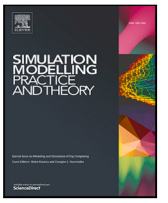
Simulation Modelling Practice and Theory 126 (2023) 102775

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat



PATRIC: A high performance parallel urban transport simulation framework based on traffic clustering

Lin Wan [a], Ganmin Yin [b,c], Jiahao Wang [a], Golan Ben-Dor [d], Aleksey Ogulenko [d], Zhou Huang [b,c,*]

[a] School of Geography and Information Engineering, China University of Geosciences, Wuhan, China
[b] Institute of Remote Sensing and Geographical Information Systems, Peking University, Beijing, China
[c] Beijing Key Lab of Spatial Information Integration & Its Applications, Peking University, Beijing, China
[d] Department of Geography and Human Environment, Porter School of Environmental and Earth Science, Tel Aviv University, Israel

ARTICLE INFO

Keywords:
Parallel simulation
Agent-based transportation simulation
Road network partition
Traffic cluster
MATSim

ABSTRACT

Parallel traffic simulation requires partitioning the road network into several components that can be assigned to different computing nodes (CPNs). Existing studies focus more on reducing edge-cuts (message-passing pipes between CPNs) to decrease synchronization message amongst CPNs for efficiency improvement. However, even reducing edge-cuts drastically, the volume of messages transmitted might still be high, which does not significantly improve performance. Based on observation that some *traffic clusters* (TCs) exist during simulation, i.e., areas with high internal and low external traffic density. For high-performance urban transport simulation, we propose a data-driven parallel approach named PATRIC, which can generate parallel partitions automatically based on traffic clustering. Specifically, the *TC-based automatic partitioner* (TAP) is designed to automatically identify TCs and then construct partitions in parallel. We present a partition-growing algorithm that prevents traffic-intensive TCs being split across multiple CPNs when distributing computing workloads, resulting in more balanced load and fewer synchronization operations. Unlike prior work using fixed thresholds for load balancing, we develop the *adaptive partition updater* (APU) to fit the dynamic traffic in the road network, which achieves a better trade-off between balancing workload and lowering communication for higher efficiency. Experiments on real-world datasets demonstrate that our approach outperforms the state-of-the-art methods.

1. Introduction

Multi-agent systems (MAS) possess an inherent ability to represent agents' behaviour and decision-making [1]. Through agent-based modelling and simulation (ABMS), MAS can evaluate the dynamics of a society of computer agents with highly heterogeneous individual behaviour. Agents are discrete and autonomous, and each has its own goals and can adapt behaviour to varying conditions. Given its flexibility, MAS has been widely applied in different scientific fields such as biology, economics, transportation, business, system engineering, and so on [2].

With their inherent traffic and transportation problems, cities are the objects of spatially explicit MAS modelling inquiry. When considering spatially explicit transportation infrastructures, the agent population should be sufficiently large to understand the collective dynamics of this complex system. However, the larger the city is, the higher is the demand for model performance. Such

* Corresponding author at: Institute of Remote Sensing and Geographical Information Systems, Peking University, Beijing, China.
E-mail address: huangzhou@pku.edu.cn (Z. Huang).

https://doi.org/10.1016/j.simpat.2023.102775
Received 31 October 2022; Received in revised form 27 February 2023; Accepted 2 May 2023
Available online 6 May 2023
1569-190X/

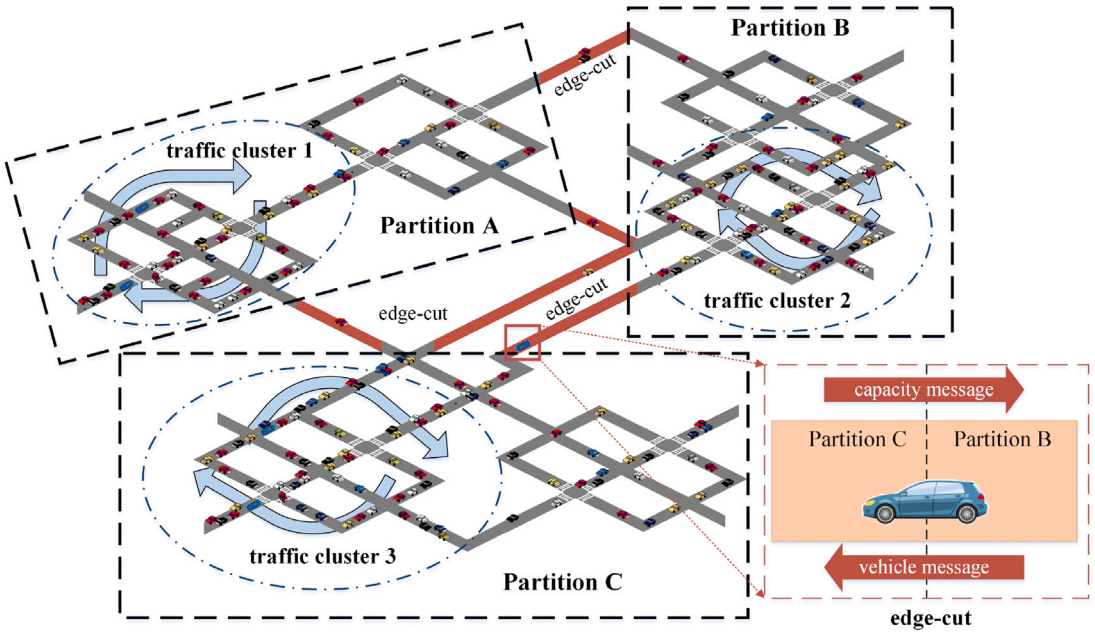


Fig. 1. An example of three traffic clusters.

relationship between the city size and computational cost is fundamentally super-linear, since each agent (a traveller or vehicle) will interact with many other agents when moving in space. Thus, studying transportation dynamics in metropolitan areas with millions of travellers becomes a significant challenge. Consequentially, it is essential for the modeller to achieve the balance between the limited computational resources (e.g., hardware scarcity) and urgent simulation demand (i.e., massive agents constantly interacting with each other in a highly heterogeneous urban space).

