QarSUMO: A Parallel, Congestion-optimized Traffic Simulator

Hao Chen*
chenhao@hbku.edu.qa
University of Science and Technology
of China & Qatar Computing
Research Institute, HBKU

Ke Yang*
yke@hbku.edu.qa
Tsinghua University & Qatar
Computing Research Institute, HBKU

Stefano Giovanni Rizzo
strizzo@hbku.edu.qa
Qatar Computing Research Institute,
HBKU

Giovanna Vantini
gvantini@hbku.edu.qa
Qatar Computing Research Institute,
HBKU

Phillip Taylor
phillipt@princeton.edu
Princeton University

Xiaosong Ma
xma@hbku.edu.qa
Qatar Computing Research Institute,
HBKU

Sanjay Chawla
schawla@hbku.edu.qa
Qatar Computing Research Institute,
HBKU

ABSTRACT

Traffic simulators are important tools for tasks such as urban planning and transportation management. Microscopic simulators allow per-vehicle movement simulation, but require longer simulation time. The simulation overhead is exacerbated when there is traffic congestion and most vehicles move slowly. This in particular hurts the productivity of emerging urban computing studies based on reinforcement learning, where traffic simulations are heavily and repeatedly used for designing policies to optimize traffic related tasks.

In this paper, we develop QarSUMO, a parallel, congestion-optimized version of the popular SUMO open-source traffic simulator. QarSUMO performs *high-level parallelization* on top of SUMO, to utilize powerful multi-core servers and enables future extension to multi-node parallel simulation if necessary. The proposed design, while partly sacrificing speedup, makes QarSUMO compatible with future SUMO improvements. We further contribute such an improvement by modifying the SUMO simulation engine for congestion scenarios where the update computation of *consecutive and slow-moving vehicles* can be simplified. We evaluate QarSUMO with both real-world and synthetic road network and traffic data, and examine its execution time as well as simulation accuracy relative to the original, sequential SUMO.

CCS CONCEPTS

• **Computer systems organization** → **Parallel architectures**;
• **Information systems** → **Spatial-temporal systems**.

---

*Both authors contributed equally to this research.

KEYWORDS

transportation simulation, distributed and parallel computing

1 INTRODUCTION

The design of road infrastructures and the planning of traffic control are challenging tasks, often requiring complex modeling to verify and analyze feasible solutions safely and efficiently. Road traffic simulators are specialized software widely used in research and practice and make it possible to recreate real-world scenarios, such as the congestion surrounding a stadium after a sport event, or the traffic flow in a major city during rush hours. Through simulators it is possible to experiment with different "what-if" scenarios, predicting their outcome with a certain degree of accuracy, as well as validating different solutions to mitigate unwanted effects.

There are three main types of traffic simulators: *macroscopic*, *mesoscopic*, and *microscopic* [21]. Macroscopic simulators focus on generating higher-level aggregate traffic statistics, while mesoscopic ones work at the level of groups of vehicles. The focus of our work is on microscopic simulators, where individual vehicles in a network are modeled. The demand for such detailed simulation is increasing, as they enable the evaluation of fine-grained traffic control policies, such as the impact of a new traffic signal system on rush hour traffic with each individual vehicle adopting dynamic routing. However, the main challenge with microscopic simulation is that such detailed computation is time-consuming - as the scale of the simulation grows, the per-timestep simulation overhead increases accordingly, possibly preventing quick and timely responses as requested under real time operating conditions. The modeling of traffic congestion further dramatically increases the simulation time. As common microscopic simulation performs discreet object

simulation by moving each vehicle at preset physical (wall) timestep intervals, such as 0.5 second, in congested scenarios the number of vehicles and interactions between them grows, resulting in poor performance. Unfortunately, traffic congestion scenarios are important study subjects and natural optimization targets, making their costly simulation necessary for users.

The above challenges are highlighted in new use scenarios of traffic simulation. In particular, traffic simulators and their performance have played a central role in the recent trend of applying Reinforcement Learning (RL) techniques in urban computing [12, 15, 17, 20, 26]. All these approaches rely heavily on the repeated simulation of thousands of *episodes*, in which an agent, such as a traffic light planner, explore a huge number of combinations of actions in order to learn an optimal policy. It follows that in order to enable such techniques for traffic control, the simulations have to run at a speed that is orders of magnitude faster then the real time. This is one of the reasons why RL has been so far applied only on single intersections [20, 27] or on simple grid-like networks [15, 25]. It is clear that the future success of these approaches is strongly tied to the efficiency and scalability of the underlying traffic simulators.

In this paper, we tackle the scalable and efficient traffic simulation challenge by enabling parallel simulation. Rather than building new systems, we choose to improve the popular open-source SUMO simulation software [16], which has already established a sizable user community (with an annual user conference [2]). In exchange for its rich features (such as detailed microscopic simulation, support of different traffic and routing modes, and powerful interfaces), SUMO suffers longer execution times compared to lightweight alternatives [7, 19, 22]. For example simulating one hour of traffic involving a million vehicles traversing a city, it can take SUMO to processing time non responding to requested needs [14].

Our proposed system, QarSUMO, adds high-level parallelism on top of SUMO simulation, by adding network partitioning and inter-partition vehicle state synchronization along the border edges. This approach retains SUMO internal designs, allowing orthogonal optimizations to be easily incorporated. Considering that powerful multi-core servers today provide considerable hardware parallelism, and that typical road network data could easily fit into a single node's main memory, we focus in this work on single-server parallel execution using multiple threads. QarSUMO, however, adopts MPI communication [9] and can be easily extended to multi-node parallel execution in the future.

We then focus on SUMO internal optimizations to speed up traffic simulation under congestion. The intuition here is that vehicles move very slowly, maintaining their close distances, while events like lane change become more rare. Therefore, when we temporarily reduce the simulation granularity and model a consecutive sequence of vehicles within the same lane, it does not significantly affect the simulation outcome. Based on dynamically monitored congestion level, length of road segment, and distance to the next road junction, QarSUMO judiciously applies such *group simulation* to save simulation time while retaining reasonable accuracy relative to the SUMO baseline.

We evaluate QarSUMO with both synthetic grid networks and a real road network of the Doha Corniche area. This area hosts arterial traffic linking key government agencies, major tourist attractions, and many hotels, making it a strategic transportation management target in preparation for the Qatar 2022 FIFA World Cup. Our results show that QarSUMO is able to bring an order of magnitude performance improvement for SUMO simulations, while maintaining reasonably high accuracy. QarSUMO will be released as open-source software.

The rest of the paper is organized as follows. Section 2 gives background information on SUMO. Section 3 and Section 4 describes QarSUMO design on parallel simulation and congestion simulation optimizations, respectively. Section 5 reports evaluation and Section 6 discusses related work. Finally, Section 7 summarizes the paper as well as potential future work.

