SMARTS: Scalable Microscopic Adaptive Road Traffic Simulator

Kotagiri Ramamohanarao, University of Melbourne
Hairuo Xie, University of Melbourne
Lars Kulik, University of Melbourne
Shanika Karunasekera, University of Melbourne
Egemen Tanin, University of Melbourne
Rui Zhang, University of Melbourne
Eman Bin Khunayn, University of Melbourne

Microscopic traffic simulators are important tools for studying transportation systems as they describe the evolution of traffic to the highest level of detail. A major challenge to microscopic simulators is the slow simulation speed due to the complexity of traffic models. We have developed SMARTS, a distributed microscopic traffic simulator that can utilize multiple independent processes in parallel. SMARTS can perform fast large-scale simulations. For example, when simulating one million vehicles in an area the size of Melbourne, the system runs 1.14 times faster than real time with 30 computing nodes and 0.2 second simulation time step. SMARTS supports various driver models and traffic rules, such as car-following model and lane-changing model, which can be driver dependent. It can simulate multiple vehicle types, including bus and tram. The simulator is equipped with a wide range of features that help to customize, calibrate and monitor simulations. Simulations are accurate and confirm with real traffic behaviours. For example, it achieves 79.1% accuracy in predicting traffic on a 10-kilometre freeway 90 minutes into the future. The simulator can be used for predictive traffic advisories as well as traffic management decisions as simulations complete well ahead of real time. SMARTS can be easily deployed to different operating systems as it is developed with the standard Java libraries.

CCS Concepts: • **Computing methodologies** → **Distributed simulation;**

General Terms: Design, Performance

Additional Key Words and Phrases: Microscopic Traffic Simulation, Distributed Computing

ACM Reference Format:
Kotagiri Ramamohanarao, Hairuo Xie, Lars Kulik, Shanika Karunasekera, Egemen Tanin, Rui Zhang, and Eman Bin Khunayn. 2016. SMARTS: Scalable Microscopic Adaptive Road Traffic Simulator. *ACM Trans. Intell. Syst. Technol.* V, N, Article A (January YYYY), 20 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

With the increasing complexity of road infrastructures and the high demand for road use, many cities around the world are facing significant traffic problems. Traffic simulators are software programs that can help to address these problems. With the capability of describing the evolution of traffic, traffic simulators are widely used in traffic forecast, traffic model development and traffic visualization. Compared with other research methods, such as collecting real trajectory data, traffic simulations have several advantages. Simulations can be performed for highly customized scenarios. For example, one can use traffic simulations to study the impact of changes in a city's setting such as the addition of a new road [Zheng et al. 2014]. As another example, one can use simulations to find how long it would take an emergency vehicle, e.g., an ambulance or police car, to cover a trajectory under certain road traffic conditions. Simulations can be used to predict traffic in the future, which

Author's addresses: Kotagiri Ramamohanarao and Hairuo Xie and Lars Kulik and Shanika Karunasekera and Egemen Tanin and Rui Zhang and Eman Bin Khunayn, Department of Computing and Information Systems, University of Melbourne, Australia.

DOI: http://dx.doi.org/10.1145/0000000.0000000

can be difficult to do by only processing trajectory data from real vehicles. Simulations can also generate a massive volume of realistic synthetic trajectories for a wide range of purposes, such as trajectory data mining [Zheng 2015; Wang et al. 2015]. Generating trajectory data from simulations can be significantly easier than collecting real data, which is highly costly in many scenarios.

Traffic simulations can be performed at three levels of detail. The most detailed ones are microscopic simulations [Yang and Koutsopoulos 1996; Ehlert and Rothkrantz 2001; Krajzewicz et al. 2002; Hidas 2002; Barceló et al. 2005], where individual elements in transportation systems, e.g., vehicles, are modelled. The less detailed ones are mesoscopic simulations, which represent traffic as platoons of homogeneous vehicles. Macroscopic simulations are more abstract as they focus on aggregated status of traffic. There are also hybrid simulations that mix some of the types. We are interested in microscopic simulations, which provide the best analytical tools for the aforementioned applications, as they can describe traffic to the finest level of detail.

Unfortunately, microscopic simulations can be significantly slower than other simulations [Taori and Rathi 1996] because such simulations normally involve a large amount of mathematical computation based on complex models. When there are a large number of vehicles involved in a simulation, e.g., simulating one million vehicles in a large city, the simulation is highly likely to run slower than real time on a single processor. Our ultimate goal is to build a system that can provide guidance to real drivers based on highly complex simulations (called hyper-real simulations) using commodity hardware. The hyper-real simulations would model variances between individual drivers for a comprehensive range of behaviours. The computation workload of such simulations would be even higher than traditional microscopic simulations. In order to provide useful guidance, the system must be able to predict the traffic well ahead of real time. Given the specifications of the existing commodity processors, it would be difficult to make such a system run faster than real time on a single processor for a large number of vehicles. Recent research is focused on speeding up traffic simulations using distributed processors that are connected to the same network [Nagel and Rickert 2001; Klefstad et al. 2005]. By utilizing the computing power of multiple processors simultaneously, it would be relatively easy to multiply the computing power of the whole system, allowing fast large-scale simulations. Therefore, we are interested in distributed traffic simulations.

We have developed a *Scalable Microscopic Adaptive Road Traffic Simulator*, which we call **SMARTS**. In contrast to many existing traffic simulators, our simulator is built on a distributed architecture that enables simulations to run with multiple independent processes in parallel. This is key to achieve a high level of efficiency and scalability. A simulation area can be partitioned into multiple non-overlapping sub-areas. Simulation of traffic in different sub-areas can be performed at different processes. The processes run in parallel, resulting in a significant reduction of simulation time compared to using only one process. The simulator offers versatile functionalities with a high degree of calibration. SMARTS runs in discrete time steps. Traffic is simulated in a continuous spatial domain. To achieve a high level of cross-platform portability, all components of the system are implemented with the standard Java libraries. The simulator has been tested on Windows, Linux and Mac OS. A demo video of our simulator can be watched from [SMARTS Team 2015].

1.1. Insight into System Design

A challenge to designing a complex traffic simulations is generalizing the effects of heterogeneous factors on the movement of vehicles. There can be a large number of factors that constantly affect a vehicle's movement, such as front cars, conflicting traffic at intersections, traffic lights, etc. It is important to implement a mechanism that can evaluate the effects of all the factors and allow straight-forward extension of traffic rules and road facilities. We find that an effective way to address this challenge is converting the factors into *impeding objects* that can cause vehicles to slow down. For example, the impeding object converted from a red light is a stationary object. By doing this, we can compute the ideal acceleration of vehicles based on heterogeneous factors using car-following model. Another challenge is reducing communication costs between distributed processes. In order to maintain the correctness of simulation across different processes that simulate different areas, information of the vehicles near the border between adjacent areas needs to be exchanged between the respective processes, which can result in a significantly long period of time on communication. The system needs to minimize the communication cost. The key novelty of our work is a spatial workload balancing strategy that balances the number of vehicles in different processes while keeping the communication cost at a low level. We observe that the communication cost is generally proportional to the number of communication channels between the distributed processes. Based on this observation, we implement a workload balancing strategy that minimizes the number of communication channels, which can help to reduce communication costs. Our simulator can further reduce communication costs by utilizing a decentralized synchronization method, which can synchronize the simulation at different slave processes without a master process.

1.2. Efficiency and Scalability

SMARTS achieves a high level of efficiency and scalability. By exploiting the distributed computing architecture, the workload balancing strategy and the decentralized synchronization, SMARTS can perform fast large-scale simulations. For example, it can simulate traffic 1.14 times faster than real time with one million vehicles in an area of $600km^2$ when all the vehicles' positions are updated 5 times per second at 30 computing nodes. The simulator can run 2.72 times faster than real time when simulating $300,000$ vehicles if the vehicles are updated 5 times per second. The same simulation with $300,000$ vehicles can run 9.44 times faster than real time if the vehicles are updated once per second. As the simulations can run faster than real time, SMARTS can be used to predict traffic in the future.

