacuation scenario, we did not restrict the pedestrian agents to pedestrian crossings. The 8 pedestrians were set southbound to meet with three northbound right turning vehicles at an unsignalized junction. All pedestrian agents (ID 0-7) are represented by circles and all vehicle agents (ID 8-10) are represented by rectangles.

The resulting speed profiles of the agents are shown in Fig. 6.14. The vehicle agents have produced remarkably smooth speed profiles even if their trajectories are obstructed by a number of pedestrian agents. In section 4, rules of vehicle agents are set to decelerate to stop when collision with pedestrian agents is detected. As is seen in Fig. 6.14, this rule does not really make each car agent decelerate until its speed is zero. Under the fear of collision, a vehicle agent applies brakes according to Eq. 6.4 as long as it detects collision with a pedestrian. As soon its path becomes clear, it resumes under free-flow conditions. This demonstrates that Eq. 6.4 is not too restrictive.

Figure 6.15 to 6.20 show snapshots of the locations and speeds of each agent. The number and arrow associated with each agent indicate its speed and velocity, respectively. As seen in Fig. 6.14, vehicle agent with ID 8 predicted it had a collision-free trajectory, and passed the junction at free-flow speed (see Fig. 6.15). The agent with ID 9 predicted that it may collide with some pedestrians while reducing the speed for turning right. Therefore, the agent 9 reduced its speed at $t = 8.4$ s as seen in Fig. 6.17 so that it could avoid the collision. While agent with GID 9 was reducing its speed, the corresponding pedestrian also changed its direction such that they will never collide with ID 9 (see Fig. 6.18). Therefore, at $t = 10 seconds$, the vehicle agent ID 9 resumed at its preferred speed. The last vehicle agent with ID 10 predicted that it will collide with pedestrian agents. Hence, it reduced its speed to almost match with pedestrians' speed, and pedestrian agents could safely avoid the collision as seen in Fig. 6.19. Finally, the vehicle agent with ID 10 predicted that it would not collide with the pedestrian in the second row and resume at its preferred speed.

As demonstrated in this simple example, the developed agent-based model is capable of simulating unsignalized junctions with many pedestrian-vehicle interactions. We plan to compare the simulations with real observations to identify potential further improvements.

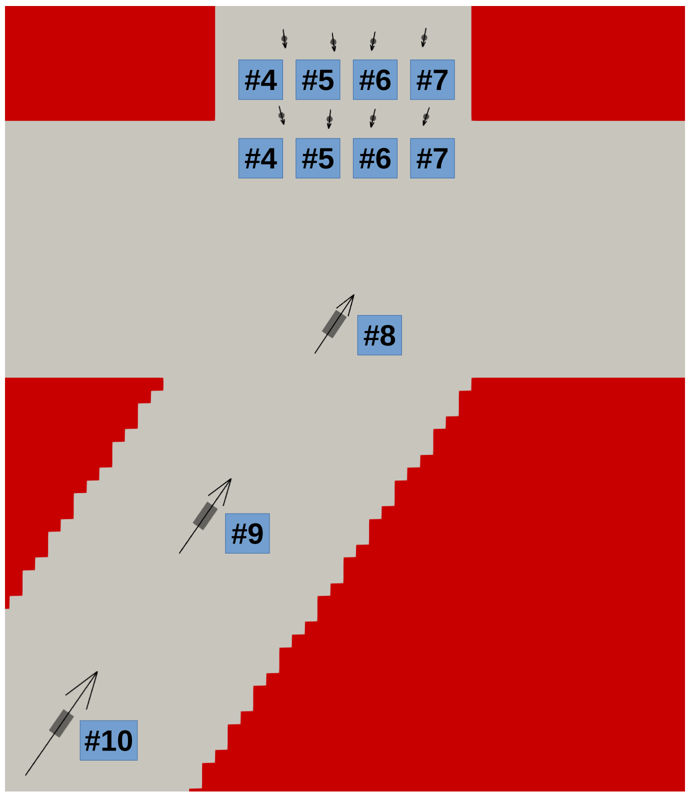


Figure 6.13: The problem settings for the pedestrian-car mixed-mode interaction experiment.