2 BACKGROUND: SUMO

2.1 Microscopic Traffic Simulation

Most microscopic simulators model the complete traffic ecosystem including the road network with lanes, traffic signals, routing strategy, and the kinematics of vehicles. The movement of vehicles are modeled using a *car following model (CFM)* and a *lane changing model (LCM)* [10, 21]. At the most basic level, a CFM models the acceleration $a_{i,t+1}$ of a vehicle i at time step $t + 1$ as:

$$a_{i,t+1} = f(v_{i,t}, v_d, v_{i-1,t})$$

Here $v_{i,t}$ is the speed of the vehicle i at time t, v_d is the desired speed, and $v_{i-i,t}$ is the speed of the immediate vehicle ahead. When the vehicle in question sees no immediate traffic ahead or faces a traffic signal, the vehicle ahead is replaced by a fictitious vehicle or an impeding object, respectively. The function $f()$ depends upon the specific CFM.

The LCM, on the other hand, is a binary decision-making function that decides whether it is appropriate for a vehicle to change lanes at a given time step:

$$\mathbb{1}_{>0}(i, t + 1) = g(a_{i,t}, a_{i-1,t}, a_{i+1,t})$$

Note that unlike the CFM, LCM depends not only on the vehicle in front ($v_{i-1,t}$), but also the vehicle that is behind in the target lane ($v_{i+1,t}$). Here the function $g()$ depends upon the specific LCM that is being used.

From a distributed computing perspective, the key challenge is to synchronize the computation of both $f()$ and $g()$ across partitions. For example, with CFM, if the current vehicle and the vehicle immediately ahead are in different partitions then provisions have to be made to communicate the information across partitions. This is the key reason that parallelizing a microscopic simulator in a manner that is accurate and efficient at the same time is non-trivial.

There are several variations of both CFM and LCM in the literature [10, 21]. One advantage of using open-source simulators like SUMO (to be described below) is that these alternative models can be implemented and tested by users.

2.2 SUMO Overview

SUMO [16] is an open source, freely available microscopic simulation software. It offers a great number of features and extensions, and is currently used worldwide for traffic research [1].

SUMO can read road networks from multiple popular formats, including OpenStreetMap, OpenDRIVE, Shapefile, Vissim, and MAT-Sim. Traffic demand can be modeled in SUMO as specified routes, individual trips from origin to destination, or high-level flows generating periodic trips. Routes can be generated statically or dynamically and vehicles are distributed among the routes with different strategies, for example, according to predefined statistics on turning ratios or detectors data. SUMO can also simulate multi- and inter-modal trips: each trip can use single- or multi- transportation mode. For example each individual can move by walking, riding a vehicle, or using public transport, and also transferring among different modes, *e.g.*, by car and then by rail.

Among the many advanced features of the vehicle interactions, SUMO supports different strategies for lane-changing, driving impatience parameters for overtaking, multiple pollutants emission models, and the configuration of onboard bluetooth and wifi devices, to simulate wireless sensor detection. Finally, a very fine-grain level of details can be managed and analyzed on each single vehicle and road segment. For electric vehicles, for example, parameters such as battery capacity, drive efficiency, and the minimum velocity to start charging can be specified, along with charging station locations in a road network.

SUMO allows external programmatic interactions through two different interfaces: TraCI (Traffic Control Interface) and Libsumo. Both these interfaces provide the capability to retrieve values of any simulated objects, modify most of the objects settings, and have control on the running simulation while it is running. The TraCI interface[1] is a TCP-based client/server API and the most common way of interacting with SUMO. The main advantage of TraCI is that it abstracts from any specific language or platform, leaving users with full flexibility. On the flip side, TraCI brings high communication overhead, an issue particularly in partitioned simulations [3]. Libsumo on the other hand provides a much more efficient coupling, exposing the same interface methods of TraCI as C++ static functions, though with few limitations[2] and less mature support than with TraCI.

Finally, SUMO provides user-friendly GUI for the user to interact with the simulation and visualize intermediate results, thus facilitating the understanding and analysis of traffic scenarios. Figure 1 gives a sample SUMO screenshot that shows a running simulation on a real road network. The map network can be navigated and zoomed, while specific objects can be selected to monitor related parameters, such as the current route of a vehicle or its velocity.

2.3 Key Entities and Actions in SUMO

We briefly introduce the key objects and important data structures in SUMO, before we move to QarSUMO design.

In order to simulate traffic, SUMO keeps track of real traffic networks as well as moving entities using its internal representations. Major objects concerned in this paper's discussion include:

- *Junction*: a junction (node) is a single point where at least one road segment starts or ends. Junctions may have traffic lights, individually controlled by configurable signal transition policies.
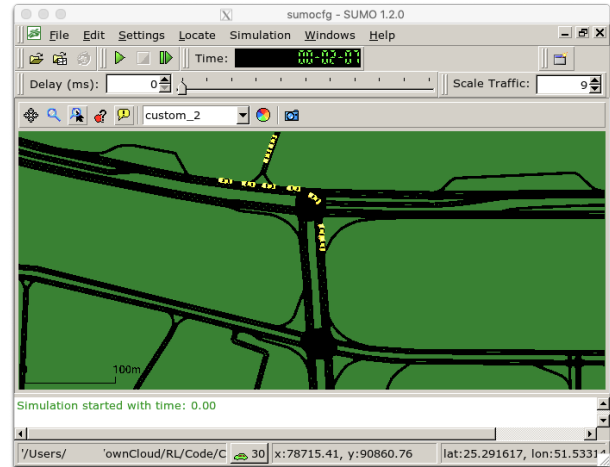
Figure 1: Screenshot of a running simulation in SUMO.

- *Edge*: an edge represents a one-way, uninterrupted road segment between two road junctions, each with its speed limit and potentially containing multiple *lanes*. Together, edges and junctions define a traffic network. Note that at intersections, we also have arch-shaped or straight lanes connecting the corresponding lanes of intersecting edges.
- *Vehicle*: vehicles are the main simulation objects and contain many attributes, such as the current speed, position, and acceleration. They are stored in a per-lane array, for their successive updates.

The simulation timestep is configurable, with SUMO default set at 0.5 second. At each timestep, SUMO examines the vehicles in each lane sequentially, consistent with the natural vehicle following behavior in real world. For each vehicle, SUMO adjusts its speed using the selected CFM, considering factors such as the distance with the vehicle ahead, speed limit, distance to junction, acceleration, etc. A vehicle may initiate other events, such as lane change, as required by routing or enabled as probabilistic actions.

To simulate real-world driving scenarios, SUMO also inserts configurable randomness into the vehicles' behavior. As such non-deterministic executions make it more difficult for QarSUMO to compare with the original SUMO for simulation accuracy, we disable such randomness insertion in our evaluation.

3 META-PARALLELIZATION OF SUMO

3.1 QarSUMO Architecture Overview

QarSUMO enables parallel simulation of SUMO by partitioning the road network (and the corresponding traffic). QarSUMO adopts a fully distributed design, with no central coordinator. Each partition is processed by a QarSUMO process, who performs local updates, and exchanges vehicle states on *cut edges* (the edges spanning two neighboring partitions) at the end of each simulation timestep.