1.3. Versatile Functionalities

SMARTS provides a comprehensive range of functionalities. It can simulate traffic in arbitrary road networks. It is possible to include any number of vehicle types in a simulation. Each vehicle follows a specific route plan, which can be imported or be generated by the simulator itself. A vehicle with user-defined route plan can start moving at a specific time and make temporary stops in its trip. SMARTS simulates real driver behaviour based on a car-following model and a lane-changing model. Traffic lights and various traffic rules are also implemented. For example, the simulator implements the local rules related to trams that share roads with ordinary vehicles such as cars and trucks. The simulator offers a *Graphical User Interface (GUI)* to visualize simulations. The GUI allows users to configure, monitor and control simulations. For example, it allows users to block traffic lanes during a simulation.

1.4. Flexibility and Adaptability

SMARTS is developed as a highly flexible simulator. It is adaptive to a diverse range of computing environments, from single computer to distributed computing nodes. It can simulate traffic of any scale, in terms of the size of simulation area and the number of vehicles. The simulator is also adaptive to various simulation scenarios as one can customize simulations by importing specific vehicles and making changes to many aspects of the simulations, such as routing algorithms, parameters of the implemented models and the timing strategy of traffic lights. The simulator also allows users to study the behaviour of a subset of the vehicles that move with a large volume of background traffic.

1.5. High Degree of Calibration

Calibration is important to realistic simulation. For example, recent research uses digital footprints on social networks to calibrate travel plan of people in traffic simulations [McArdle et al. 2014]. In SMARTS, users can calibrate driver models by adjusting the relevant parameters. With simple extensions, it is also possible to initialize traffic and adjust the influx of traffic based on live traffic data or historical traffic data.

1.6. Summary of Contributions

— A microscopic simulator that can simulate traffic at a large scale on any distributed network of computers. The simulator can run on common operating systems.

Table I. Comparison of Key Features of Traffic Simulators: SUMO [Krajzewicz et al. 2002], TRANSIMS [Nagel and Rickert 2001], VISSIM [PTV Group 2015], MATSIM [Waraich et al. 2015], ParamGrid [Klefstad et al. 2005], and SMARTS.

	SUMO	TRANSIMS	VISSIM	MATSIM	ParamGrid	SMARTS
Distributed computing	×	√	×	×	√	√
OpenStreetMap data	√	×	×	√	×	√
Continuous spatial automaton	√	×	√	√	√	√
Workload balancing	×	√	×	×	√	√
Decentralized synchronization	×	×	×	×	√	√
Cross-platform portability	×	×	×	√	×	√

— A versatile simulation platform that supports various traffic rules such as the rules related to trams that share roads with ordinary vehicles. The platform can also differentiate foreground traffic from background traffic to study various scenarios. The customizations of driver models, vehicle routes and many other aspects of simulation are also supported.
— A spatial workload balancing strategy that helps to reduce the computation time of the distributed processes. The strategy also minimizes the communication time by keeping the number of communication channels at a low level.
— A decentralized synchronization strategy to eliminate communication bottleneck at the master process.
— An approach for calibration of simulation to match real traffic data.

We present the related work in Section 2. Section 3 details the features of the simulator. We detail the distributed architecture, including the workload balancing strategy, in Section 4. The experimental results are shown in Section 5. We conclude the paper in Section 6.

2. RELATED WORK

There has been a significant body of work on transportation simulations. For example, SUMO [Krajzewicz et al. 2002] and MATSIM [Waraich et al. 2015] are two of the most prominent software suits for traffic simulations. These systems have a wide range of features, such as generating origin-destination matrix, estimating noise and emission, simulating inter-vehicle communication, etc. However, many existing traffic simulators lack certain features that our simulator is capable of. Table I shows a comparison between certain prominent traffic simulators and our simulator.

Different to some of the simulators, SMARTS is optimized for distributed simulation, which enables it to perform fast large-scale simulations and scale up easily. Some simulators do not support OpenStreetMap or can only display static map image based on OpenStreetMap. In contrast, SMARTS can build road network based on OpenStreetMap data, which provides low-cost map information across the world. Some existing simulators, such as TRANSIMS, use the cellular automata technique to describe the simulation space. Discrete description of space can help to improve simulation speed but may not be as realistic as the continuous description of space. SMARTS performs simulation in a continuous spatial domain. SMARTS is also more portable across different operating systems than certain simulators as it is developed with the standard Java libraries.

An important use of traffic simulators is to generate trajectory data. Research has been done on data generators that can output trajectories of moving objects. Düntgen et. al. [Düntgen et al. 2009] developed a moving object data generator based on certain vehicle movement patterns. For example, a vehicle accelerates if its distance to the end of the current road link is longer than a threshold and the vehicle's current speed is under a limit. The generator can also disturb the positions of vehicles to create realistic noisy data. Another prominent moving object data generator is developed by Brinkhoff [Brinkhoff 2002]. The generator computes routes of vehicles by considering the predicted road usage. Origins and destinations of routes can be generated with certain probability models. MNTG [Mokbel et al. 2013] is a web application that makes it more convenient to use the aforementioned generators. The application provides a web interface that allows users to submit the requirements of trajectory data. The application creates the trajectories in the back end using the existing generators and returns the results to the user. A major difference between these trajectory data generators and our simulator is that the vehicles in our simulator do not collide with each other based on the car-following model. In addition, the movement of vehicles in our simulator is significantly more fine-grained based on a number of traffic rules. We also simulate the lane-changing behaviour, which is not considered in the trajectory data generators.

Research has been done for workload balancing strategy as it is crucial to the performance of distributed traffic simulations. One direction in this area is functionality-based distribution. An example can be seen in SIMLAB [Yang and Koutsopoulos 1996], where the simulation system consists of several modules, which are responsible for updating traffic lights, generating routes, computing movement of vehicles and collecting simulated data. Different modules can run on different computers at the same time. This type of balancing may not work well when the workload of certain modules is significantly higher than the workload of other modules. Different to this approach, most of the existing distributed traffic simulators divide workload based on the spatial partition of road network. For example, ParamGrid [Klefstad et al. 2005] partitions a simulation area into equal-sized rectangular sub-areas. Different processes simulate traffic in different sub-areas. The processes can run on different machines in parallel. The problem with this approach is that the workload can be unbalanced due to the difference of traffic densities between the sub-areas. TRANSIMS [Nagel and Rickert 2001] partitions the simulation space based on orthogonal recursive splitting of space. The strategy can achieve a better workload balancing but the respective area of a process may be adjacent to the areas of a number of other processes. Consequently, a process may have to maintain a high number of communication channels, which may result in a high level of communication cost. Different to this approach, our workload balancing strategy can keep the number of communication channels at a low level regardless of the complexity of road network.

Similar to other distributed traffic simulators, SMARTS is based on a master-slave model. In order to synchronize the simulation at different slave processes, most of the existing simulators require that all the slave processes communicate with the master process at each time step, e.g., [Nagel and Rickert 2001]. This centralized approach can potentially cause communication bottleneck at the master process. In contrast, ParamGrid uses decentralized synchronization such that each slave process only needs to communicate with other slave processes for synchronization [Klefstad et al. 2005]. However, the system requires each slave process to synchronize with all other slave processes. This can still incur a high level of communication cost when there are a large number of slave processes. The implementation of decentralized synchronization in SMARTS only requires a slave process to synchronize with its neighbour processes that share one or more road links with it. This can be significantly more efficient than the existing decentralized approach.

Some traffic simulators are optimized for parallel computing with multiple threads on single computer [Barceló et al. 2005; Lee and Chandrasekar 2002]. Our simulator is designed for distributed environment but it can also be used on single computer. When SMARTS runs on a computer with multiple CPU cores, it can exploit multiple cores with multiple slave processes.

3. SIMULATION FEATURES

The simulator implements a comprehensive set of features (Figure 1). We divide the features into three categories, *input*, *simulation* and *output*. We detail these features in this section.

3.1. Input

3.1.1. Road Network. Simulated vehicles travel on a road network that consists of nodes and edges. A node is connected with a certain number of *outward* edges that start from the node and a certain number of *inward* edges that end at the node. An edge is of a particular type, such as *primary* or *motorway*, as specified in the map data. An edge can contain multiple lanes, each of which can be occupied by vehicles. Figure 2 shows the relationship between nodes, edges, lanes and vehicles. The current implementation creates a constant number of lanes for all the edges.