After analysing the state of large-scale MAS applications, Parry and Bithell proposed two practical approaches to cope with systems that exceed the capabilities of the current hardware [3]. The first one, hardware upgrading, does not require re-programming but may increase the financial cost while not providing any guarantee of success [2]. The second approach, parallelization, requires advanced computing skills to handle the overhead of communication among the cores. Typically, parallel solutions are problem-specific [4] but, if successful, they can take advantage of the cloud computing infrastructure.

This paper focuses on parallel urban traffic simulation, which requires dividing the transportation network into several parts and assigning them to a set of computation nodes (CPNs), such as physical CPU cores [5]. Each CPN independently uses its local resources to process intra-partition agents' travel schedules. When agents move between different CPNs, interactions (i.e., messages carrying information about vehicles' properties or trajectories, CPN's capacity, etc.) between CPNs will occur. Such communication may bring additional overhead [6–8] and incur a bottleneck on simulation performance [9–11] in real-world scenarios.

For above problems, considerable attention has been devoted to increasing the degree of parallelism and reducing the communication overhead based on graph partitioning techniques [10–13]. Several studies have contributed to generating a balanced partition scheme [14,15] for improving the degree of parallelism. However, they consider load balancing and internal communication overhead reduction separately, neglecting the interplay between them. Only seeking an even workload distribution throughout parallel partitions is no help for lowering cross-CPN agent synchronizations (and sometimes even counterproductive), and vice versa. Some researchers believed that cutting down the amount of messages exchanged between border nodes can effectively accelerate the parallel simulation process. For instance, Leng and Potuzak [12,16] introduced the topological structure to partition road networks and achieved promising results by reducing *edge-cuts*, which can be regarded as "message-pipes" among CPNs [17]. However, reducing such "message-pipes" does not mean cutting down the volume of synchronization messages transferred in the remaining pipes, and may not lead to actual reduction in the overall communication cost. Some work used fixed empirical thresholds such as load imbalance value to trigger partition updating during simulation [10,18], which may not better adapt to spatiotemporal dynamics of the traffic.

In urban transportation scenarios, the spatial distribution of traffic presents unevenness and changes over time [19–21]. Consequently, the computational workload of each CPN may differ with some CPNs having a large computational load while others might be idle. This brings an imbalanced workload among CPNs which may cause insufficient parallelism. Furthermore, the unbalanced distribution of traffic could generate massive agent migrations across several CPNs, incurring additional communication costs.

We observe the occurrence of the above two situations as related to specific areas of dense traffic with a large number of internal agents. We call these areas *traffic clusters* (TCs), see Fig. 1. In reality, TC is common, on the one hand, it may be derived from the

natural existence of certain "communities" (or cluster structures) in the road network, on the other hand, the traffic distribution changing over time will cause traffic hotspots at certain times (such as transportation hubs, commercial centres, etc. at peak hours). By TC's definition, the workload of CPN mainly comes from internal TCs' traffic. Distributing TC across multiple CPNs will bring massive interactions between parts. Therefore, TCs' determination and allocation are the crucial factor that affects the performance of the parallel simulation.

In this paper, we propose a new approach to parallel simulation from the perspective of TCs and propose a high-performance parallel transport simulation framework, namely PATRIC (Parallel urban transport simulation framework based on Traffic Clustering). It not only optimizes workload distribution but also reduces synchronization operations among parallel nodes, substantially boosting the overall performance of the simulation system. Unlike previous work [22,23], PATRIC can decrease the total messaging rather than the number of edge-cuts, empirically speeding up the simulation process. Different from prior fixed partition methods [12,24], PATRIC is capable of dynamically adjusting parallel partitions according to the variation of TC distribution, better adapting to the dynamics of traffic during the simulation. Compared to current state-of-the-arts, the proposed approach has higher simulation performance.

The main contributions of the study can be summarized as follows: (1) a parallel urban transport simulation framework based on data-driven traffic clustering is proposed; (2) by achieving a better distribution of agents across the CPNs, our approach greatly reduces the unnecessary synchronization operations during the parallel simulation process and shows superior performances compared to prior edge-cuts reduction methods; (3) to our best knowledge, our work is one of the few studies aiming at adaptive parallelization for multiagent-based transport simulation; (4) experiments on a real-world scenario demonstrate that PATRIC outperforms state-of-the-arts.

The remainder of this paper is organized as follows: Section 2 describes related works of agent-based simulation and parallel simulation. The description of the parallel agent-based traffic simulation is provided in Section 3. Section 4 presents the TC-based parallel simulation approach. Section 5 discusses the experiments on a real-world traffic network. Section 6 summarizes the whole work and puts forward some inspirations for future work.