Figure 2 illustrates the QarSUMO architecture, using a simple example of 3-partition execution. QarSUMO adopts the widely used METIS graph partitioning tool [11] to generate desired road network partitions. Each QarSUMO process runs one instance of
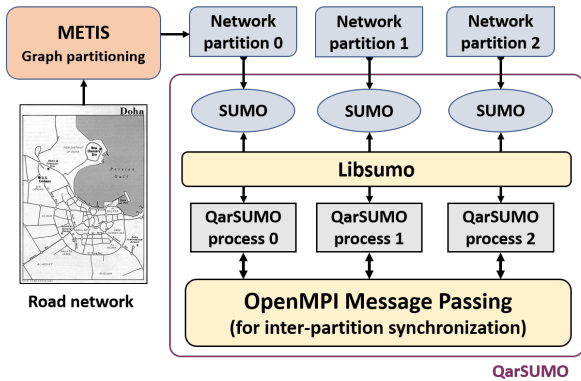
Figure 2: QarSUMO Architecture

sequential SUMO simulation on its own network partition. This leaves all the communication and synchronization to the QarSUMO processes, using OpenMPI [9] for message passing. The SUMO instance invoked within each QarSUMO process, meanwhile, retains their sequential simulation workflow.

QarSUMO processes only need to interact with their corresponding SUMO instance at timestep granularity and the interaction is limited to only the traffic on border edges (along with their associated junctions). This is done via the standard Libsumo C++ interfaces. Details are to be given in Section 3.3.

This design allows QarSUMO's parallel execution to stay high-level and relatively independent of the SUMO sequential simulation implementation. As SUMO itself is been increasingly adopted and professionally maintained, such decoupled design allows QarSUMO to easily accommodate new SUMO optimizations or upgrades. Our congestion optimization technique (to be presented in Section 4, for example, is one such enhancement that does modify SUMO internal implementation).

Also, QarSUMO's design is made considering potential future SUMO's own parallelization. Likely due to growing user demands, recently the SUMO team is adding its own multi-threaded parallel implementation. However, the work does not seem to be complete yet, and our evaluation finds the current SUMO parallel execution often producing longer simulation time than the sequential version, due to heavy inter-thread synchronization. However, with potential future versions achieving reasonable multi-threaded execution efficiency, QarSUMO's MPI-based meta-parallelization could provide additional speedup on top of it. For example, if SUMO is able to scale out to 4 or 8 threads doing parallel simulation using shared memory, on a 32-core server QarSUMO could run 8 or 4 such SUMO instances, each working on a network partition and invoked by a QarSUMO process still communicating using message passing.

Finally, MPI-based message passing design easily extends to multi-node parallel execution. In this paper we focus on single-node evaluation, considering our moderate sub-city scale road networks and users' preference of avoiding cluster setup. Meanwhile, QarSUMO has no problem utilizing multiple server nodes, when needed in processing larger networks or traffic volumes.

3.2 Network Partitioning

For parallel simulation, QarSUMO needs to first partition the road network. It adopts *vertex partitioning*, so that each junction is uniquely assigned to one partition. If both ends of an edge belong to the same partition, then the edge becomes an *internal edge* there. Otherwise, the edge sits across two partitions and becomes a *border edge* (cut edge).

Graph partitioning is a relatively mature field and QarSUMO chooses to leverage the popular METIS tool [11], which supports balanced partitioning with a variety of optimization objectives. QarSUMO first extracts the road graph from the input traffic network, then partitions the graph using METIS, and finally converts the original network according to the partitioning results into corresponding network partitions. Compared to the simulation time (especially with iterative episode simulation for reinforcement learning training), the time spent on such preprocessing is minor. For example, it takes METIS 0.01 second to partition the networks used in our evaluation (details in Section 5.1).

Therefore, for long-running experiments simulating consistently imbalanced traffic (with some of the road segments much busier than others), it is worthwhile to factor in the traffic distribution to periodically re-generate network partitions in a load-balanced manner. To this end, QarSUMO supports optional enhanced network partitioning by taking into account the routing information of vehicles (typically stored in the route file for SUMO simulation).

More specifically, METIS balances graph partitions by total vertex weight. Therefore QarSUMO first traverses the static routing paths of all vehicles to be simulated and count the per-edge accesses. Then for each junction v, we calculate its traffic-aware vertex weight w_v as:

$$w'_v = \sum_{i=1}^{n} C_{e_i} L_{e_i}, \tag{1}$$

$$w_v = \frac{1}{|V|} \sum_{i=1}^{|V|} w'_{v_i} + w'_v \tag{2}$$

Here e_i enumerates v's incident edges, while C_{e_i} and L_{e_i} are the access count and length of edge e_i, respectively. All junctions are assigned $\frac{1}{|V|} \sum_{i=1}^{|V|} w'_{v_i}$ (the average weight over all vertices) as a base weight, so that they can be properly involved in partitioning even with no expected traffic.

With METIS partitioning assigns vertices (junctions in our case) to partitions. For parallel processing, the junctions connected by a border edge also need to be replicated at both partitions. For such a junction, we call its copy at the partition it is assigned to by METIS the *primary junction* and the replicated one its *shadow junctions*.

In addition to generating partitions with balanced total edge weight, METIS allows an additional user-specified optimization goal. QarSUMO elects to minimize edge cut, as the communication overhead is heavily influenced by the number of border edges (more details on communication in Section 3.3 next).

Table 1 demonstrates sampling partitioning results of METIS. Less than 1% of edges are border edges when we have only two partitions. As expected, as we increase the number of partitions,

Network/Partition	2	4	8	16	32
Corniche	0.43%	1.01%	2.67%	4.54%	8.00%
Cologne	0.07%	0.17%	0.33%	0.77%	1.04%
Grid	0.39%	1.13%	2.68%	6.51%	11.02%

Table 1: Ratio of border edges

both networks have growing percentage of edges becoming border edges.

3.3 Inter-Partition Synchronization
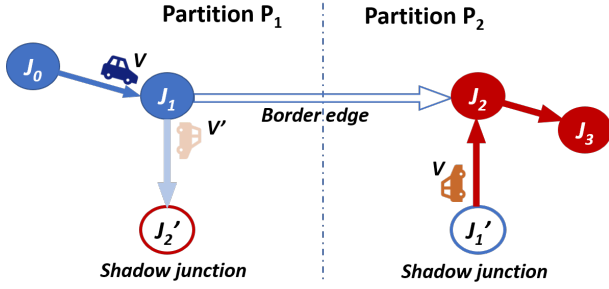
Figure 3: Synchronization between two partitions in QarSUMO

At the end of each simulation timestep, QarSUMO performs vehicle/lane state synchronization along the border edges. This is done by the replication of both road network and vehicle data traversing the border edges.

As mentioned earlier, junctions are replicated across partitions. A border edge, accordingly, is also replicated. Figure 3 illustrates a directed border edge, $J_1 J_2$, sits across two partitions P_1 and P_2, where J_1 and J_2 are assigned, respectively. The shadow junctions, J_1' and J_2', reside at the opposite side.

In QarSUMO's partitioned network, the border edge $J_1 J_2$ does not exist. Instead, we have the replicated border edge $J_1 J_2'$ in P_1, and $J_1' J_2$ in P_2. Considering that the traffic *ahead* of a certain vehicle has far more influence on its movement than the traffic behind it, we consider the destination partition has more information to dictate a vehicle's state computation. Therefore, the edge $J_1' J_2$ (in P_2) is the *primary edge* here, as P_2 hosts all the incident edges of J_2. while $J_1 J_2'$ becomes its *shadow edge*. The border edge of opposite direction, $J_2 J_1$, has symmetric arrangements.