Nodes and edges can be extracted from OpenStreetMap data, which can be loaded from an external file during the initialization of simulation. Although OpenStreetMap data supports a large

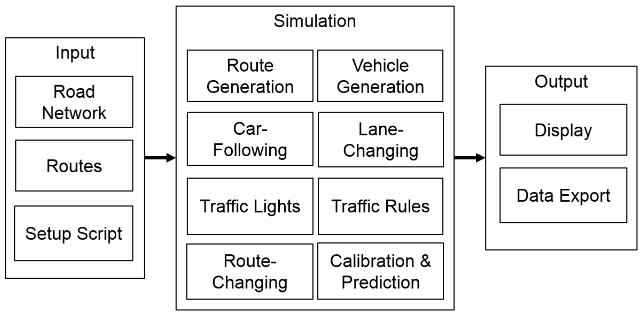


Fig. 1. Feature modules of SMARTS.

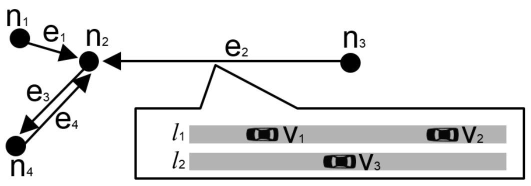


Fig. 2. Example of nodes (n_1 to n_4), edges (e_1 to e_4), lanes (l_1 to l_2) and vehicles (v_1 to v_3). Arrows indicate traffic direction.

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version='0.6'>
  <node id='1' lat='-37.8209' lon='144.9616' />
  <node id='2' lat='-37.8210' lon='144.9618' />
  <node id='3' lat='-37.8210' lon='144.9621' />
  <way id='1'>
    <nd ref='1' />
    <nd ref='2' />
    <nd ref='3' />
    <tag k='highway' v='primary' />
    <tag k='name' v='Road A' />
  </way>
</osm>
```

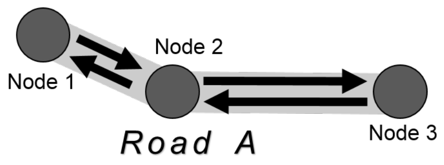


Fig. 3. Example of OpenStreetMap data and the corresponding road network.

number of road types, SMARTS can extract roads belonging to a subset of the types. We show an example of OpenStreetMap data in Figure 3. The left sub-figure shows the format of the data, where a *node* element corresponds to a node in the road network and a *way* element corresponds to a series of edges connecting with the nodes. The right sub-figure shows the corresponding road network based on the data.

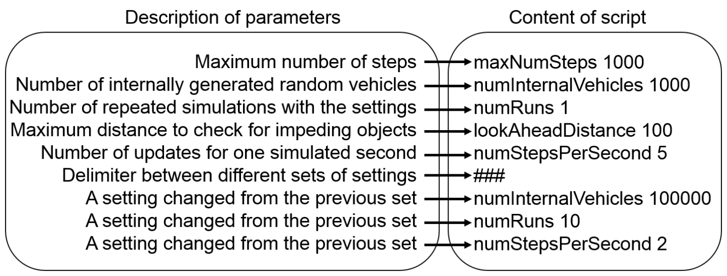
3.1.2. Routes. Routes of buses and trams can be extracted from OpenStreetMap data. User-defined route plan can also be loaded into the simulator during the initialization of simulation. The user-defined route plan can contain the routes of any number of vehicles. For each vehicle, the plan specifies the start time of the vehicle, the type of the vehicle, the ID of the vehicle and the sequence of the nodes that the vehicle will visit. In addition, the route plan can contain the time duration that the vehicle needs to stop at specific intermediate nodes. An example route file is shown in Figure 4. Note that the routes can also be generated by the simulator itself during simulations. This is detailed in Section 3.2.1.

3.1.3. Setup Script. SMARTS can automatically run multiple simulations in a sequence based on a user-defined script. This enables users to configure a large number of simulations beforehand. Figure 5 shows some of the settings that can be configured in a script. As the example shows, the same simulation can be repeated for any number of times as defined by the *numRuns* parameter.

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
<vehicle id="V_1" type="TRUCK" start_time="100">
<node id="1492152182"/>
<node id="2189145388" stop_duration="20"/>
<node id="2189145386"/>
<node id="2181275327"/>
<node id="2181275333" stop_duration="30.5"/>
<node id="2181275339"/>
</vehicle>
</data>
```

Fig. 4. An example route file. It shows the route of a truck named V_1. The truck will start its trip at the 100th second. It will make temporary stops, one for 20 seconds and another for 30.5 seconds, at two intermediate nodes.

From the second set of settings, we only need to specify the parameters that are changed from the previous set. This makes it convenient to configure a large number of similar simulations.



Description of parameters	Content of script
Maximum number of steps	maxNumSteps 1000
Number of internally generated random vehicles	numInternalVehicles 1000
Number of repeated simulations with the settings	numRuns 1
Maximum distance to check for impeding objects	lookAheadDistance 100
Number of updates for one simulated second	numStepsPerSecond 5
Delimiter between different sets of settings	###
A setting changed from the previous set	numInternalVehicles 100000
A setting changed from the previous set	numRuns 10
A setting changed from the previous set	numStepsPerSecond 2

Fig. 5. An example setup script. The script is shown on the right. The description of each row of the script is shown on the left.

3.2. Simulation

3.2.1. Route Generation. SMARTS can load pre-defined routes at simulation initialization (Section 3.1.2). Our simulator has a complementary feature that generates routes internally. The origin and destination of an internally-generated route can be randomly picked by the simulator. Users can also restrict origins and destinations to specific areas using the GUI. The routes can be computed with certain commonly-used routing algorithms. One of them is Dijkstra's algorithm [Dijkstra 1959], which generates the shortest path from an origin to a destination. Another is an extended version of the Overdo A* algorithm [Jacob et al. 1999]. Our algorithm generates a route, which is close to the optimal shortest path, while reducing the possibility that the route goes through congested areas. The algorithm has two unique characteristics. First, it can diversify the traffic flows by using a randomization factor. Given an intermediate node on a route, the algorithm adds the randomization factor to the travel cost of each edge that goes out of the node. The algorithm prioritizes the edges with the lowest randomized cost when finding the route. If the range of the randomization factor is properly set, the randomized path can slightly deviate from the shortest path, which helps to reduce traffic congestions when a large number of vehicles have similar origins and destinations [Nguyen et al. 2015]. Second, with simple modification of the algorithm, the routes can be adaptive to dynamic traffic conditions by considering the current speed of vehicles occupying the road links. This can be done by reducing the travel cost of edges, where vehicles travel at a high speed. As a result, the suggested routes are likely to bypass congested areas.

3.2.2. Vehicle Generation. A simulation can contain a large number of vehicles. A simulated vehicle is of a specific type and has a number of properties, such as position and speed. SMARTS

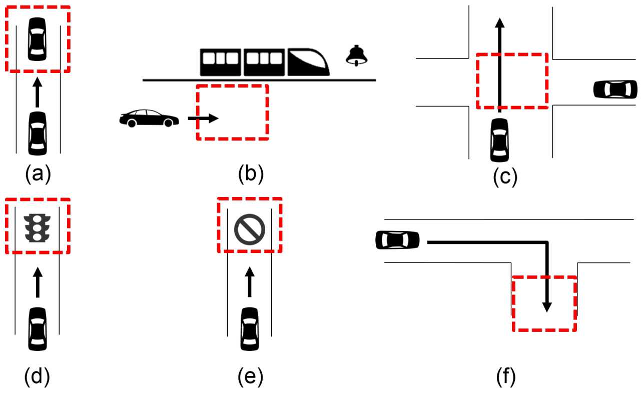


Fig. 6. Positions of impeding objects based on certain factors: (a) a front vehicle; (b) a stopped tram at a tram stop; (c) an uncontrolled intersection with conflicting traffic; (d) a traffic light; (e) a closed lane; (f) a turning point. The positions are marked with dashed rectangles. Some icons in this figure are from Metro Studio [Syncfusion 2015].

supports multiple vehicle types. A vehicle follows a route plan, which can be imported from external source (Section 3.1.2) or be generated by the simulator internally (Section 3.2.1).

When the current time reaches the planned start time of a vehicle, SMARTS will try to insert the vehicle at the start point of its route. The insertion point must be safe for the vehicle such that there is no chance that the vehicle will collide with the existing vehicles on the road. If there is no safe insertion point, the simulator will skip inserting the vehicle and try again in the next time step. This way of simulation corresponds to a real situation.

The simulator can maintain a constant number of internally generated vehicles during simulation. For example, when an internally-generated vehicle reaches its destination and is removed from simulation, SMARTS automatically generates a new vehicle.