2. Related work

2.1. Multi-agent transportation simulation and the challenge of scaling

An agent-based simulation is widely used for evaluating complex traffic systems [9]. The core of transportation simulation consists of 1) a traffic flow model that quantifies the relationship between traffic flow, density, and speed, and 2) modelling the interactions among different agents [25]. Within a transportation network, there may be several routes between each origin and destination for an agent [26]. During the simulation, agents must choose the optimal route to minimize travel time and distance.

Based on this idea, various simulation frameworks have been proposed [25,27–32]. Lopez et al. [33] introduced the simulator coupling, model development, and validation on the open-source simulator SUMO. Ehlert et al. [34] explored a mesoscopic simulation method that focuses more on traffic junction control. For on-demand mobility services, an agent-based simulator is presented for users to evaluate the performance of different traffic optimization schemes [35]. Auld et al. [36] discussed the development of agent-based modelling software which can achieve dynamic simulation of travel demand, network supply, and network operations. These agent-based simulations can model travellers' behaviour and evaluate the traffic status. However, simulating thousands of individuals in large cities would consume a lot of computing resources and bring high computational costs. Therefore, more and more attention has been paid to parallelization for accelerating the simulation process [25].

We have integrated PATRIC into the Multi-Agent Transport Simulation (MATSim) model [37], which is an open-source leading system for agent-based multimodal transportation simulation. MATSim accepts urban infrastructure data as a set of GIS layers and simulates urban traffic focusing on individual travel choices and behaviour. MATSim agents adapt to the existing service conditions through learning and self-correction towards achieving a User Equilibrium (UE) steady state. MATSim is chosen to integrate PATRIC into it due to its inherent ability to simulate traffic in large areas. In a comparison against other 83 ABMs in various fields, MATSim was described as one of the most developed environments under limited hardware conditions. Besides, MATSim was the only transportation ABM ranked at the top level ("extreme-scale") due to its adaptive ability to simulate millions of agents over vast urban networks [38]. For model scaling, MATSim downscales the target population [39], i.e. one agent represents several, and model parameters are adjusted to represent the full-scale dynamics correctly.

MATSim avoids an explicit simulation of a car-following behaviour. The mobility simulation module (MOBSim) of MATSim offers the user several simulation engines. The default engine, QSim, employs a *First-In-First-Out* (FIFO) principle for simulating how vehicles traverse road links. In QSim, a link is considered a gate, and a car entering it at one end is added to the tail of the vehicle queue building up at the other end. The vehicle remains in this queue until it reaches the queue head (Fig. 2(a)). Thus, no car can enter a link if the *storage capacity* – the maximal number of vehicles that can be physically allocated on a link – is exhausted. The traversal time of a link depends on its current traffic conditions [37], which, in turn, are defined by *flow capacity* – the maximal number of cars that exit a link per time unit. Since both storage and flow capacity are limited, cars queueing on a link of a limited capacity can generate traffic congestion.

Fig. 2(b) illustrates how the *flow capacity* and *storage capacity* parameters influence traffic flow. Looking from left to right, the following will happen between moments $t = 0$ and $t = 1$: since the flow capacity of *link1* is 2 cars/min, two cars will try to exit *link1* and proceed to *link2*; since the flow capacity of *link2* is also 2 cars/min, two vehicles will try to leave *link2* and move to *link3*; and, since the flow capacity of *link3* is 1 cars/min, one car will exit it. Looking from right to left, possible transitions from *link2* to *link3*

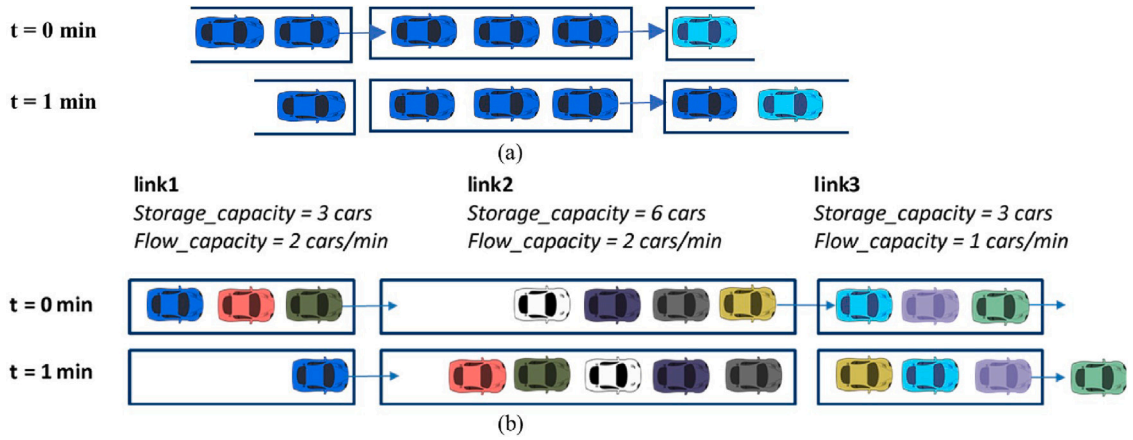


Fig. 2. Traffic microsimulation in MATSim. (a) Illustration of the QSim FIFO rule of vehicle advancement on one link and between two consecutive links. (b) Effects of MATSim link storage capacity and flow capacity on the simulated traffic flow.

would not be fully executed since only one unit of the storage capacity on *link3* will be emptied. At the same time, the transitions from *link1* to *link2* will be implemented in full since, after one car, the unexploited storage capacity of *link2*, which would transfer to *link3*, will be equal to three cars.