When a vehicle V moves from junction J_0 to $J1$, continuing via $J_1 J_2$ (conceptually) toward $J2$, it emerges first in $P1$ on the shadow edge $J_1 J_2'$. At the end of that timestep, the bi-directional communication between P_1 and P_2 will lead to the insertion of V's replica in P_2, on edge $J_1' J_2$. Since this edge is a primary edge, the vehicle replica on it now becomes the *primary vehicle*, with the original copy in P_1 reduced to be its *shadow vehicle*. From that point, P_2 takes over the main control of V's simulation, and at the end of each timestep, while V is still on $J_1' J_2$, sends updates to P_1.

Note that the QarSUMO process in charge of P_1 cannot simply drop V from the shadow edge, which would affect the update of newly inserted vehicles after it. Instead, it performs its local updates, while adjusting it after getting updates from P_2, allowing

traffic situation to propagate from J_2 to J_1 along this border edge. At the end, when V arrives in J_2 and moves on to an internal junction J_3, V's shadow vehicle is removed from the shadow edge and disappears from P_1. Care is taken to deal with corner cases such as V immediately returns to P_1 via another border edge.

For cross-partition synchronization, at the end of each simulation timestep, a QarSUMO process retrieves vehicle state updates concerning border edges from the underlying SUMO instance, using the aforementioned Libsumo interfaces. It aggregates such per-vehicle data to be sent to the same destination partition into a single MPI message. All QarSUMO processes use the `MPI_Alltoall` collective communication call to simultaneously scatter/gather updates efficiently. Finally, they each applies the appropriate update for vehicle insertion and shadow vehicle state update, again via Libsumo.

4 COMPUTATION OPTIMIZATION UNDER CONGESTION

4.1 Optimization Rationale

As mentioned earlier, traffic simulation is especially time-consuming when simulating vehicles under traffic congestion. As vehicles move much slower compared with under light traffic, it takes much more simulation timesteps for them to move the same distance. Close examination of SUMO implementation reveals that the computation follows the same workflow regardless of congestion situations: each vehicle calculates its speed ahead according to the selected car following model, only to be severely constrained by the vehicle immediately ahead. It nevertheless follows its entire procedure, including spending considerable computation on checking lane change opportunities.

In real life, cars stuck in traffic have little choice but to follow the car ahead in lock step, and drivers have little motivation to change lanes for the sake of moving faster. Therefore, under heavily congested traffic, we are spending a lot of cycles on simulating scenarios with not much happening. The intuition here is that one could probably achieve very similar simulation outcome when reducing the simulation granularity.

We first explored enlarging the temporal granularity by increasing the simulation timestep, *e.g.*, using larger timestep values than the SUMO default of 1 second. However, we found that this approach significantly impacts simulation accuracy when applied globally. To have varied simulation timesteps at different regions according to the traffic congestion level, on the other hand, significantly increases software complexity. In particular, SUMO has other timestep settings, such as "action timestep", which is used to recompute vehicle's speed and lane-changing decisions. The interaction of these related parameters making it difficult to adjust the temporal granularity of simulation. Things become even more challenging when we consider QarSUMO's parallelization, where the synchronization (described in the previous section) is built upon having consistent simulation timestep across all partitions and border edges.

We then investigated varying the spatial granularity. Here we have two choices. One is to replace a sequence of vehicles packed under heavy congestion into a long "super-vehicle", whose movement is replicated to its member vehicles till its decomposition (due to dissolving congestion or approaching intersection). The other

is to create a virtual group of such a vehicle sequence, make the first vehicle in the sequence a *leader*, while skipping most of the simulation computation steps for the *followers* by directly *copying* leader simulation updates.

Our examination of the SUMO simulation processing finds the second approach more lightweight and easier to implement. It also reserves each vehicle's individual examination, making it easier to address their individual situations. Less modification to SUMO data structures are needed (mostly extensions to the lane and vehicle object structures to add additional attributes). We discuss the design of this virtual grouping ("grouping" for short) approach in the rest of this section.
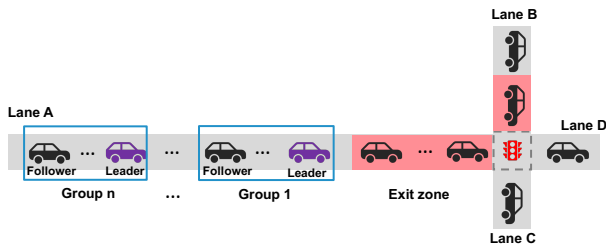

Figure 4: Group design

4.2 Vehicle Grouping

QarSUMO performs its grouping judiciously, only considering grouping vehicles when they appear to be consistently moving slowly, relative to the speed limit of their lanes.

Such a criterion requires the examination of a sequence of vehicles. While a lane seems a natural container where we can assess the average vehicle speed, lanes (especially long ones) often have uneven congestion situations, especially with traffic lights.

QarSUMO takes an ad-hoc approach that takes into consideration both vehicle speed and lane length. For each lane, we first define an *exit zone* at the end of the lane where vehicles, if grouped, will "escape" and return to the original, individual SUMO simulation. The length of this exit zone is set relative to the length of the corresponding lane (currently configured at 10% of the latter), while bounded by a configurable cap (by default 50 meters in QarSUMO). This allows vehicles, under congestion, to disband in time to handle the upcoming junction, including changing lanes if necessary. Note that we do not enforce a lower limit on exit zone length. This is based on the observation that very short edges (road segments) are unlikely to have multiple lanes in the first place, and the group leader always follow the full SUMO individual vehicle simulation process.

Figure 4 illustrates a lane under heavy congestion, with the exit zone marked facing the destination junction. For the remaining lane region, we partition them into k zones, with vehicles within each zone examined collectively. In our current implementation, we empirically set k at 3. The rationale behind is that when there is heavy congestion, we hope to achieve large groups on long lanes, while on short lanes we need more agile reaction to traffic changes and lane ends.

Within each lane zone, at the beginning of each timestep, Qar-SUMO computes the average speed of the vehicles. It identify the zone as *congested*, if this average speed is under $\alpha \cdot S$, where S is the speed limit of the current edge. α is a configurable parameter. Our evaluation sets this threshold conservatively, at 0, meaning that we only turn on QarSUMO's grouping when a zone have cars not moving at all in the previous timestep.

4.3 Vehicle Simulation under Grouping

If a lane zone is marked as congested, the vehicles currently within this zone forms a virtual group. Within each group, the foremost vehicle naturally becomes the *leader*, who undergoes normal SUMO simulation. The other members of the group, called *followers*, each has its leader attribute set appropriately. For leaders or vehicles not under grouping, this attribute remains NULL.