When a vehicle is generated, it is labelled as *foreground* or *background*. The simulator can output certain information of the foreground vehicles, e.g., travel time. By default all the internally-generated vehicles are background vehicles. When loading vehicle routes from an external file, user can specify whether the vehicles generated with the file are foreground or background. This feature can help to study the behaviour of certain vehicles that travel with random background traffic. For example, researchers can use the feature to collect the travel time of a small number of foreground vehicles, which follow user-defined routes and travel with hundreds of thousands of random background vehicles.

3.2.3. Car-Following. The movement of vehicles is affected by two driver models, a *car-following model* and a *lane-changing model*. The car-following model computes the ideal speed change of a vehicle such that the vehicle can keep a safe distance to an *impeding object* and travel within the speed limit of the road. The car-following model in the current implementation is the *Intelligent Driver Model (IDM)* [Kesting et al. 2010]. It is straight-forward to add other car-following models to the simulator.

SMARTS considers six types of impeding objects when updating vehicle speed with IDM. Each type corresponds to a specific factor such as front vehicles, red lights, etc. Figure 6 shows the positions of impeding objects in certain scenarios. As shown in the figure, an impeding object may or may not be at the same position of an actual vehicle or an intersection. More details about the properties of the impeding objects are presented in Section 3.2.6.

When searching for an impeding object of a specific type, we set a limit of the searching distance, called *look-ahead distance*. Figure 7 shows an example of the look-ahead distance. If there is no impeding object between the head of a vehicle and the end of the vehicle's current link, the simulator checks the next link on the vehicle's route. The search will continue until all the links that start within

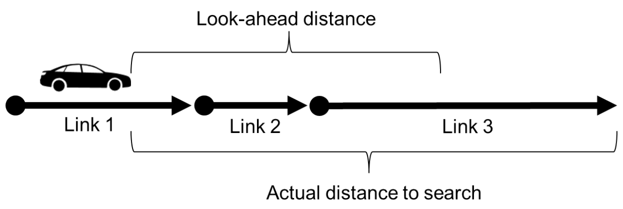


Fig. 7. The look-ahead distance and the actual distance for searching impeding objects.

the look-ahead distance are searched or an impeding object is found. For realistic simulation, the look-ahead distance should be set to a reasonable value, such as 100 meters. If no actual impeding object is found, SMARTS creates a virtual impeding object at a long distance, which will not slow down the vehicle.

Speed changes calculated from different types of impeding objects can be vastly different. For example, if the impeding object is a front vehicle that is moving fast at a long distance, IDM may suggest an acceleration. However, if the impeding object is a red light at a short distance ahead, IDM may suggest a deceleration.

As driver safety is of priority, SMARTS considers all types of impeding objects. For each type, the simulator computes an ideal speed change based on IDM. The speed change that results in the lowest acceleration, i.e., the highest deceleration, is used as the actual speed change. In other words, we always prefer braking over accelerating. Taking the impeding objects in Figure 6 as an example, let us assume that the speed changes corresponding to the impeding objects are: $1m/s^2$, $-1m/s^2$, $-2m/s^2$, $0m/s^2$, $2m/s^2$ and $-0.5m/s^2$. The simulator will set the actual speed change as the lowest acceleration, $-2m/s^2$.

The IDM model is based on Equation 1. For a given vehicle, $\frac{dv}{dt}$ is the speed change suggested by the model, a is the maximum acceleration, v is the vehicle's current speed, v_0 is the desired speed (e.g., the speed limit of a road), δ is the acceleration exponent, Δv is the speed difference between the vehicle and the impeding object, s is the current bumper-to-bumper distance between the vehicle and the impeding object, s_0 is the minimum bumper-to-bumper distance between the vehicle and the impeding object, T is the desired time headway for safety and b is the desired deceleration value.

$$\frac{dv}{dt} = a\left[1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{s^*(v,\Delta v)}{s}\right)^2\right] \tag{1}$$

where

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

3.2.4. Lane-Changing. The simulator uses the lane-changing model, *Minimizing Overall Braking Induced by Lane Changes (MOBIL)*[Kesting et al. 2007], which evaluates whether it is safe and worthy for a vehicle to change lanes. The model first calculates the potential deceleration of the back vehicle in the target lane. If the potential deceleration is within the safe deceleration range of the back vehicle, the lane-change is regarded as safe. Otherwise, the lane-change is not allowed.

If it is safe to make a lane-change, the model checks whether the lane-change would be beneficial. If the following inequation is satisfied, the lane-change is regarded as beneficial. In this inequation, $acc'(M')$ is the new acceleration of the vehicle in the target lane if the vehicle changes to the target lane at the moment. $acc(M)$ is the current acceleration of the vehicle in its current lane. $acc(B')$ is the current acceleration of the back vehicle in the target lane. $acc'(B')$ is the new acceleration of the back vehicle, which is normally lower than $acc(B')$ as the back vehicle in the target lane will be impeded by the vehicle that moves to its front. p is a politeness factor that controls the aggressiveness of drivers in lane-changing. a_{thr} is a threshold for preventing frantic lane-changing. All the acceleration values in MOBIL are calculated based on IDM.

$$acc'(M') - acc(M) > p[acc(B') - acc'(B')] + a_{thr} \tag{2}$$

3.2.5. Traffic Lights. SMARTS can build traffic lights based on OpenStreetMap data. During the set up of simulations, user can choose the timing strategy of traffic lights or disable the traffic lights. Traffic lights for different approaches in the same street are synchronized at an intersection. The simulator gives green lights to different streets at an intersection in succession. The timing of traffic lights can be *static* or *dynamic*. In contrast to static timing, dynamic timing adjusts the time length of green lights based on traffic demand. That is, if there is no traffic in a street with green light but there is traffic in another street, the simulator will give green light to the street with traffic as soon as possible.

3.2.6. Traffic Rules. The simulator supports a range of traffic rules. Similar to the real world, these traffic rules limit the movement of vehicles such that the vehicles must slow down or stop in certain circumstances. Therefore, the simulator generates impeding objects corresponding to these rules.

Our simulator is designed to support complex traffic networks of cities such as Melbourne. We implement the local road rule such that a non-tram vehicle cannot overtake a stopped tram at a tram stop when the road is shared. Figure 6(b) shows the position of the impeding object corresponding to this rule. We should note that the impeding object is placed in the non-tram vehicle's lane, which is parallel to the tram lane. The speed of the impeding object is 0.

Give-way rules for uncontrolled intersections are also implemented. An example is shown in Figure 6(c). A vehicle must give way to other vehicles from certain opposing directions. If a vehicle needs to give way at an intersection, the speed of the corresponding impeding object is set to 0.

A vehicles travelling towards a traffic light must stop when the light is red. If the traffic light is yellow, the vehicle should slow down if it can do a complete stop before reaching the light. Figure 6(d) shows an example. The speed of the corresponding impeding object is set to 0.

A vehicle should move at a relatively low speed when making a turn at an intersection. Figure 6(f) shows an example based on this rule. SMARTS sets the speed of the corresponding impeding object to the recommended turning speed. The impeding object is placed at a short distance from the intersection.

3.2.7. Route-Changing. The simulator can change the route plan of a vehicle under certain circumstances. The first circumstance is when the vehicle moves slowly for a long period of time, e.g., when the vehicle is stuck in a traffic congestion. The second circumstance is when the next link on the vehicle's route is manually blocked during simulation. When the route plan needs to be changed, the simulator computes a new route that starts from the vehicle's current position to the original destination. The new route will bypass the next link on the old route.

3.2.8. Calibration and Prediction. Users can calibrate the car-following model and the lane-changing model by varying the parameters of the models such that the simulated traffic can match real traffic data. This can be done as follows. First, a simulation with the default parameter values is performed. Based on the difference between simulation results and the real traffic data, the user identifies the parameters that can cause the difference. Then, the user changes the values of these parameters and repeat the simulation. This process should be repeated until the simulation result matches the real data. For example, we calibrate the simulation based on real loop detector data[Geroliminis and Daganzo 2008]. During this process, we first perform a simulation with the default parameter values in IDM model and compare the simulation result against the real data. We then choose the parameters that are likely to cause the difference between the two sets of data. The values of the chosen parameters are gradually changed in a number of runs until we find a good match between the simulated data and the real data. Figure 8 shows the calibrated traffic, which shows a good match to the real data as shown in [Geroliminis and Daganzo 2008]. The final parameters of the IDM model are set as follows. The desired acceleration is $1m/s^2$. The desired

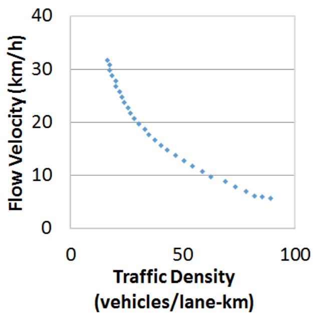


Fig. 8. Density vs. velocity from calibrated simulation.