Such queue-based approach substantially improves QSim's performance compared to other microsimulations that consider direct vehicle interactions [40]. In particular, the QSim approach prohibits the volume-to-capacity ratio (V/C) from exceeding 1. In contrast, the traditional models that use an impedance function permit conditions where V/C is greater than one as applied in environments such as EMME [41], TransCad [42], or Cube [43].

More recently, Graur et al. [44] proposed a new mobility simulator called HERMES, which is twice as fast as QSim, that serves as an alternative for QSim. It employs an event-driven approach and includes data structure optimization and can work for multimodal scenarios, simulating both private cars and public transit traffic.

In MATSim, agents' behaviour is described by spatiotemporally explicit activities, also called plans, which are input for MATSim. This *activity-based* approach has been exploited in several other models (e.g., ALBATROSS [45]; FEATHERS [46]; CEDMA [47]) that aim at the integration of agents' mobility at different spatiotemporal levels. Plans are implemented with the MOBSim and Scoring simulation modules. MOBSim executes agents' trips and enables the adaptation of the agents to the changing traffic conditions. Innovation can also be introduced in the MOBSim module through randomly selecting some agents to potentially change or adjust their routes, modes, or departure time. The performance of each agent is *scored* in the Scoring module using pre-established utility functions. For those agents that get improved scores due to the innovation, they would accept and maintain such changes. Conversely, agents whose score decreases will reject it. The changes in agents' behaviour entail the changes of traffic. After certain iterations, this process of travellers' co-adaptation will converge to a state of user equilibrium (UE). At the day when the UE state is reached, no agent can improve its score by changing its behaviour. This process is known as the MATSim *daily loop* (Fig. 3).

Specifically, for transportation modelling, Dobler [48] demonstrated that the multithreaded parallel version of the MATSim is faster than the sequential one for a lower number of cores (less than 12). However, a further increase in cores does not decrease the simulation runtime. GPU-based parallel processing pushes this threshold somewhat further but is also limited [49]. In addition, multi-threading is also limited in scalability compared to multi-processing approaches [50], which makes it difficult to apply to large-scale multi-agent simulation.

2.2. Parallel acceleration for transportation simulation

Parallel simulation is a key method that can be utilized in improving efficiency for urban traffic simulation. Several studies have already contributed to performing traffic simulation on a distributed cluster [14,51,52]. For instance, Cordasco et al. [53] presented D-Mason, which can overcome the limitation on the maximum number of agents during simulation. For another example, Liu and Meng [54] used a distributed computing system for accelerating Monte Carlo simulation.

The previous work can be categorized into two groups: (1) traffic model optimization by designing a complete message-passing mechanism and an effective synchronization method to reduce the communication overhead between different CPNs. (2) domain decomposition to split a road network and assign sub-networks to CPNs, reducing synchronization costs and improving load balance [25].

Parallel simulation can be improved by reducing the synchronization time interval, where reducing lookahead is commonly used [55–57]. Xu et al. [57] used dead-reckoning to predict the next positions of vehicles. Fu et al. [55] proposed a method to further provide the trade-off between simulation precision and the execution time using the free margin. They took advantage of the uncertainty in traffic simulation to improve time resolution, but the simulation accuracy may be affected.

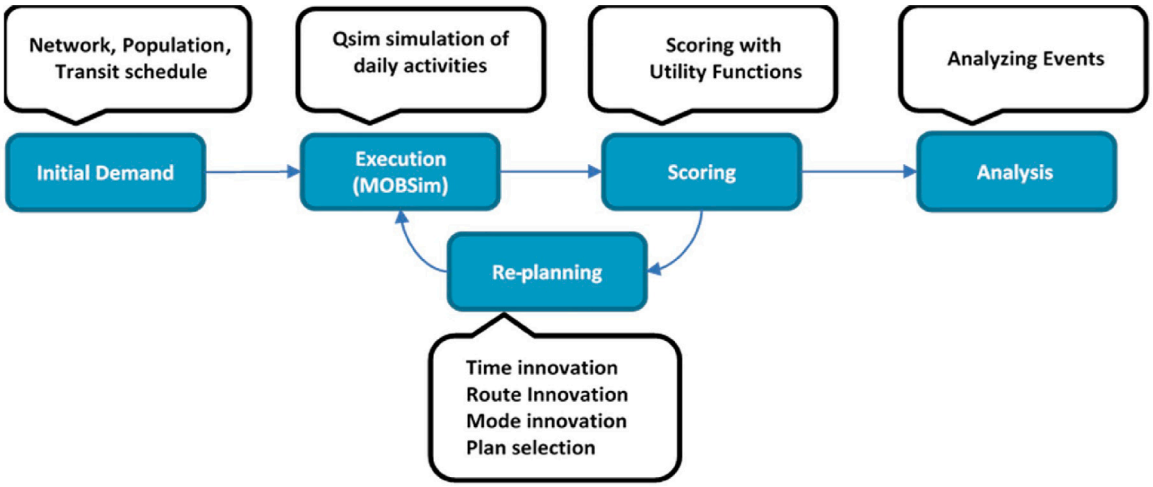


Fig. 3. Flowchart of the MATSim daily loop.