At each simulation timestep, SUMO goes through several major steps in updating a vehicle's state: (1) *plan move* (where it computes safe speeds for all vehicles for the next a few lanes, and registers approaching vehicle information for all incoming lanes (which are expected to emerge on the current lane), (2) *set junction approaches* (where it registers junction approaches based on planned speeds as basis for right-of-way decision), (3) *execute movement* (where it decides right-of-way and executes movements), and finally (4) *change lanes* (where it processes potential lane changes). Our profiling shows that over 90% of the simulation time is spent on these four steps. QarSUMO adds a preceding step, where it examines the grouping status of the vehicle being updated.

Based on the check result, leaders and ungrouped (individual) vehicles move on with business as usual. The followers, on the other hand, skip most of the first three steps listed above (except for limited metadata updates), as they do not need sophisticated environment assessment and reaction under congestion. Instead, they directly move to step (4), where they simply copy the updated speed from their leaders, and adjust their locations accordingly.

The formed groups are re-examined per timestep. A group is disbanded under either of two circumstances: it fails the congestion speed requirement, or its leader steps into the exit zone. Rather than shifting vehicles across groups, QarSUMO simply recalculates the grouping for the whole lane, which we found to contribute to less than 1% of the total per-timestep simulation overhead.
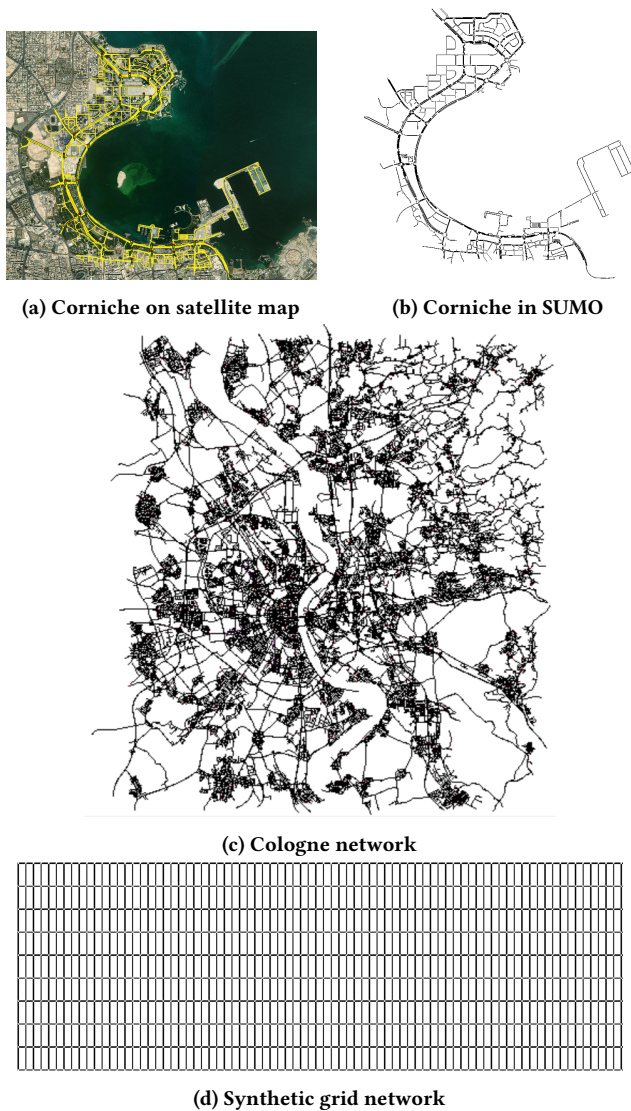
5 EVALUATION

5.1 Experiment Setup

Testbed We conduct all of our experiments on an AWS EC2 c6g.xlarge instance running on Ubuntu 18.04. It has 32 ARM cores, 64GB DRAM, and 32MB L3 cache. Our baseline uses the latest SUMO version (master branch as of June 2020).[3]

Road networks and traffic We use both real and synthetic road networks in our development testing and performance evaluation.

Figure 5a and 5b show the aforementioned Corniche network (satellite map on the left and SUMO-visualized network on the right). The area surrounds the 7.8-km long Al Corniche street (3 lanes in each direction) in Doha and the network we used contains its busiest section, within Doha's West Bay ("downtown" area).

---

[3]https://github.com/eclipse/sumo

(a) Corniche on satellite map

(b) Corniche in SUMO

(c) Cologne network

(d) Synthetic grid network

Figure 5: Networks used: Doha Corniche, Cologne, and synthetic grid

This section contains access to key institutes (such as the Qatar Parliament, the Ministry of Interior, the Ministry of Public Health, the National Mosque), major tourist attractions (such as the famed Museum of Islamic Art, the Souq market, and the scenic Corniche Promenade), and many hotels. Right outside of the network section we have other visitor hot-spots: the Pearl and the Katara Cultural Village at the north and the new Qatar National Museum at the south. Its prominent location and frequent congestion conditions have made the Corniche area an important target in transportation management for the upcoming Qatar 2022 FIFA World Cup event. The event also poses unprecedented traffic challenges, as it will be the first time in World Cup history where all matches are hosted by a single city.

This SUMO road network (Figure 5b) includes the Corniche road and smaller streets joining it, containing 1186 junctions, 5661 edges (7931 lanes in total), and 29 traffic signals. The lanes (counting only a single direction of each road segment) add up to 193 kilometers.

Our parallel simulation scalability test also used a much larger Cologne city network (Figure 5c) available from the SUMO website,[4] with 82399 lanes and 11454 km total length, with synthetic traffic.

To adjust network parameters and scale, we also follow the common practice of using grid networks. Figure 5d illustrates a sample grid map used in our evaluation, which roughly approximates the size and shape of Manhattan. It contains a 150x10 grid, with edge length of 100m (horizontal) and 300m (vertical) respectively.

We generate synthetic traffic on the two road networks with SUMO's built-in tools. First we use randomTrips.py to generate random Original-Destination (OD) pairs, then duarouter to generate routes from the OD pairs. By varying the car insertion rate, we control the traffic intensity and congestion levels.

In all the experiments we use a fine-grain time step of 0.5 seconds.

## 5.2 QarSUMO Parallelization
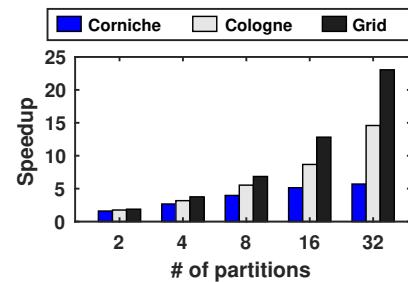
Figure 6: Performance scalability of QarSUMO parallel execution. QarSUMO is able to significantly reduce simulation time with parallelization, while the speedup achieved depends on the road network and traffic.

We start by assessing QarSUMO's parallel simulation performance. Figure 6 shows the speedup over native SUMO's sequential execution. Here we give strong scalability results, where the road network and traffic is fixed while we increase the number of network partitions (and QarSUMO processes).

With the larger and more regular grid network, QarSUMO maintains its efficiency relatively well, achieving a speedup of 23.05 at 32 partitions. With the much less regular Corniche network, however, QarSUMO's performance improves little beyond 8 partitions, reaching only 5.70 at 32. The reason is both due to the complex road layout and the relatively small size of the Corniche graph. Both contribute to making load balancing harder, while the latter also makes synchronization overhead occupying higher weight in overall computation. With the also irregular, but significantly larger Cologne city network, QarSUMO's scalability is between the above two cases, achieving a speedup of 14.6 at 32 partitions.