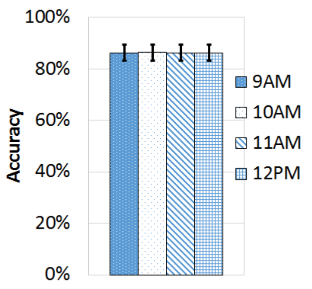


Fig. 9. Accuracy of predicted traffic two minutes in the future.

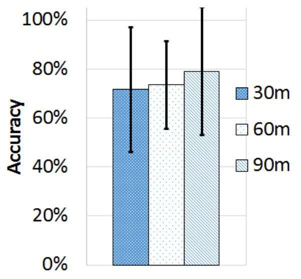


Fig. 10. Accuracy of predicted traffic up to 90 minutes in the future.

deceleration is $2m/s^2$. The minimum bumper-to-bumper distance is 2 metres. The safety headway is 2.5 seconds.

As SMARTS can simulate faster than real time, it can be used to forecast traffic in the future. We conduct tests on traffic prediction based on the real traffic data provided by TomTom [TomTom 2015]. In these tests, we first calibrate traffic by filling up road links with random vehicles. The number of vehicles inserted to a link is determined by the average speed of vehicles on the link, which is extracted from TomTom's data. The number of vehicles is computed with the following formula, in which N is the number of vehicles, l_{edge} is the length of the link, n_{lane} is the number of lanes on the link, $l_{vehicle}$ is the average length of vehicles, v is the average speed of vehicles and T is the safety headway in the car-following model, IDM.

$$N = \frac{l_{edge}}{l_{vehicle} + v \times T} \times n_{lane}$$

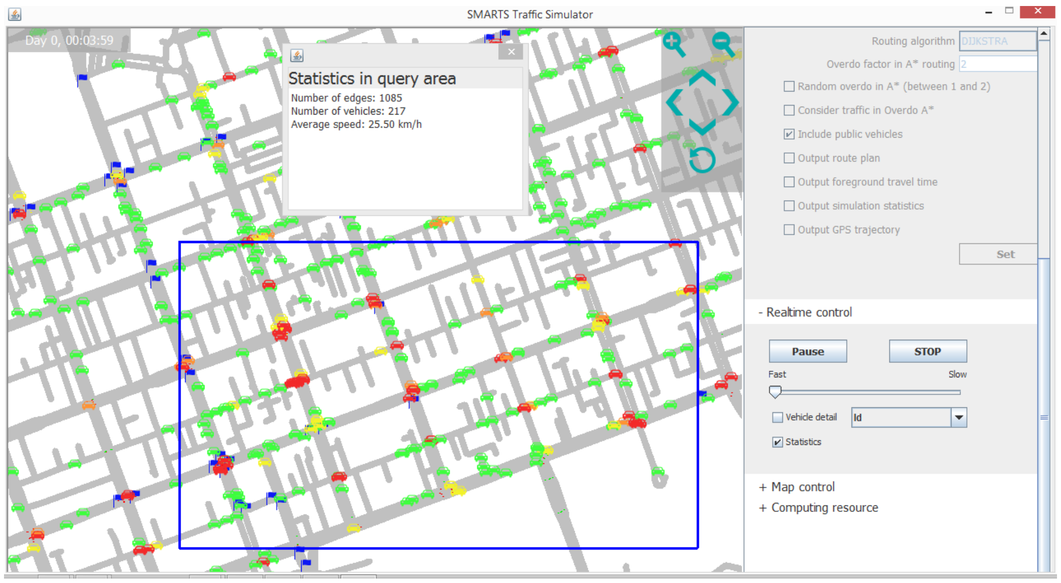
Once simulation is started, we maintain the rate that new traffic enters the simulation area from the border of the area. Due to the highly dynamic nature of traffic, we re-compute the rate each time that the traffic is calibrated with the real data.

To validate whether the predicted traffic matches the real traffic, we collect the prediction accuracy for each simulated road link and average the accuracy from all the links. The prediction accuracy is computed based on the following definition, in which v_s is the average speed of vehicles on a specific link in simulation and v_r is the real average speed of vehicles on the link, which is extracted from the real data.

$$accuracy = \begin{cases} 1 - \frac{|v_s - v_r|}{v_r}, & \text{if } |v_s - v_r| < v_r. \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

We measure the accuracy of traffic prediction while simulating traffic on Eastern Freeway, which is one of the major freeways in Melbourne. The length of the roads in the simulated section is 10.2 kilometres. In the first test, we predict the traffic two-minutes in the future. We perform this at four time points and compute the accuracy. As Figure 9 shows, the accuracy of the predicated traffic maintains at 86%.

In another test, we predict traffic for up to 90 minutes in the future. We simulate traffic on a tollway in Melbourne. The length of the simulated road is 9.2 kilometres. The traffic is initialized based on the live traffic data collected just before the simulation. The simulation is conducted for 90 minutes in real time during the morning peak hours. Figure 10 shows that the accuracy of the predicated traffic is between 71.6% and 79.1%. The simulator achieves a lower accuracy with a higher deviation, compared with the previous test. This is understandable as the traffic can be affected by a larger number of random factors in the real world during a longer period of time.



Statistics in query area
Number of edges: 1085
Number of vehicles: 217
Average speed: 25.50 km/h

Routing algorithm DIJKSTRA
Overdo factor in A* routing 2
☐ Random overdo in A* (between 1 and 2)
☐ Consider traffic in Overdo A*
☑ Include public vehicles
☐ Output route plan
☐ Output foreground travel time
☐ Output simulation statistics
☐ Output GPS trajectory
☐ Vehicle detail Id
☑ Statistics

Fig. 11. Simulating traffic in Melbourne CBD. The left panel shows the simulated traffic. The right panel provides a range of options for controlling simulations. Vehicles are shown as icons at the current zoom level. Colour of vehicles indicates their speed. Traffic statistics collected from a user-defined rectangle is shown in a popup window.

In the final test, we perform a simulation based on the real data collected 24 hours before the simulation. Both days related to the test are weekdays that show similar traffic patterns. At the end of the simulation, we compare the simulated traffic with the real traffic collected on the day of the simulation. SMARTS achieves an accuracy of 73.3%.

3.3. Output

3.3.1. Display. The simulator can run with or without a *Graphical User Interface (GUI)*. The GUI helps users set up, monitor and control simulations at runtime. Figure 11 shows a screenshot of the GUI when running a simulation of Melbourne's road network. Users can perform the following tasks with the GUI:

— Load external vehicle routes.
— Configure simulation parameters, such as the parameters of driver models.
— Define areas of origin and destination for internally-generated vehicles.
— Control simulation speed during simulation.
— Block/unblock road lanes.
— Display details of road links and vehicles.
— Highlight a specific vehicle.
— Add/remove traffic lights at intersections.
— Show traffic statistics collected from a user-defined query window.
— Retrieve background map image from external map providers.
— Import map data from external files.
— Download map data.

3.3.2. Data Export. The simulator can save certain types of data to external files. One of them is the route plan of vehicles. The other is the time-stamped GPS trajectory of vehicles. The travel time of individual foreground vehicles can also be exported. In addition, SMARTS can save certain simulation statistics, such as the computation time and the communication time, to an external file.

4. WORKLOAD DISTRIBUTION

SMARTS is developed as a distributed system that requires one master process, referred to as *server*, and one or more slave processes, referred to as *workers*. The server is responsible for simulation management and visualization. The workers are responsible for the computation of driver models and the update of traffic lights. The server and the workers can run on the same computer or on network-connected computers.

When multiple workers are used, the simulation workload is distributed among the workers. As the workers run in parallel, the simulation time can be significantly reduced from using single worker. The workload is balanced such that each worker is responsible for simulating a similar number of vehicles. We detail the workload balancing strategy later.