Appropriate road network partitioning is another important aspect of parallel traffic simulation. The result of partitioning significantly impacts synchronization overhead and load balancing, thereby affecting the overall efficiency. Many studies were committed to improve the efficiency of parallel simulation by designing an optimal partition scheme. In early research, domain decomposition were often implemented by cutting a road network into stripes or grid cells using coordinate information [58–60], which neglects the serial messages among CPNs. Some researchers have achieved good simulation acceleration results by applying graph partition algorithms to address the problem of messages transmission among CPNs [61]. Graph partition approaches can be categorized as stream-based, distributed, and dynamic methods [10,11,62,63].

In a stream-based method, nodes are read sequentially and allocated to a specific CPN until the simulation ends. Stanton [64] proposed a linear deterministic greedy (LDG) heuristic algorithm and obtained outstanding results. A simple block-based graph partition was applied by Zhu et al. [65], which divides the vertex stream into k consecutive vertex blocks. Xu et al. [66] proposed another graph partition method based on log information. The partial reflow method [67] uses the multi-path reflow method to partition the loading part of the graph. Patwary et al. [68] proposed a window-based streaming graph partitioning algorithm to realize efficient processing of graph information. In addition, new tools such as game theory are also used for partitioning and has achieved a satisfying effect [18].

Distributed partitioning that incorporates label propagation methods is another commonly used approach. As Rahimian et al. [69] showed, the initial partitions are randomly generated with load balance. In each recursion, each vertex selects one neighbour to form a vertex pair and exchange their labels. However, the operation of label exchanging increases the number of edge-cuts. To overcome this shortcoming, Chen and Li [70] proposed a method allowing only two partitions to exchange their vertices simultaneously. The partitions of the road network generated by either stream-based or distributed methods are fixed and inevitably have the limitations for failing to respond the dynamic traffic distribution in the simulation process.

Dynamic partitioning methods gain much attention for their adaptabilities to dynamic traffic, which makes up for the defects of steam-based and distributed methods [61]. A multi-level approach (i.e., METIS) [71] had been applied to parallel traffic simulation and performed well in reducing the number of edge-cuts [10,12]. Buffoon [13] is another multi-level graph partitioning algorithm that uses natural cuts in road networks as a preprocessing technique to obtain a coarser graph. Compared to METIS, Buffoon generates fewer edge-cuts. However, it is slower than METIS, making it less suitable for dynamic graph partitioning. Object function is usually introduced to update partitions dynamically, of which load balancing and the number of edge-cuts are the most commonly considered factors [10,72–79]. Vaquero et al. [79] presented an algorithm named XDGP which attempted to reduce communication overheads. External communication was taken into account by GPS [78], while X-pregel [80] considered both external and internal communication. Additionally, Catch was proposed by Shang and Yu [81], which considered computational loads during the whole simulation process. Onizuka et al. [77] combined the first-fit algorithm to achieve low communication costs and better load balancing.

Recent studies tended to consider the accessed frequency of vertices for graph partitioning [22]. A parallel balanced graph partitioning framework called JA-BE-JA-RIK [22] was presented. It improved the original heuristic JA-BE-JA [69] algorithm by evaluating the richness of implicit knowledge (RIK) to obtain fewer edge-cuts and high knowledge cohesion. HBP [23] partitioned the graph by distributing vertices with discrimination according to their hotness, aiming to distribute them among CPNs evenly. Based on the network's internal structure, Bai et al. [82] presented a novel two-stage optimization method for network community partition. Focusing on the field of emergency evacuation, Yin et al. [11] put forward ViCTS algorithm for graph partitioning to improve the scalability of ABM.

Although most existing methods seek to diminish the edge-cuts to cut costs on synchronizations among CPNs, such strategies could not really decrease the volume of agents' synchronization messages transferred on edge-cuts. So far, how to efficiently reduce the total synchronization operations and minimize idle-wait time among CPNs, remains an open question.

3. Problem statement

In multi-agent traffic simulation, a collection of autonomous agents will interact with each other, denoted by $\{a_1, \ a_2 \dots, a_n\}$, and they can be considered as mobile objects such as vehicles, buses, or pedestrians. These agents move in a road network that is typically represented as a graph $G(V, \ E)$, where the set of vertices $V = \{v_1, \ v_2 \dots, v_k\}$ represents road network junctions intersected by a set of road links $E = \{e_1, \ e_2 \dots, e_m\}$. Each agent a_i $(1 \le i \le n)$ follows its own daily travel schedule, denoted by a origin–destination (OD) pair (O_i, D_i). During the simulation, a_i moves on the road network from its origin O_i to destination D_i, attempting to minimize the travel cost comprised of time and distance. Competition can occur during the interaction among the agents as the transportation infrastructure is limited. In this situation, the traffic may surge in some components of the road network. For serial simulation approaches, long delay may occur to the agents processing in the "busy" parts of the network, leading to poor system efficiency.