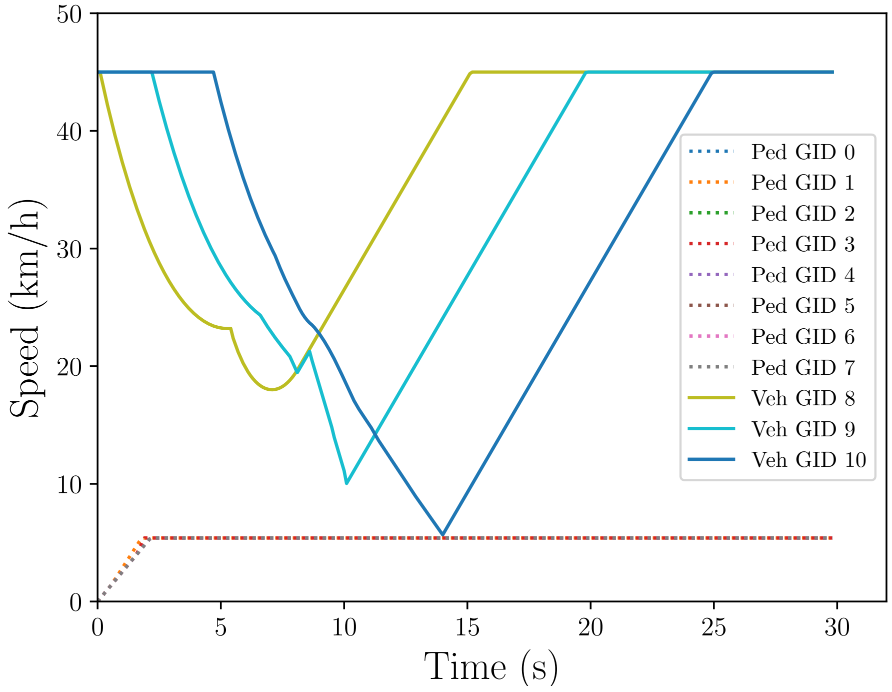


Figure 6.14: Resulting speed profiles of agents, for the pedestrian-car mixed-mode interaction experiment.