Distributing the workload can help to improve the simulation speed but one must also consider the communication cost of distributed simulations. In order to maintain the correctness of simulation, a process may need to periodically send certain types of data to other processes. For example, when a vehicle leaves the respective area of a process and enters the respective area of another process, the details of the vehicle, such as its route plan, need to be transferred to the new process. As there is inevitable overhead in communicating the handover, inappropriate system design can result in extremely high communication cost, negating the benefit of the workload distribution. This can happen if the respective space partitioning is not well chosen, which leads to a considerable amount of time spent on communication.

Communication cost in distributed traffic simulations can be determined by two types of factors [Nagel and Rickert 2001]. The first type is the latency that is proportional to the number of messages and is independent to the size of messages. For example, when passing a message over a TCP connection, there has to be a certain amount of time spent on initiating the communication. The second type depends on the message size and the available bandwidth of the communication channel. The impact of this type of factor can be prominent in low-bandwidth communication environments. The research cloud that we use to perform distributed simulations has high-bandwidth connections between the computing nodes. The bandwidth of the connections is sufficient to handle messages in large size without causing significant delays. Therefore, we focus on the communication cost that is determined by the number of messages.

The number of messages that a process needs to handle is proportional to the number of the other processes, which need to exchange information with the process. We propose a spatial workload balancing strategy that effectively reduces the communication cost by minimizing the number of processes that a process needs to communicate with.

For exchanging messages between distributed processes, we develop a light-weight communication framework based on TCP sockets. We chose TCP sockets rather than high-level APIs, e.g., Java RMI, due to the fact that communication with TCP sockets can be more efficient [Potuzak and Herout 2007].

4.1. Spatial Workload Distribution

SMARTS distributes simulation workload between different workers based on spatial partitioning of the simulation area. When there are multiple workers, the whole simulation area is divided into multiple non-overlapping sub-areas. Each worker will be responsible for simulating the traffic in a specific sub-area.

The simulator uses a virtual grid to facilitate the distribution of workload. The simulation area of a worker consists of one or more grid cells. At the start of simulation, the server organizes the grid cells into one or more groups, each of which is assigned to a worker. Different groups are assigned to different workers. The server informs the assignment of the grid cells to all the workers.

The granularity of the virtual grid can have an impact on the performance of distributed simulations. As detailed in Section 4.3, the workload is balanced between different workers based on the estimated number of vehicles in the grid cells. If the granularity of the grid is significantly low, i.e., the size of individual grid cells is significantly large, the distribution of vehicles may not be even

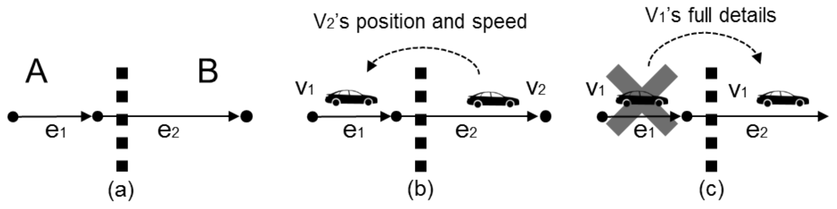


Fig. 12. The exchange of traffic information between two workers. (a) shows two workers, *A* and *B*. The cross-border edge is e_2. (b) shows two vehicles, V_1 and V_2, which are travelling on e_1 and e_2 respectively. Worker *B* sends the status of V_2 to *A*. In (c), worker *A* sends details of V_1 to worker *B* and removes V_1 from its own memory.

across different workers due to the significant difference in the number of vehicles between adjacent cells. For example, let us assume that a large rectangular area is divided into only two cells. One of them contains a dense road network. Another does not contain any road. If the two cells are assigned to two different workers, one worker needs to simulate all the vehicles while another remains idle. This can result in a low performance of the distributed simulations. To prevent this problem, the simulator adjusts the granularity of the grid until the following two conditions are met. First, the number of columns and the number of rows are equal to or higher than the number of workers. Second, the size of any grid cell must be smaller than a user-defined threshold, e.g., $500m \times 500m$.

4.2. Exchange of Traffic Information

In order to maintain the correctness of simulation across different workers, the workers need to exchange certain information of the vehicles that are travelling on *cross-border edges* or entering the edges. A cross-border edge starts from the respective area of a worker but ends in the respective area of another worker. There are two types of information that needs to be exchanged between the workers.

The first type is the status of the last vehicles on cross-border edges. A vehicle is the last one on an edge if it is the furthest vehicle to the end point of the edge. Vehicles on cross-border edges are simulated by the workers, whose respective area covers the end point of the edges. An example is shown in Figure 12. In the second sub-figure, we can see that there are two vehicles, V_1 and V_2. V_1 is simulated by worker *A*. V_2 is simulated by worker *B*. In order to compute the acceleration of V_1, worker *A* needs to know the position and speed of the last vehicle on the cross-border edge e_2, which is V_2 in this case. Therefore, worker *B* needs to send the status of V_2 to worker *A*.

The second type is the full details of the vehicles that are entering cross-border edges. When a vehicle enters a cross-border edge, it leaves the current worker and needs to be transferred to another worker, whose respective area covers the end point of the edge. The vehicle's full details, such as ID, type and route, need to be transferred to the new worker. The third sub-figure in Figure 12 shows an example. Worker *A* detects that V_1 enters the cross-border edge, e_2, at this time step. It transfers the full details of V_1 to worker *B* and removes the vehicle from its own memory. Upon receiving the information, worker *B* creates V_1 at its side.

4.3. Spatial Workload Balancing

When multiple workers run simulation in parallel, it is important that all the workers can finish the same step at approximately the same time. This is because the whole system progresses to the next step only when all the workers finish their current step. If certain workers run significantly longer than the others, the whole system will be slowed down. Therefore, SMARTS needs to balance the workload between the workers. As the length of simulation time is approximately linear to the number of vehicles, the workload is balanced such that different workers need to handle approximately the same number of vehicles.

An important factor that needs to be considered for workload balancing is communication costs. As shown in Section 4.2, a worker must exchange traffic information with other workers, whose respective areas are connected with the respective area of the worker by one or more cross-border edges. When two workers exchange information, each of them sends a message and receives a message. Based on our experience, the number of messages that a worker needs to handle is approximately proportional to the communication time. Therefore, a natural way to reduce communication costs is to reduce the number of workers that a worker needs to communicate with.

Our workload balancing strategy works as follows. The ideal number of vehicles that a worker needs to simulate is computed as the total number of vehicles divided by the number of workers. For each grid cell, we estimate the number of vehicles that will be created in the cell. This number can be known based on the configuration of simulations. For example, if the simulator needs to run a large number of vehicles that are created uniformly at random over the road network, the number of vehicles in a cell will be proportional to the total road length in the cell. In this case, we can estimate the number of vehicles in a cell based on the ratio between the total road length of the cell and the total road length of the whole simulation area. Starting from the first cell on the first row, we aggregate the number of vehicles that will be created in the cells on the same row. When the aggregated number of vehicles reaches the ideal number of vehicles per worker, we assign the cells that participate in the aggregation to one worker. This process repeats until all the grid cells are assigned to the workers. By doing this, it is highly likely that the simulation area of a worker is adjacent to the simulation area of only one or two other workers, unless the grid cells are too small such that certain edges crosses more than two responsible areas.

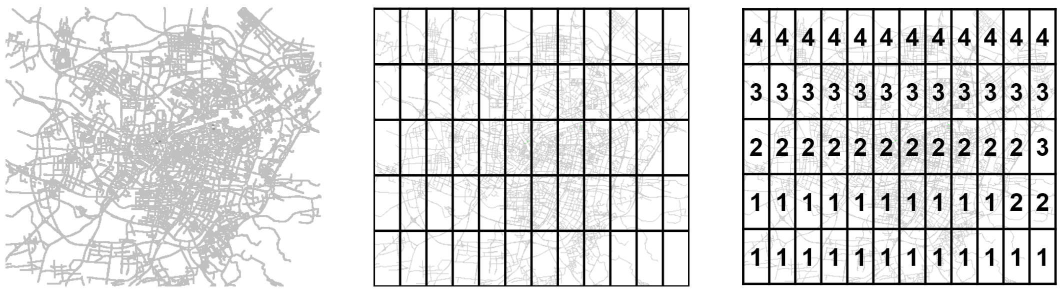


Fig. 13. An example of load balancing. The first sub-figure shows a road network. The second sub-figure shows a grid of 5 rows and 12 columns covering the road network. The last sub-figure shows the work areas of 4 workers after load balancing. Each number in the sub-figure is the ID of the worker that simulates traffic in the corresponding cell.