Parallel approaches can accelerate the simulation process by taking advantage of contemporary multicore hardware. Specifically, the road network is partitioned into N disjoint spatial components $G_1, \ G_2 \dots, G_N$. Then, a separate parallel computation node CPN_i is assigned for each $G_i (1 \le i \le N)$ to execute the traffic generated by agents residing in it. The partitioning scheme $G = \{G_1, \ G_2 \dots, G_N\}$ is critical for reducing execution time compared to a sequential approach. If an improper G is chosen, it might incur severe load imbalance, where many CPNs are idle wait for overloaded CPNs finishing their tasks. Furthermore, frequent cross-CPN's synchronizations could also occur when G splits heavy-traffic roads into different CPNs, which consumes vast amount of processing resources, as a great deal of agents on edge-cuts need to synchronize their information such as vehicles' capacity, speed, size, etc. with the target CPNs, while the targets would send back road congestion information to the initiators for determining whether the agents need to wait or cross. Let t and c denotes an agent's synchronization time and processing time, respectively. The partitioning problem can be formulated as an optimization task seeking an argument G to minimize the following objective function

$$\begin{aligned} \min_{G} \quad & t \cdot Flow_e(G) + c \cdot \{\max P(G) - \min P(G)\} \\ s.t. \quad & G = \{G_1, G_2, \dots, G_N\} \end{aligned} \tag{1}$$

where $Flow_e(G)$ denotes the total traffic flows through edge-cuts in G, $P(G)$ is the set of workloads for each component of G. The objective function defined by Eq. (1) consists of two parts: the overall synchronization time and the idle wait time among CPNs. Solving this multi-objective optimization problem is challenging because these two objectives are not independent from each other. To our knowledge, most current studies focused either on load-balance based solutions or edge-cut reduction approaches, yielding suboptimal rather than globally optimal solutions. Some recent works [12,16] on synchronization reductions proposed to use static partition schemes to reduce the number of "message pipes" among CPNs, only reducing synchronization messages locally not globally, nor decreasing the idle wait time. In this paper, we consider simultaneously optimizing the two objectives and present a heuristic solution to solve the problem in Eq. (1).

4. Traffic cluster-based parallel urban transport simulation

In this paper, we propose PATRIC, a parallel urban transport simulation framework based on traffic clustering, which simultaneously reduces the number of messages among CPNs and keeps the dynamic load balancing during the simulation, effectively accelerating the parallel simulation performance.

4.1. Traffic cluster

In conventional parallel simulation approaches, a road network is usually divided into several parts with approximately equal traffic to balance a load and assign each part to a separate CPN [83]. Although simulation performance can be improved in such load-balanced way, it overlooks the considerable communication costs caused by agent synchronizations among CPNs. Recently, there were attempts to alleviate this problem by reducing the number of edge-cuts [10,16]. However, only diminishing the number of edge-cuts does not mean cutting down the volume of messages passing on these channels. Traffic concentrated areas (i.e., Traffic clusters) in parallel simulations often distributes across multiple components. For example, Traffic cluster D in Fig. 4 overlaps with components 2, 3, and 4, resulting in a great deal of intra-traffic-cluster messages (represented by the thickness of the red lines in Fig. 4) on partitions' edge-cuts, which consumes large amounts of computing resources, degrading the whole simulation performance.

To address this challenge, we propose a TC-based partitioning solution, which prioritizes TCs being not split by components as much as possible. Based on that, we simultaneously consider reducing cross-CPN's message transfer and maintaining load balancing among CPNs from a global perspective, leading a highly efficient parallel simulation. Furthermore, by partition re-assignment algorithm of edge-cut nodes, our TC-based approach owns the capability of prompt adaptation to the traffic fluctuations during the simulation.

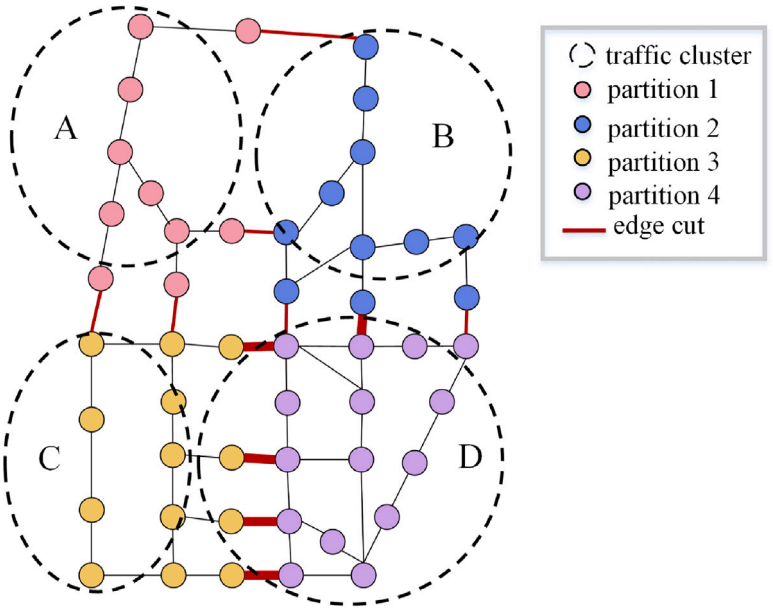


Fig. 4. Traffic clusters and parallel partitions in a road network.

4.2. The PATRIC framework

4.2.1. Outline of PATRIC

The framework of PATRIC consists of two main modules: *TC-based automatic partitioner* (TAP, Fig. 5(a)) and *adaptive partition updater* (APU, Fig. 5(b)). The former generates initial partitions in which traffic flow is evenly distributed and minimizes the splitting of TCs across CPNs. For example, if there is a road network, as illustrated in the top-left of Fig. 5(a), TAP would automatically search some representative nodes that summarizes current TCs in the network, we called them *Attractors*. Based on that, a *partition growing algorithm* is conducted to 'attract' nodes around *attractors* into local TCs, thereby generating all possible partitions according to the network traffic distribution. This process generates the initial partitioning $\{P_1, P_2, P_3, P_4\}$ for parallel simulation. During the simulation, traffic flow changes in the road network would often lead to workload fluctuations. Prior works usually attempt to adjust partitions for retaining load balance. However, this operation may cause TCs to span across multiple CPNs, incurring more synchronization communication. To address this issue, APU re-assigns edge-cut nodes into their adjacent partitions to seek decreasing the traffic flows in their edge-cuts or partitions, iteratively updating the partitioning scheme until finding the optimal solution that can minimize the overall computational overhead of the system. For instance, as shown in Fig. 5(b), after several rounds of re-partition iterations, we get the updated partitions for the traffic flow change, in which edge-cut nodes 5 and 6 are re-partitioned from P_1 and P_2 to P_3 and P_2, respectively.