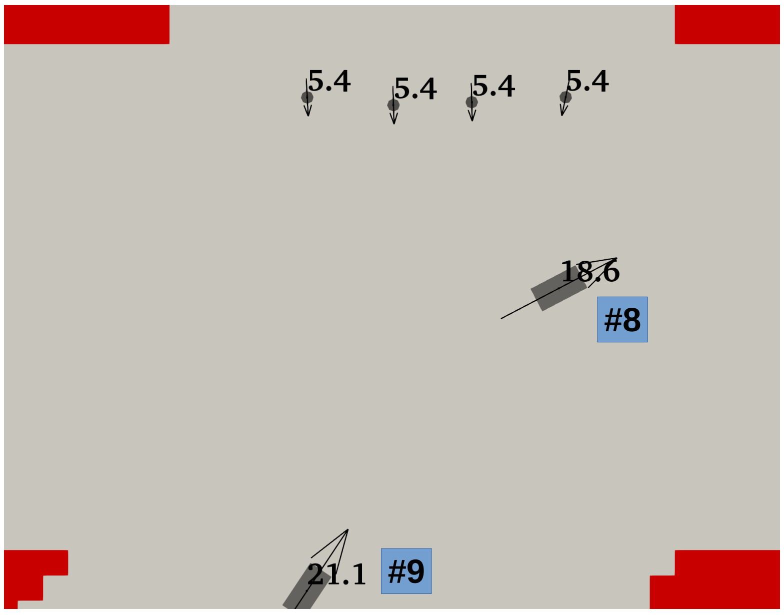


Figure 6.15: Snapshots of the car and pedestrian mixed experiment at $t = 7.7$ s

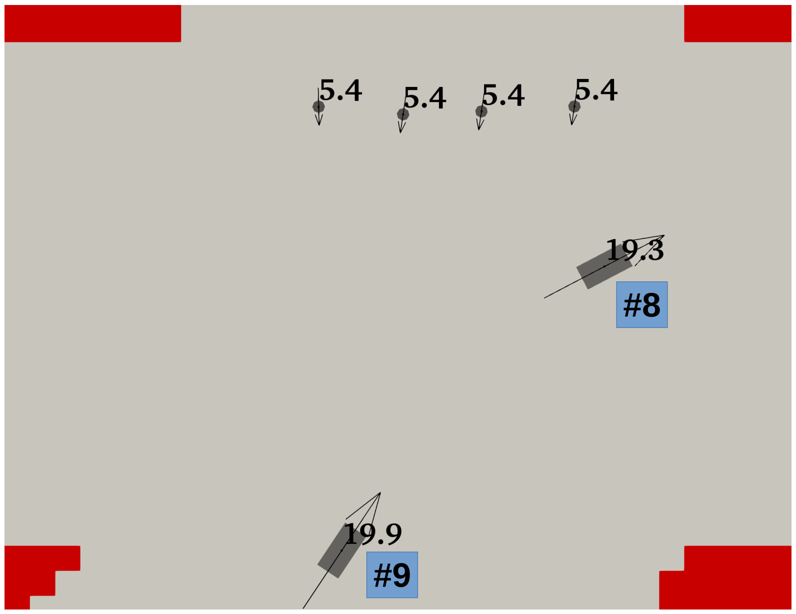


Figure 6.16: Snapshots of the car and pedestrian mixed experiment at $t = 8.4$ s

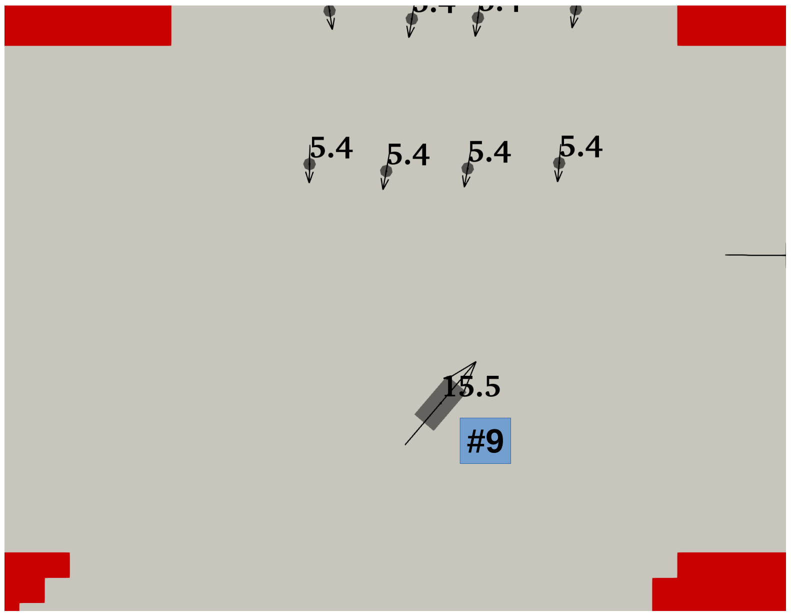


Figure 6.17: Snapshots of the car and pedestrian mixed experiment at $t = 9.4$ s

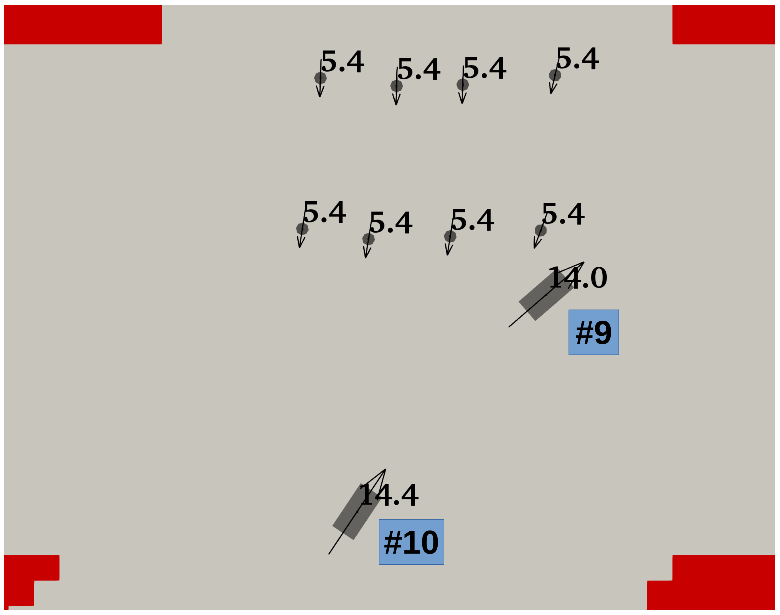


Figure 6.18: Snapshots of the car and pedestrian mixed experiment at $t = 11.2$ s