Figure 13 shows an example of workload balancing. As shown in the figure, the respective area of each worker is adjacent to the respective area of one or two other workers. Therefore, each worker only needs to communicate with one or two other workers during the simulation. The figure also shows that the size of the respective areas can be different between the workers. Similar to this example, our experiments with 30 processors show that the maximum number of workers that a worker needs to communicate with is 2 for simulating the traffic in an area of $600km^2$.

4.4. Simulation Synchronization

Simulation needs to be synchronized between different workers such that a worker can only proceed to the next time step when all its neighbour workers, who need to exchange traffic information with it, complete the current time step. Many traditional distributed systems require a master process to synchronize the distributed work. In contrast, SMARTS is capable of two types of synchronization. One of them is centralized that involves the master process. The other is decentralized that does not involve the master process. The decentralized approach effectively eliminates the risk of a communication bottleneck for the master process.

The centralized synchronization works as follows. At each time step, the server asks all the workers to exchange information with their neighbours. A worker notifies the server after it sends information to all its neighbours. A worker also notifies the server after it receives information from all its neighbours. After all the workers complete the exchange of traffic information, the server asks them to simulate one time step, i.e., updating the status of individual vehicles and traffic lights. A worker notifies the server when it completes the step. When all the workers complete the step, the server proceeds to the next time step by asking the workers to exchange traffic information again. The process will repeat until the simulation is ended.

The decentralized synchronization can save time on communication, especially when there are a large number of workers. Different to the centralized synchronization, once simulation is started, the workers do not communicate with the server until the simulation is finished. A worker automatically starts to simulate one step when it completes the exchange of traffic information with its neighbours. When the worker completes simulating the step, it automatically starts to exchange information with its neighbours for the next time step. The process will repeat until the worker completes all the steps. Then the worker will notify the server.

4.5. Time Costs

Simulation time consists of two parts. One is computation time, e.g., the time for updating vehicles based on the driver models. Another is communication time, e.g., the time for exchanging traffic information between workers. Assuming the workers synchronize simulation in a decentralized fashion, the overall simulation time measured depends on the worker with the longest computation time and the worker with the longest communication time. This is shown in Equation 4, where $t'(i)$ and $t''(i)$ are computation time and communication time of worker i, which is one of the n workers.

$$Time = \max_{1 \leq i \leq n} t'(i) + \max_{1 \leq i \leq n} t''(i) \tag{4}$$

$$Time = \frac{V}{n} t + C \tag{5}$$

We can predict the time cost of running a simulation when two conditions are satisfied. First, the workload is perfectly balanced across different workers. Second, all workers run on different computing nodes, which have the same computing power. The computation time in Equation 4, $\max_{1 \leq i \leq n} t'(i)$, can be seen as $\frac{V}{n}t$, where V is the total number of vehicles, n is the number of workers and t is the computation time for one vehicle. The communication time in Equation 4, $\max_{1 \leq i \leq n} t''(i)$, can be seen as a constant, C, as it is largely dependent on the number of messages, which is a predictable value. Equation 5 shows the model for predicting the cost. Our experiments show that the model works well (Section 5.1.2).

5. EXPERIMENTS

The experiments are conducted on a research cloud with a number of network-connected computing nodes. Any process (server or worker) runs on a node that is different to other processes. The node for a server process uses two virtual CPUs and 8 Gigabytes of memory. The node for a worker process uses one virtual CPU and 4 Gigabytes of memory. The virtual CPUs run on physical CPUs that are clocked at 2.6GHz. Each node hosts a virtual machine that runs Ubuntu 14.04. We perform a number of simulations based on the road network of Melbourne, Australia. The size of the simulation area is $26km \times 23km$. The road network graph contains approximately 150 thousand nodes and 275 thousand directional edges. Each edge has two traffic lanes. There are 2700 nodes with traffic lights.

Based on the traffic statistics of the capital cities in Australia [Victorian Government 2010; Transport for New South Wales 2014], we estimate that the maximum vehicle traffic load in Melbourne is $300,000$. We set the default number of vehicles based on this value. The number of vehicles is constant throughout a simulation. We set the default step length, i.e., the time gap between two steps, to a relatively low value, 0.2 second, which can result in a smooth evolution of traffic.

Table II shows the settings of the experiments. For each setting, we run 5 simulations, each of which lasts 20 minutes of real time. We measure the performance of the simulator using the *real-*

Table II. Experimental Settings

No. of Workers	No. of Vehicles	Step Length	Look-Ahead Distance	Synchronization
$10-30$	$300k$	$0.2s$	$100m$	decentralized
30	$100k-500k, 1M$	$0.2s$	$100m$	decentralized
30	$300k$	$0.2s-1s$	$100m$	decentralized
30	$300k$	$0.2s$	$20m-100m$	decentralized
30	$300k$	$0.2s$	$100m$	decentralized/centralized

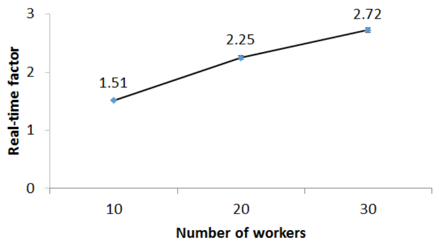


Fig. 14. Real-time factor with the number of workers. Each simulation includes 300,000 vehicles with the look-ahead distance set to 100 metres. The step length is 0.2 second. Simulations run with decentralized synchronization.

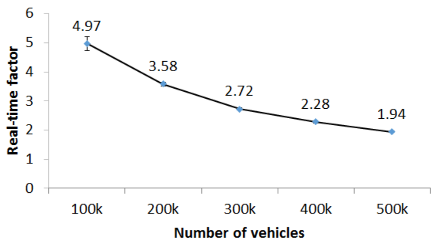


Fig. 15. Real-time factor with the number of vehicles. There are 30 workers. The look-ahead distance is set to 100 metres. The step length is 0.2 second. Simulations run with decentralized synchronization.

time factor, which is defined as the length of real time divided by the length of simulation time. For example, if it takes the simulator 10 minutes to simulate 20 minutes real-time traffic, the real-time factor is 2. The real-time factors from all the 5 simulations are averaged in the final results. The deviation of the real-time factors is also computed.

5.1. Results

5.1.1. Number of Workers. As different workers run on different nodes in parallel, the number of workers has a significant impact on the performance of the system. Figure 14 shows that a higher number of workers helps to speed up the simulations. The performance improvement is almost linear with the increase of the number of workers.

5.1.2. Number of Vehicles. We change the number of vehicles in increment of 100,000. As expected, results show that simulations slow down with more vehicles (Figure 15). When simulating 500,000 vehicles, the simulator can still achieve a real-time factor of 1.94.

We also test the simulator for an extreme scenario, where one million vehicles travel in the same area at the same time. The average real-time factor from the 5 runs is 1.14 with a deviation of 0.01.

The results show that the model for predicting simulation time, which is detailed in Section 4.5, works well. Let us assume that t is the computation time for one vehicle and C is the total communication time of the respective worker. Based on the results for 300,000 vehicles and 400,000 vehicles, we get the following equations.

$$\frac{300000}{30}t + C = \frac{1}{2.72} \times 1200$$

$$\frac{400000}{30}t + C = \frac{1}{2.28} \times 1200$$

Based on these two equations, t is 0.0255 second and C is 185.76 seconds. Using the results for 500,000 vehicles and one million vehicles, we get the following equations.

$$\frac{500000}{30}t + C = \frac{1}{1.94} \times 1200$$

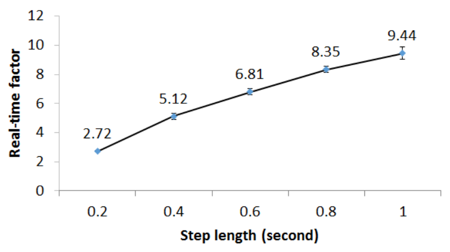


Fig. 16. Real-time factor with step length. There are 30 workers. Each simulation includes 300,000 vehicles with the look-ahead distance set to 100 metres. Simulations run with decentralized synchronization.