We confirm our analysis by examining the load balance situation with 32 partitions, for all three networks. Figure 7 shows the

---

[4]https://sumo.dlr.de/docs/Data/Scenarios/TAPASCologne.html
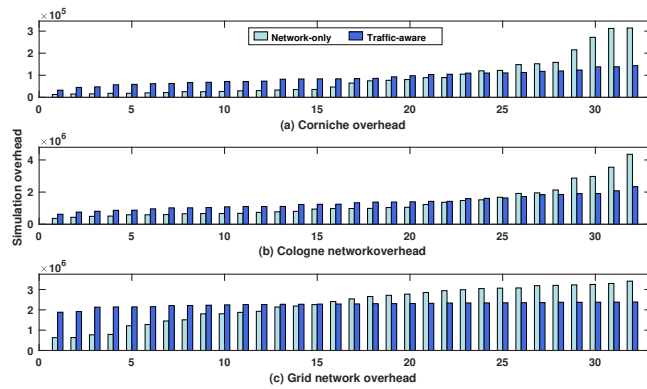
Figure 7: Traffic-aware partitioning results in significant reduction in simulation overhead, for all three networks at 32 partitions.

(sorted) per-partition total simulation overhead, calculated as the total number of simulation steps each partition performed (one simulation step per vehicle per timestep). Compared with the static partitioning, where METIS is fed with only road network topology for communication cost minimization, the traffic-aware partitioning adopted by QarSUMO does significantly improve load balance across the partitions for both networks. Meanwhile, it is evident when comparing the Corniche, Cologne, and grid network results, the former two have much heavier load imbalance, even after traffic-aware partitioning. This reveals challenges in simulation load balancing with irregular real-world networks, which is a topic for future study.

Vehicles	3600	7200	14400	28800
Message Size (MB)	27.41	89.14	377.17	828.30
Communication Time (%)	6.91%	3.24%	1.24%	0.71%

Table 2: Communication overhead, measured by message size and the percentage of time spent on message passing, evaluated on Corniche network with different traffic scale

Next we assess the communication overhead. Table 2 gives the total message size as well as the fraction of simulation time spent on inter-partition message passing, for a 32-partition simulation on Corniche (our least scalable network) with variable number of vehicles. Intuitively, the total message size scales up as more vehicles are simulated. Note that the increase appears beyond linear: when the number of vehicles reaches 28800 (8×), the transmitted message size increases by 30×. This is due to that the increased vehicle volumes slow down traffic, requiring more simulation steps for the vehicles to finish their trips and consequently producing higher total message volumes.

The relative cost of communication, however, remains low and actually decreases with the vehicle volume. With 3600 vehicles, QarSUMO spends only 6.91% of its total execution time on synchronization and communication via MPI, with 27.41MB messages transmitted in total. With 28800 vehicles, such relative overhead of message passing drops to 0.71%, thanks to our aggressive message

batching: more messages get aggregated to the same destination partition, making communication cheaper.
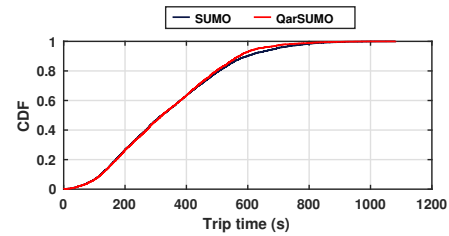
Figure 8: Trip time distribution comparison on Corniche network. The distributions are very similar except a slight increase for QarSUMO for higher trip times.

Figure 8 depicts sample accuracy results of a 32-partition Corniche traffic simulation. Here we give the CDF of all the vehicles' trip time (time to travel from origin to destination), according to the simulation results by SUMO and QarSUMO. As shown in the figure, the systems generate highly similar trip time distributions. The minor differences in simulation results are due to the imperfection of the synchronization across the border edges. For example, the state of the primary vehicle of one timestep can only be propagated to the shadow vehicle in the next step, resulting in one timestep latency (0.5 seconds in our case), which could get accumulated as a vehicle traverses more and more border edges.

| Network/Partition	2	4	8	16	32
Corniche	2.07%	2.45%	4.65%	4.87%	5.46%
Cologne	1.82%	1.97%	2.10%	2.85%	2.53%
Grid	0.91%	1.03%	1.26%	1.45%	1.94%

Table 3: Relative trip time difference between SUMO and QarSUMO

Table 3 gives more detailed accuracy evaluation results. For each test, we compared the trip time of individual vehicles simulated by SUMO and QarSUMO, and calculate the relative difference as their absolute difference divided by the SUMO time. For the Corniche network, the errors starts small, and grow larger as the number of partitions increase, but mostly remain under 5%. The aforementioned tiny discrepancies due to the 0.5-second insertion delay would be accumulated, and render larger differences once in a while at a traffic light. The grid network, on the other hand, is a larger network, rendering lower ratio of edges being border edges, and reduces the overall relative impact of the "simulation drift". As expected, the Cologne network also has accuracy results between Corniche and grid. It is among our immediate future work items to investigate solutions to further reduce parallel simulation inaccuracies.

5.3 QarSUMO Simulation Under Congestion

Next, we evaluate the effectiveness of our simulation optimization under congestion.
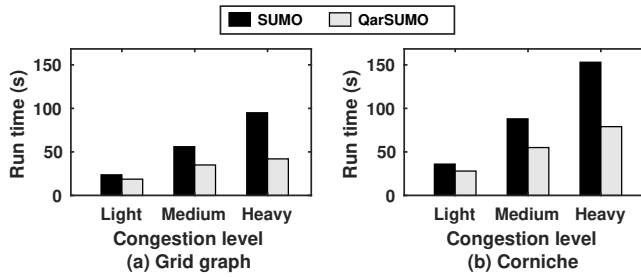
Figure 9: Virtual grouping results in a substantial saving in simulation time for both the Grid and Corniche networks.

Figure 9 gives the execution time of both networks using synthetic traffic that generated different levels of congestion. As it is hard to generate the exact matching levels of congestion across the two networks, we produce three levels of traffic for each network. For Corniche, the "light", "medium", and "heavy" traffic correspond to 40.37%, 64.82%, and 80.33% of vehicle-steps satisfying QarSUMO's strict grouping criterion (0 speed and not within exit zones), respectively. For the grid network, the three levels of traffic have 42.95%, 63.16%, and 82.59% of vehicle-steps meeting the threshold.