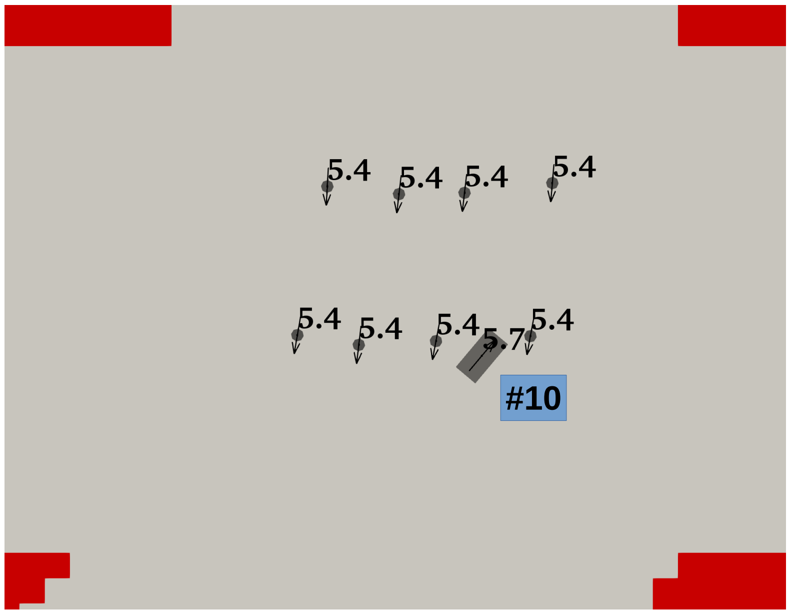


Figure 6.19: Snapshots of the car and pedestrian mixed experiment at $t = 14.0$ s

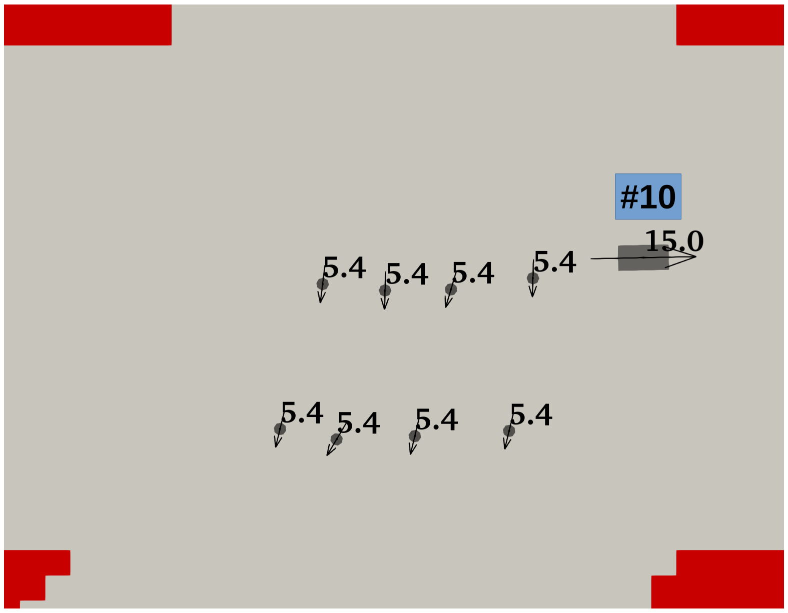


Figure 6.20: Snapshots of the car and pedestrian mixed experiment at $t = 16.6$ s

Chapter 7

Concluding Remarks

In this thesis, we present methodologies to improve a macroscopic traffic simulator and microscopic traffic simulator so that it can be used efficiently for disaster management application within a sufficiently short time.

In a macroscopic traffic simulator chapter, we presented a distributed memory parallel implementation of day-to-day traffic assignment algorithm (D2DTA) which is being developed with the aim of finding near-optimal traffic assignment for large networks. It is demonstrated that embarrassingly parallel pathfinding does not only substantially improve pathfinding scalability, but it is also several orders of magnitudes faster compared to parallel pathfinding. The use of only the active sub-network in early iterations greatly reduced the computational time of LTM based traffic flow simulation; nearly 20 times faster than using the full network. Further, we presented a link execution sequence which significantly reduces the number of iterations required to complete LTM based traffic flow simulations; it can reduce the traffic flow simulation time around 30%.