4.2.2. Automatic partition-grow algorithm

To generate TC-based parallel partitions, PATRIC first adopts the local traffic density of road-network nodes to determine TCs. As shown in Fig. 5(a), we propose a density-based algorithm (Algorithm 1) to search current TC *attractors* (i.e., local maximum traffic density nodes) in the road network. This algorithm can automatically determine the number of components without needing repeated trials. Considering that both road network nodes and vehicles contain location information, we treat them as homogeneous points in the attractor discovery process to obtain the traffic $\mathcal{T}_{v_i}$ surrounding each road-network node v_i. To estimate the traffic density at a node v_i, we define a circle search "window" for it, that is, a circle centred at v_i with radius r. We can set $r = Q_\delta(S')$, which chooses the $(100 \times \delta)$th quantile of the road-network distance distribution S' (line 4–5), and δ is the scaling factor (empirically set to 0.01) that controls the length of r. The density $\mathcal{F}_{v_i}$ then is estimated as the traffic sum of nodes lying within the circle window centred at v_i with radius r (line 6–10 in Algorithm 1). Considering that TC *attractors* are high traffic density nodes surrounded by neighbours with lower density, while having a much longer distance from other nodes with higher density. We scale $\mathcal{F}_{v_i}$ using the minimal distance from v_i to nodes with higher density as the weight $\mathcal{D}_{v_i}$ (line 12). Note that for the highest traffic density node in the road network, we define its weight as the maximal node-wise distance to other nodes. In this way, we get a weighted $\mathcal{G}_{v_i}$ for each v_i and then a weighted traffic density vector $\mathcal{W}$. Apparently, the most likely TC attractors are nodes with large value of $\mathcal{G}$. Hence, we reorder $\mathcal{W}$ in descending order of $\mathcal{G}$, and traverse it to successively select attractor nodes that have longer distance and higher traffic density than their neighbours, that is, they should simultaneously satisfy the two conditions:

$$\mathcal{D}_{v_k} \geq 2 \cdot \sigma(\mathcal{D}) \tag{2}$$

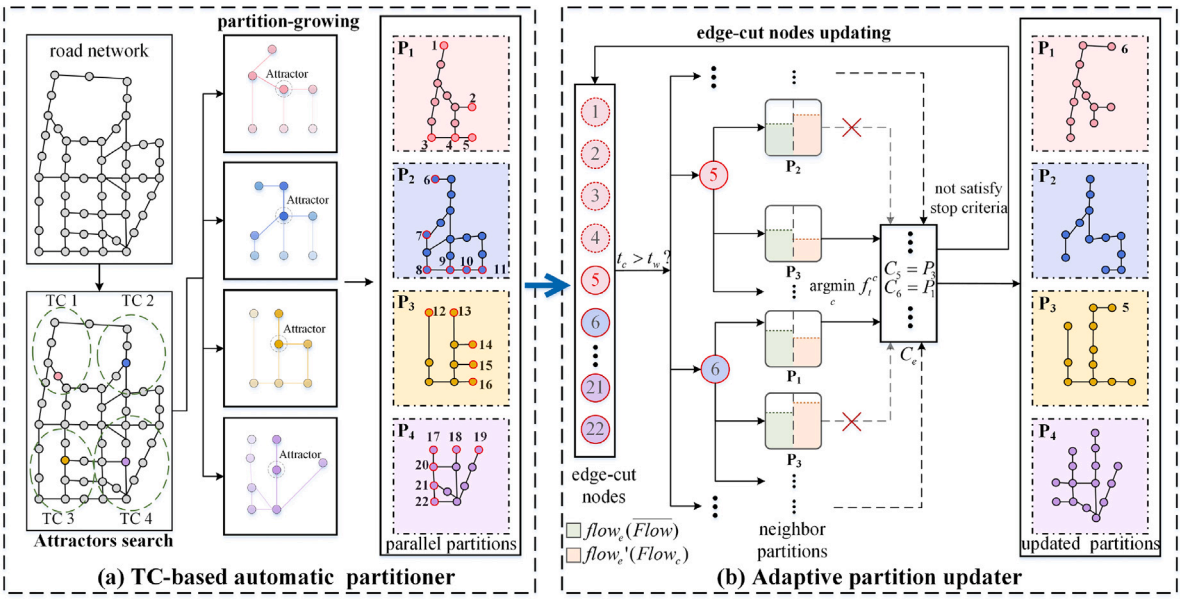


Fig. 5. The overall framework of PATRIC.

$$\mathcal{F}_{v_k} \geq \mu(\mathcal{F}) \tag{3}$$

where $\mathcal{D}_{v_k}$ is the minimal distance of node v_k (cf. line 12), and $\mathcal{F}_{v_k}$ denotes the traffic density of v_k. We use $\sigma(D)$ to represent the standard deviation of the distance distribution D, and $\mu(\mathcal{F})$ the mean for the density distribution $\mathcal{F}$. We can easily find the first v_k that does not satisfy either of the above conditions by traversing $\mathcal{W}$, and then choose the first k−1 nodes in $\mathcal{W}$ to form TC attractor set $\mathcal{A}$ for the road network (line 17–20). The rest of nodes can be assigned to TCs which are represented by their neighbour attractor nodes.