Compared with the native SUMO, with its virtual grouping optimization, QarSUMO significantly saves the simulation time by reducing detailed, per-vehicle updates under congestion. As expected, the saving grows as congestion intensifies, obtaining a speedup of 1.93× with Corniche, and 2.26× with the grid network.
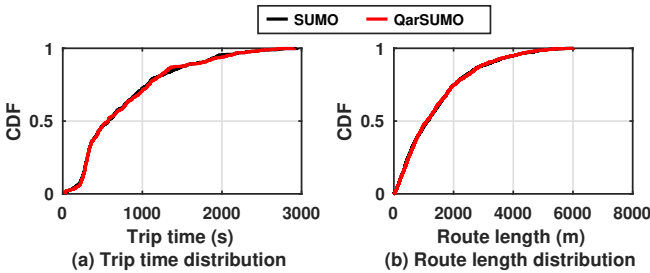
Figure 10: Trip time (for arrived vehicles) and travel distance (for en-route vehicles) CDFs

Figure 10 gives a high-level accuracy comparison between SUMO and QarSUMO. As we add congestion to the experiments and evaluate the congested segments of the simulation runs, many vehicles have not arrived at their destination at the end of the simulation. For the vehicles that have arrived, we draw the CDF of their trip time (Figure 10(a)); for those stuck on the way, we draw the CDF of their traveled distance (Figure 10(b)). Again QarSUMO generates nearly identical distributions as by the native SUMO.

Table 4 calculates the per-vehicle relative difference between SUMO and QarSUMO, for the above two types of vehicles, each sorted by trip time or travel distance. In the majority of cases, the relative difference is within 3%. Like in the parallel execution evaluation, we see higher errors with the larger and signal-dense grid network, where it is easier for slight differences to accumulate into larger per-vehicle diversions.

	Congestion level	Slight	Medium	Heavy
Grid	Trip time difference	4.39%	6.22%	6.22%
	Route length difference	2.28%	8.80%	8.80%
Corniche	Trip time difference	2.98%	2.55%	2.55%
	Travel distance difference	1.43%	1.18%	1.18%

Table 4: Congestion optimization accuracy

5.4 Overall Performance

Finally, we evaluate the combined QarSUMO, with both parallel execution and grouping optimization. Here we perform typical training sample data generation for reinforcement learning (RL), simulating a one-hour episode on both networks. We use a moderate congestion level (between 30% and 40% of vehicles meeting our congestion/grouping criteria), which are of great interest to RL-based policy optimization. Such training typically require hundreds to thousands of such simulation-generated episodes.
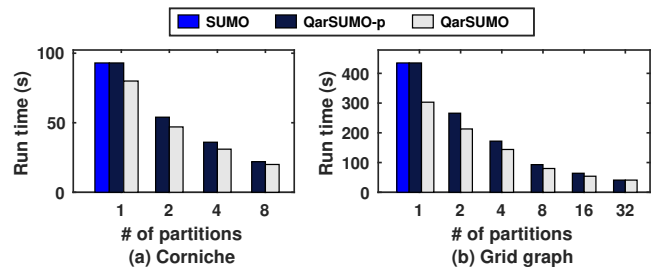
Figure 11: Overall QarSUMO performance in episode generation for RL

Figure11 gives execution time results, where QarSUMO-p denotes QarSUMO with parallelization but not grouping. Considering results shown QarSUMO leveling off beyond 8 partitions with Corniche, here we only test 2 to 8 partitions. We see that QarSUMO delivers considerable speedup (up to 4.22× with Corniche and 10.60× with grid) by parallelization alone.

On top of that, QarSUMO's congestion simulation optimization brings additional time saving (15-30% simulation time reduction from QarSUMO-p). First, here we only have medium congestion level, with under 40% of simulation steps meeting the speed threshold for grouping. Second, with parallelization, the overall simulation time is dominated by the slowest partition for each timestep, making only congestion saving to this particular partition visible.

Together, the two techniques reduces the overall one-hour episode generation time by 4.65× for Corniche, and 10.61× for grid, greatly saving the cost of RL training. For example, the time to generate 500 hours of continuous episodes for the grid network will be reduced from 60.41 hours to 5.70 hours.

Table 5 summarizes accuracy evaluation results, using the same method as in Table 3. Again the grid network delivers higher accuracy (error below 2.03%), while the smaller Corniche network has larger errors (up to 8.21%) with 4 or 8 partitions. In our future work, we plan to evaluate the impact of such approximated simulation on RL learning efficiency.

Network/Partition	2	4	8	16	32
Corniche	2.07%	8.21%	6.6%	-	-
Grid	0.93%	1.02%	1.11%	1.37%	2.03%

Table 5: Relative route length difference between SUMO and QarSUMO, in one-hour episode generation

6 RELATED WORK

Traffic simulation is an important and active topic within transportation research. Prominent traffic simulators that are used both in practice and research include SUMO [16], VISSIM [6], ParamGrid [13], MATSim [24], SMARTS [19] and GeoSparkSim [8]. The SMARTS project [19] proposed a taxonomy of features to compare different simulators including support for distributed computing and decentralized synchronization. The official release of SUMO does not support distributed computing and thus cannot support large-scale simulations.

Two prominent systems that do enable distributed traffic simulation are SMARTS [19] and GeoSparkSim [8]. SMARTS supports master-slave computation model as well as decentralized synchronization, where workers only communicate with other workers that share a road edge. In order to ensure that each simulation timestep finishes within similar time within different slaves, SMARTS uses a workload distribution strategy that creates a number of workers among whom the workload can be nearly evenly distributed. Recall that due to the Car Following Model (CFM) and the Lane Changing Model (LCM), essential to most traffic simulation software, workers will have to coordinate amongst each other when two consecutive cars on a road network end up being assigned to different workers.

GeoSparkSim [8] is a recent proposal that extends GeoSpark [28] to support traffic simulation. GeoSparkSim relaxes a key assumption in SMARTS that the spatial distributions of vehicles remain the same during the simulation. GeoSparkSim introduces VehicleRDD, an in-memory extension of RDDs (Resilient Distributed Dataset) from Spark [29], which also records spatio-temporal information of vehicles. As the vehicular traffic moves in the network the VehicleRDDs get transformed to maintain a spatio-temporal workload balance.

Several attempts have been made to address the lack of multithreaded and distributed computing support in SUMO, such as Spartsim [23] and dSUMO [4]. Like QarSUMO, the core idea of these methods is also based on network partitioning. However in both proposals the overhead of keeping the compute nodes synchronized and maintaining a balanced workload outstrips the inherent advantages of distributed computing. QarSUMO also uses METIS for network partitioning, but provides better scalability than reported by the above systems as it reduces the inter-partition synchronization overhead. It also avoids expensive dynamic load balancing by performing traffic-aware network partitioning using the available traffic profile.

There is also a parallel implementation of the TRANSIMS microsimulation tool [18]. It uses cellular automata (CA) to model driving dynamics (while SUMO adopts the more powerful continuous model). Like QarSUMO, it also adopts METIS-based network partitioning and edge-based inter-partition communication. It is able to improve load balancing by collecting load statistics from a prior run, targeting the scenario that similar traffic situations are repeatedly simulated. QarSUMO, instead, can take into account routing information for the current simulation for its load balancing.

Another recent study [3] parallelizing SUMO only targets grid networks and cuts the network horizontally or vertically to smaller rectangles. Similarly, the recently developed Cityflow traffic simulator, also targeting reinforcement learning training, is also limited to grid networks. QarSUMO, on the other hand, tackles real world networks which often have irregular shapes, which makes the synchronization much more challenging.