The remaining bottleneck in a macroscopic traffic simulator is the scalability of traffic flow simulator. There is a big imbalance of execution time of each MPI processor in each iteration. The imbalance of execution time is caused by the difficulty of assigning an equal workload to each MPI processor. Therefore, in the future, the dynamic load balancer technique could be applied. This technique is promising for reducing the imbalance of runtime. In addition, we could redesign the forward wave propagation so that we can improve the scalability of our traffic simulator.

In a microscopic traffic simulator chapter, we present a basic idea of an agent-based model which we use as a traffic simulator. We explained the partition scheme of our agent-based traffic simulator and technique which is used to increase the parallel efficiency: communication hiding and dynamic load balancing. In a macroscopic traffic simulator, we improved the domain generator to improve the quality of a topological graph which improves the result of the simulation since the agents can make their decision more precisely. Because of enhanced environments, it enables us to apply high fidelity intersection model. Hence, we implement the traffic light control at the intersection and we formulate the high fidelity intersection behavior of vehicle agents.

The agent-based model for vehicle-vehicle and vehicle-pedestrian is formulated. This model can capture realistic delayed travel time in case of free flow traffic. In addition, this model can capture the realistic trajectories of vehicles at arbitrary junctions. The proposed model is theoretically faster than optimization based intersection behavior because we used only a simply physics equations.

Appendix A

Derivations of equations

A.1 Speed profile approximation

We have discussed roughly about the approximated speed profile which is used to navigate the vehicle agents in section **??**. In this appendix, we aim to provide the full details of the mentioned equation. Firstly, we represent the speed profile by using a cubic polynomial equation as following:

$$v(x) = ax^3 + bx^2 + cx + d.$$

Variable x is the distance a vehicle agent traveled from an entering point of the trajectory curve. As discussed in section **??**, we used 4 initial conditions as following:

$$v(0) = d = V_{approach},$$

$$v(\frac{L}{2}) = \frac{aL^3}{8} + \frac{bL^2}{4} + \frac{cL}{2} + d = V_{min},$$

$$v(L) = aL^3 + bL^2 + cL + d = V_{depart}$$

, and

$$\frac{dv(x)}{dx}|_{x=\frac{L}{2}} = 3\frac{L^2}{4}a + 2Lb + c = 0.$$

variable L which is used in all equation are the length of trajectory which is shown in Fig. 6.1. The first initial condition means that the vehicle agent approach the intersection with an approach velocity ($V_{approach}$). The second initial condition means that the lowest speed of turning vehicle at the middle of trajectory is minimum speed (V_{min}). The third initial condition means that the vehicle depart the intersection with a depart velocity (V_{depart}). The last initial condition is that the minimum point of speed profile has zero slope. After combining initial condition equations, we obtain a system of linear equations.

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ \frac{L^3}{8} & \frac{L^2}{4} & \frac{L}{2} & 1 \\ L^3 & L^2 & L & 1 \\ \frac{3L^2}{4} & 2L & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} V_{approach} \\ V_{min} \\ V_{depart} \\ 0 \end{bmatrix}$$

After solving the system of linear equations, we obtain the coefficient of the system as following:

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} \frac{4.0*(-V_{approach}+V_{depart})}{L^3} \\ \frac{4.0*(2.0*V_{approach}-V_{min}-V_{depart})}{L^2} \\ \frac{-5.0V_{approach}+4.0V_{min}+V_{depart}}{L} \\ V_{approach} \end{bmatrix}$$

.

The final form of the speed profile is as following:

$$v(x) = (-4V_{approach} + 4V_{depart})\left(\frac{x}{L}\right)^3 + (8V_{approach} - 4V_{depart})\left(\frac{x}{L}\right)^2 + (-5V_{approach} + 4V_{min} + V_{depart})\left(\frac{x}{L}\right) + V_{approach}, x \in [0, L]$$

Bibliography

[1] Leonel Aguilar, Maddegedara Lalith, Tsuyoshi Ichimura, and Muneo Hori. On the performance and scalability of an hpc enhanced multi agent system based evacuation simulator. *Procedia Computer Science*, 108:937 – 947, 2017. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.