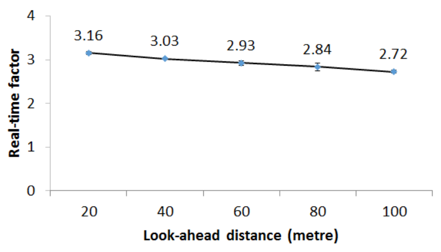


Fig. 17. Real-time factor with look-ahead distance. There are 30 workers. Each simulation includes 300,000 vehicles with the step length set to 0.2 second. Simulations run with decentralized synchronization.

$$\frac{1000000}{30}t + C = \frac{1}{1.14} \times 1200$$

Based on these equations, t is 0.026 second and C is 184.48 seconds, which match the results from the previous set. This means that the performance of the system depends linearly on the number of vehicles.

5.1.3. Step Length. Step length has a significant impact on the system performance. When step length is increased, the number of steps for completing simulation decreases. For example, assuming we want to simulate 20-minute traffic, the simulation needs to run 6000 steps at 0.2-second step length. If the step length is changed to 1 second, the simulation only needs to run 1200 steps. Figure 16 shows the boost of simulation speed with the rise of step length.

5.1.4. Look-Ahead Distance. When look-ahead distance increases, a simulated vehicle driver needs to search impeding objects in a higher number of edges, resulting in a higher computation workload. Therefore, the simulation slows down with the increase of the look-ahead distance. Figure 17 shows this trend.

5.1.5. Synchronization. When the server process is excluded from synchronization, i.e., the synchronization is decentralized, the system eliminates the possible communication bottleneck at server and achieves an average real-time factor of 2.72. When centralized synchronization is used, the average real-time factor drops to 2.47.

6. CONCLUSION

SMARTS is capable of performing highly scalable microscopic traffic simulations based on a distributed computing architecture. The proposed spatial workload balancing strategy and the decentralized synchronization strategy help to reduce simulation time when running simulations with multiple computing nodes, regardless of the size of road network or the number of computing nodes.

SMARTS provides a versatile platform for traffic simulation. The simulator has a comprehensive set of features that help users customize, visualize and automate the execution of simulations. It can also calibrate simulation with real traffic data. The system is developed with a number of relatively independent software modules, which provides the flexibility for future extensions.

The current implementation of SMARTS can perform faster-than-real-time simulations with up to one million vehicles. With further optimization of the system, we expect that the simulator can achieve even higher speed in large scale simulations. The simulator uses static workload balancing, which means that the partition of simulation area is constant throughout a simulation. A possible research direction is to study the effects of dynamic workload balancing on the performance of simulations. The current implementation assumes that all the vehicles behave rationally, e.g., all the vehicles obey the traffic rules. A possible direction of extension is to introduce certain random irrational driver behaviours to simulations.

ACKNOWLEDGMENTS

This research was supported by a Discovery Project grant DP130103705 and a Linkage Project grant LP120200130.

REFERENCES

J. Barceló, E. Codina, J. Casas, J. L. Ferrer, and D. García. 2005. Microscopic traffic simulation: a tool for the design, analysis and evaluation of intelligent transport systems. *Journal of Intelligent and Robotic Systems* 41, 2-3 (2005), 173–203.

T. Brinkhoff. 2002. A framework for generating network-based moving objects. *Geoinformatica* 6, 2 (2002), 153–180.

E. W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.

C. Düntgen, T. Behr, and R. H. Güting. 2009. BerlinMOD: a benchmark for moving object databases. *The VLDB Journal* 18, 6 (2009), 1335–1368.

P. A. M. Ehlert and L. J. M. Rothkrantz. 2001. Microscopic traffic simulation with reactive driving agents. In *Intelligent Transportation Systems*. IEEE, 860–865.

N. Geroliminis and C. F. Daganzo. 2008. Existence of urban-scale macroscopic fundamental diagrams: some experimental findings. *Transportation Research Part B: Methodological* 42, 9 (2008), 759–770.

P. Hidas. 2002. Modelling lane changing and merging in microscopic traffic simulation. *Transportation Research Part C: Emerging Technologies* 10, 5 (2002), 351–371.

R. Jacob, M. Marathe, and K. Nagel. 1999. A computational study of routing algorithms for realistic transportation networks. *Journal of Experimental Algorithmics* 4 (1999), 6.

A. Kesting, M. Treiber, and D. Helbing. 2007. General lane-changing model MOBIL for car-following models. *Journal of the Transportation Research Board* 1999, 1 (2007), 86–94.

A. Kesting, M. Treiber, and D. Helbing. 2010. Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 368, 1928 (2010), 4585–4605.

R. Klefstad, Y. Zhang, M. Lai, R. Jayakrishnan, and R. Lavanya. 2005. A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation. In *Intelligent Transportation Systems*. IEEE, 813–818.

D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner. 2002. SUMO (Simulation of Urban MObility) – an open–source traffic simulation. In *Middle East Symposium on Simulation and Modelling*. DLR, 183–187.

D. H. Lee and P. Chandrasekar. 2002. A framework for parallel traffic simulation using multiple instancing of a simulation program. *Intelligent Transportation Systems* 7, 3-4 (2002), 279–294.

G. McArdle, E. Furey, A. Lawlor, and A. Pozdnoukhov. 2014. Using digital footprints for a city-scale traffic simulation. *ACM TIST* 5, 3 (2014), 41.

M. F. Mokbel, L. Alarabi, J. Bao, A. Eldawy, A. Magdy, M. Sarwat, E. Waytas, and S. Yackel. 2013. MNTG: an extensible web-based traffic generator. In *Advances in Spatial and Temporal Databases*. Lecture Notes in Computer Science, Vol. 8098. Springer, 38–55.

K. Nagel and M. Rickert. 2001. Parallel implementation of the TRANSIMS micro-simulation. *Parallel Comput.* 27, 12 (2001), 1611–1639.

U. T. V. Nguyen, S. Karunasekera, L. Kulik, E. Tanin, R. Zhang, H. Zhang, H. Xie, and K. Ramamohanarao. 2015. Randomized path routing algorithm for decentralized route allocation in transportation networks. *IWCTS* (2015), 6.

T. Potuzak and P. Herout. 2007. Use of distributed traffic simulation in the JUTS project. In *The International Conference on Computer as a Tool*. IEEE, 2250–2255.

PTV Group. 2015. PTV Vissim. http://vision-traffic.ptvgroup.com/en-uk/products/ptv-vissim. (2015).

SMARTS Team. 2015. Video demo of SMARTS. https://youtu.be/sUMfHXOBCE8. (2015).

Syncfusion. 2015. Syncfusion Metro Studio. https://www.syncfusion.com/downloads/metrostudio. (2015).

S. Taori and A. Rathi. 1996. Comparison of NETSIM, NETFLO I, and NETFLO II traffic simulation models for fixed-time signal control. *Transportation Research Record: Journal of the Transportation Research Board* 1566 (1996), 20–30.

TomTom. 2015. Online Navigation Services. http://developer.tomtom.com. (2015).

Transport for New South Wales. 2014. Household travel survey report: Sydney 2012-13. http://www.bts.nsw.gov.au/ArticleDocuments/79/r2014-11-hts-summary-report_12-13.pdf.aspx. (2014).

Victorian Government. 2010. Melbourne Metro One Business Case Draft: public transport patronage forecasts. http://ptv.vic.gov.au. (2010).

X. Wang, C. Leckie, H. Xie, and T. Vaithianathan. 2015. Discovering the impact of urban traffic interventions using contrast mining on vehicle trajectory data. In *Pacific Asia Conference on Knowledge Discovery and Data Mining*, Vol. 1. Springer, 486–497.

R. A. Waraich, D. Charypar, M. Balmer, and K. W. Axhausen. 2015. Performance improvements for large-scale traffic simulation in MATSim. In *Computational Approaches for Urban Environments*. Springer, 211–233.

Q. Yang and H. N. Koutsopoulos. 1996. A microscopic traffic simulator for evaluation of dynamic traffic management systems. *Transportation Research Part C: Emerging Technologies* 4, 3 (1996), 113–129.

Y. Zheng. 2015. Trajectory data mining: an overview. *ACM TIST* 6, 3 (2015), 29.

Y. Zheng, L. Capra, O. Wolfson, and H. Yang. 2014. Urban computing: concepts, methodologies, and applications. *ACM TIST* 5, 3 (2014), 38.