Besides parallelization, higher microscopic simulation efficiency has also been achieved by considering the varying congestion level in different areas and switching from a constant time-step, queuebased simulation to an event-driven simulation [5]. Approaches like this aimed at reducing the computation time in very low traffic areas where few traffic events occur. In contrast, QarSUMO proposes a novel technique to optimize the computation time in highly congested areas, which account for the heaviest workload in SUMO. To our knowledge, our virtual grouping method is the first in reducing simulation cost for congested traffic.

7 CONCLUSION

In this paper, we present QarSUMO, an enhancement to the widely used SUMO open-source traffic simulation software. QarSUMO enables SUMO to take advantage of powerful multi-core server nodes, by adding high-level parallelization outside of its simulation kernel. It also improves SUMO's simulation efficiency under severe traffic congestions, by reducing vehicle update granularity. Our evaluation with real and synthetic networks confirm that QarSUMO is able to bring a speedup of up to a factor of five over the native SUMO performance in realistic scenarios, while maintaining reasonable simulation accuracy.

Our immediate future work plan includes comprehensive testing and code cleaning, as well as further simulation accuracy improvement, to prepare for QarSUMO's open-source release. Also, we are working on pre-processing the full traffic network for Doha, to enable much larger scale (in space and time), city-wide traffic simulation using QarSUMO. For future research, we plan to investigate dynamic load balancing (without network re-partitioning) and more detailed/flexible congestion detection schemes, as well as further simulation optimization designed specifically for reinforcement learning.

8 ACKNOWLEDGMENTS

REFERENCES

[1] 2020. *SUMO Projects Page.* https://sumo.dlr.de/docs/Other/Projects.html
[2] 2020. *SUMO User Conference.* https://www.eclipse.org/sumo/conference/
[3] Nicolás Arroyo, Andrés Acosta, Jairo Espinosa, and Jorge Espinosa. 2018. A new strategy for synchronizing traffic flow on a distributed simulation using SUMO. *EPiC Series in Engineering* 2 (2018), 152–161.
[4] Quentin Bragard, Anthony Ventresque, and B.E. Murphy, Liam. 2013. dSUMO: towards a distributed SUMO. *The first SUMO User Conference, Berlin, Germany, 15 - 17 May, 2013* (2013).
[5] David Charypar, Kay W Axhausen, and Kai Nagel. 2006. An event-driven queue-based microsimulation of traffic flow. *Arbeitsberichte Verkehrs-und Raumplanung* 406 (2006).
[6] Martin Fellendorf and Peter Vortisch. 2010. Microscopic traffic flow simulator VISSIM. In *Fundamentals of traffic simulation.* Springer, 63–93.
[7] Zishan Fu, Jia Yu, and Mohamed Sarwat. 2019. Building a large-scale microscopic road network traffic simulator in apache spark. In *2019 20th IEEE International Conference on Mobile Data Management (MDM).* IEEE, 320–328.
[8] Zishan Fu, Jia Yu, and Mohamed Sarwat. 2019. Demonstrating geosparksim: A scalable microscopic road network traffic simulator based on Apache spark. In *Proceedings of the 16th International Symposium on Spatial and Temporal Databases.* 186–189.
[9] Edgar Gabriel, Graham E Fagg, George Bosilca, Thara Angskun, Jack J Dongarra, Jeffrey M Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, et al. 2004. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting.* Springer, 97–104.
[10] Denos C. Gazis, Robert Herman, and Richard W. Rothery. 1961. Nonlinear Follow-The-Leader Models of Traffic Flow. *Operations Research* 9, 4 (1961), 545–567.
[11] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20, 1 (1998), 359–392.
[12] Nishant Kheterpal, Kanaad Parvate, Cathy Wu, Aboudy Kreidieh, Eugene Vinitsky, and Alexandre Bayen. 2018. Flow: Deep reinforcement learning for control in sumo. *EPiC Series in Engineering* 2 (2018), 134–151.
[13] Raymond Klefstad, Yue Zhang, Mingjie Lai, R Jayakrishnan, and Riju Lavanya. 2005. A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation. In *Proceedings. 2005 IEEE Intelligent Transportation Systems, 2005.* IEEE, 813–818.
[14] Daniel Krajzewicz, Georg Hertkorn, Christian Rössel, and Peter Wagner. 2002. SUMO (Simulation of Urban MObility)-an open-source traffic simulation. In *Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002).* 183–187.
[15] Lior Kuyer, Shimon Whiteson, Bram Bakker, and Nikos Vlassis. 2008. Multiagent Reinforcement Learning for Urban Traffic Control Using Coordination Graphs. In *Machine Learning and Knowledge Discovery in Databases*, Walter Daelemans, Bart Goethals, and Katharina Morik (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 656–671.
[16] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie WieBner. 2018. Microscopic traffic simulation using sumo. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC).* IEEE, 2575–2582.
[17] Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. 2017. Traffic light control using deep policy-gradient and value-function-based reinforcement learning. *IET Intelligent Transport Systems* 11, 7 (2017), 417–423.
[18] Kai Nagel and Marcus Rickert. 2001. Parallel implementation of the TRANSIMS micro-simulation. *Parallel Comput.* 27, 12 (2001), 1611–1639.
[19] 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 Transactions on Intelligent Systems and Technology (TIST)* 8, 2 (2016), 1–22.
[20] Stefano Giovanni Rizzo, Giovanna Vantini, and Sanjay Chawla. 2019. Time Critic Policy Gradient Methods for Traffic Signal Control in Complex and Congested Scenarios. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* 1654–1664.
[21] John A Sokolowski and Catherine M Banks. 2011. *Principles of modeling and simulation: a multidisciplinary approach.* John Wiley & Sons.
[22] Zheng Tang, Milind Naphade, Ming-Yu Liu, Xiaodong Yang, Stan Birchfield, Shuo Wang, Ratnesh Kumar, David Anastasiu, and Jenq-Neng Hwang. 2019. Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 8797–8806.
[23] Anthony Ventresque, Quentin Bragard, Elvis S Liu, Dawid Nowak, Liam Murphy, Georgios Theodoropoulos, and Qi Liu. 2012. Spartsim: A space partitioning guided by road network for distributed traffic simulations. In *2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications.* IEEE, 202–209.
[24] Kay W Axhausen, Andreas Horni, and Kai Nagel. 2016. *The multi-agent transport simulation MATSim.* Ubiquity Press.
[25] Hua Wei, Chacha Chen, Guanjie Zheng, Kan Wu, Vikash Gayah, Kai Xu, and Zhenhui Li. 2019. PressLight: Learning Max Pressure Control to Coordinate Traffic Signals in Arterial Network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* 1290–1298.
[26] Hua Wei, Guanjie Zheng, Vikash Gayah, and Zhenhui Li. 2019. A Survey on Traffic Signal Control Methods. arXiv:1904.08117 [cs.LG]
[27] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. 2018. IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (London, United Kingdom) *(KDD '18).* ACM, New York, NY, USA, 2496–2505.
[28] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. 2015. Geospark: A cluster computing framework for processing large-scale spatial data. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems.* 1–4.
[29] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: cluster computing with working sets. In *HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing.* 10–10.