[2] Wael K.M. Alhajyaseen, Miho Asano, Hideki Nakamura, and Dang Minh Tan. Stochastic approach for modeling the effects of intersection geometry on turning vehicle paths. *Transportation Research Part C: Emerging Technologies*, 32:179 – 192, 2013.

[3] A. Attanasi, E. Silvestri, P. Meschini, and G. Gentile. Real world applications using parallel computing techniques in dynamic traffic assignment and shortest path search. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 316–321, Sept 2015.

[4] P. Beeson, N. K. Jong, and B. Kuipers. Towards autonomous topological place detection using the extended voronoi graph. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 4373–4379, April 2005.

[5] RW BOHANNON. Comfortable and maximum walking speed of adults aged 20-79 years : Reference values and determinants. *Age Ageing*, 26:15–19, 1997.

[6] Ismail Chabini and Sridevi Ganugapati. Parallel algorithms for dynamic shortest path problems. *International Transactions in Operational Research*, 9(3):279–302, 2002.

[7] Stephanie E Chang and Nobuoto Nojima. Measuring post-disaster transportation system performance: the 1995 kobe earthquake in comparative perspective. *Transportation Research Part A: Policy and Practice*, 35(6):475 – 494, 2001.

[8] A. Colombo and D. Del Vecchio. Least restrictive supervisors for intersection collision avoidance: A scheduling approach. *IEEE Transactions on Automatic Control*, 60(6):1515–1527, June 2015.

[9] G. R. de Campos, P. Falcone, and J. Sjöberg. Autonomous cooperative driving: A velocity-based negotiation approach for intersection crossing. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 1456–1461, Oct 2013.

[10] Charitha Dias, Miho Iryo-Asano, and Takashi Oguchi. Predicting optimal trajectory of left-turning vehicle at signalized intersection. *Transportation Research Procedia*, 21:240 – 250, 2017. International Symposia of Transport Simulation (ISTS) and the International Workshop on Traffic Data Collection and its Standardization (IWTDCS).

[11] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, December 1959.

[12] A. Doniec, R. Mandiau, S. Piechowiak, and S. Espié. Anticipation based on constraint processing in a multi-agent context. *Autonomous Agents and Multi-Agent Systems*, 17(2):339–361, Oct 2008.

[13] Arnaud Doniec, René Mandiau, Sylvain Piechowiak, and Stéphane Espié. A behavioral multi-agent model for road traffic simulation. *Engineering Applications of Artificial Intelligence*, 21(8):1443 – 1454, 2008.

[14] Terry Friesz, David Bernstein, Tony Smith, Roger Tobin, and Byung-Wook Wie. A variational inequality formulation of the dynamic network user equilibrium problem. 41:179–191, 02 1993.

[15] Y. Fu, C. Li, B. Xia, W. Dong, Y. Duan, and L. Xiong. A novel warning/avoidance algorithm for intersection collision based on dynamic bayesian networks. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2016.

[16] M. R. Hafner, D. Cunningham, L. Caminiti, and D. Del Vecchio. Cooperative collision avoidance at intersections: Algorithms and experiments. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1162–1175, Sept 2013.

[17] Michelle R Hribar, Valerie E Taylor, and David E Boyce. Termination detection for parallel shortest path algorithms. *Journal of Parallel and Distributed Computing*, 55(2):153 – 165, 1998.

[18] Michelle R Hribar, Valerie E Taylor, and David E Boyce. Implementing parallel shortest path for parallel transportation applications. *Parallel Computing*, 27(12):1537 – 1568, 2001. Applications of parallel computing in transportation.

[19] Takamasa Iryo. Day-to-day dynamical model incorporating an explicit description of individuals' information collection behaviour. *Transportation Research Part B: Methodological*, 92:88 – 103, 2016. Special issue Day-to-Day Dynamics in Transportation Networks.

[20] N. Jasika, N. Alispahic, A. Elma, K. Ilvana, L. Elma, and N. Nosovic. Dijkstra's shortest path algorithm serial and parallel execution performance analysis. In *2012 Proceedings of the 35th International Convention MIPRO*, pages 1811–1815, May 2012.

[21] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, December 1998.

[22] D. Lasalle and G. Karypis. Multi-threaded graph partitioning. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 225–236, May 2013.

[23] René Mandiau, Alexis Champion, Jean-Michel Auberlet, Stéphane Espié, and Christophe Kolski. Behaviour based on decision matrices for a coordination between agents in a urban traffic simulation. *Applied Intelligence*, 28(2):121–138, Apr 2008.

[24] Leonel Enrique Aguilar Melgar, Wijerathne Maddegedara Lalith Lakshman, Muneo Hori, Tsuyoshi Ichimura, and Seizo Tanaka. On the development of an mas based evacuation simulation system: Autonomous navigation and collision avoidance. In Guido Boella, Edith Elkind, Bastin Tony Roy Savarimuthu, Frank Dignum, and Martin K. Purvis, editors, *PRIMA 2013: Principles and Practice of Multi-Agent Systems*, pages 388–395, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[25] Leonel Enrique Aguilar Melgar, Maddegedara Lalith, Muneo Hori, Tsuyoshi Ichimura, and Seizo Tanaka. A scalable workbench for large urban area simulations, comprised of resources for behavioural models, interactions and dynamic environments. In Hoa Khanh Dam, Jeremy Pitt, Yang Xu, Guido Governatori, and Takayuki Ito, editors, *PRIMA 2014: Principles and Practice of Multi-Agent Systems*, pages 166–181, Cham, 2014. Springer International Publishing.

[26] Leonel Enrique Aguilar Melgar, Maddegedara Lalith, Muneo Hori, Tsuyoshi Ichimura, and Seizo Tanaka. A scalable workbench for large urban area simulations, comprised of resources for behavioural models, interactions and dynamic environments. In Hoa Khanh Dam, Jeremy Pitt, Yang Xu, Guido Governatori, and Takayuki Ito, editors, *PRIMA 2014: Principles and Practice of Multi-Agent Systems*, pages 166–181, Cham, 2014. Springer International Publishing.

[27] Charles Arthur Nagler and William Merle Nagler. Reaction time measurements. *Forensic Science*, 2:261 – 274, 1973.

[28] Eoin A. O'Cearbhaill and Margaret O'Mahony. Parallel implementation of a transportation network model. *Journal of Parallel and Distributed Computing*, 65(1):1 – 14, 2005.

[29] Zuo-Jun Max Shen, JyothsnaRai Pannala, Rohit Rai, and Tsz Shing Tsoi. Modeling transportation networks during disruptions and emergency evacuations. *UC Berkeley: University of California Transportation Center*, 2008.

[30] M. J. Smith and M. B. Wisten. A continuous day-to-day traffic assignment model and the existence of a continuous dynamic user equilibrium. *Annals of Operations Research*, 60(1):59–79, Dec 1995.

[31] S. Thulasidasan and S. Eidenbenz. Accelerating traffic microsimulations: A parallel discrete-event queue-based approach for speed and scale. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pages 2457–2466, Dec 2009.

[32] S. Thulasidasan, S. P. Kasiviswanathan, S. Eidenbenz, and P. Romero. Explicit spatial scattering for load balancing in conservatively synchronized parallel discrete event simulations. In *2010 IEEE Workshop on Principles of Advanced and Distributed Simulation*, pages 1–8, May 2010.

[33] Jur van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In Cédric Pradalier, Roland Siegwart, and Gerhard Hirzinger, editors, *Robotics Research*, pages 3–19, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[34] Jur van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In Cédric Pradalier, Roland Siegwart, and Gerhard Hirzinger, editors, *Robotics Research*, pages 3–19, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[35] J G WARDROP. Road paper. some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers*, 1(3):325–362, 1952.

[36] M. L. L. Wijerathne, Muneo Hori, Tsuyoshi Ichimura, and Seizo Tanaka. Parallel scalability enhancements of seismic response and evacuation simulations of integrated earthquake simulator. In Michel Daydé, Osni Marques, and Kengo Nakajima, editors, *High Performance Computing for Computational Science - VECPAR 2012*, pages 105–117, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[37] M.L.L. Wijerathne, L.A. Melgar, M. Hori, T. Ichimura, and S. Tanaka. Hpc enhanced large urban area evacuation simulations with vision based autonomously navigating multi agents. *Procedia Computer Science*, 18:1515 – 1524, 2013. 2013 International Conference on Computational Science.

[38] Isaak Yperman. The link transmission model for dynamic network loading. *PhD dissertation, Katholieke Universiteit Leuven*, 2007.