Fig. 6 shows an example of TC attractors searching on a road network. In Fig. 6(a), road-network nodes present different local traffic density at some point in the simulation process. Fig. 6(b) shows that our algorithm eventually identifies four TC attractors (node 3, 6, 10, and 14). Compared with other nodes, these attractors have higher local traffic density and are far away from each other.

Instead of simply assigning remaining nodes to their nearest attractors, a weighted partition-growing algorithm (Algorithm 2) is proposed to generate the initial partitions by considering load balancing and reduction of synchronization messages among CPNs. Starting with the attractors, partition components are expanded one by one along with the links of the road network. This process can be accelerated by allowing each processor to parallelly grow partitions for attractor nodes. Specifically, for the partition generation task assigned to processor k, we assume that the neighbours of attractor A_k is given as $\mathcal{L}$. We repeatedly find the maximal traffic node v^* in $\mathcal{L}$ and add unvisited v^* to the partition (line 8–12), until $\mathcal{L}$ is empty or the CPN's work load P is above the average $\bar{P}$. If v^* has been allocated to another CPN (e.g., $\mathcal{R}(\mathcal{A}_j)$ in line 13), we compare the total traffic between v^* and both partitions to decide whether it needs to be re-assigned (line 13–15). Note that each time we access v^*, we always remove it from $\mathcal{L}$, and once v^* has been added to current partition, apart from updating the partition's load (line 19), $\mathcal{L}$ would also be updated by adding all the unvisited neighbours of v^* (line 20–21). In this way, for each TC attractor $\mathcal{A}_k$, we add to the partition $\mathcal{R}(\mathcal{A}_k)$ all the road-nodes that are attracted by it, yielding the initial partitioning $\mathcal{C}$ with balanced partition load and fewer cross-CPN's synchronization messages.

4.2.3. Adaptive partition updating

During the simulation, the traffic flow on road networks always changes over time, which constantly varies the structure of TCs and renders some CPNs heavily loaded. In this scenario, fixed partitioning would possibly create performance bottlenecks in the parallel simulation system. To evaluate the impact of the partition scheme on simulation performance at a certain moment t, we propose the following metric:

$$f_t = t_c + t_w \tag{4}$$

Here, we consider two factors that affect the efficiency of parallel simulation: the communication overhead t_c and the waiting time of CPNs t_w:

$$t_c = p_c \cdot \sum_{e \in E} flow_e \tag{5}$$

Algorithm 1 TC attractors search algorithm

Input: set of road-network nodes, $\mathcal{V}$; scale factor, δ
Output: set of traffic cluster attractors, $\mathcal{A}$

$\mathcal{A} \leftarrow \varnothing$
$\mathcal{T}_{v_i} \leftarrow$ sum of traffic for all edges adjacent to v_i,for all nodes $v_i \in \mathcal{V}$
$\mathcal{D}_{v_i} \leftarrow 0, \mathcal{G}_{v_i} \leftarrow 0, \mathcal{W}_{v_i} \leftarrow \varnothing, \mathcal{F}_{v_i} \leftarrow \mathcal{T}_{v_i}$ for all $v_i \in \mathcal{V}$
$\mathcal{S} \leftarrow \left\{ \left\| v_i - v_j \right\| : v_i, v_j \in \mathcal{V} \right\}$
//Compute road-network distance matrix
$r \leftarrow Q_\delta \left(\mathcal{S}' \right)$,where $\mathcal{S}' \leftarrow \left\{ S_{ij} \middle| S_{ij} \in \mathcal{S}, i < j \right\}$
//Radius of the circle search window
for each $v_i \in \mathcal{V}$ **do**
  for each $v_j \in \mathcal{V} \backslash \{v_i\}$ **do**
    if $\mathcal{S}(v_i, v_j) < r$ **then** $\mathcal{F}_{v_i} \leftarrow \mathcal{F}_{v_i} + \mathcal{T}_{v_j}$
    end if
  end for
end for
for each $v_i \in \mathcal{V}$ **do**
  $\mathcal{D}_{v_i} \leftarrow min \left\{ \mathcal{S}\{v_i, v_j\} \middle| v_j \in \mathcal{V} \backslash \{v_i\} \text{ and } \mathcal{F}_{v_j} > \mathcal{F}_{v_i} \right\}$
  $\mathcal{G}_{v_i} \leftarrow \mathcal{F}_{v_i} * \mathcal{D}_{v_i}$
  $\mathcal{W}_{v_i} \leftarrow \left(v_i, \mathcal{G}_{v_i} \right)$
end for
Sort $\mathcal{W}$ in descending order of $\mathcal{G}$
Traverse $\mathcal{W}$, find the first occurrence of $\left(v_k, \mathcal{G}_{v_k} \right)$, *such that* $\mathcal{D}_{v_k} < 2 \cdot \sigma(D)$ or $\mathcal{F}_{v_k} < \mu(\mathcal{F})$
for each $\mathcal{W}_i = \left(v_i, \mathcal{G}_{v_i} \right) \in \mathcal{W}$, $i \le k - 1$ **do**
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{v_i\}$
end for
return $\mathcal{A}$

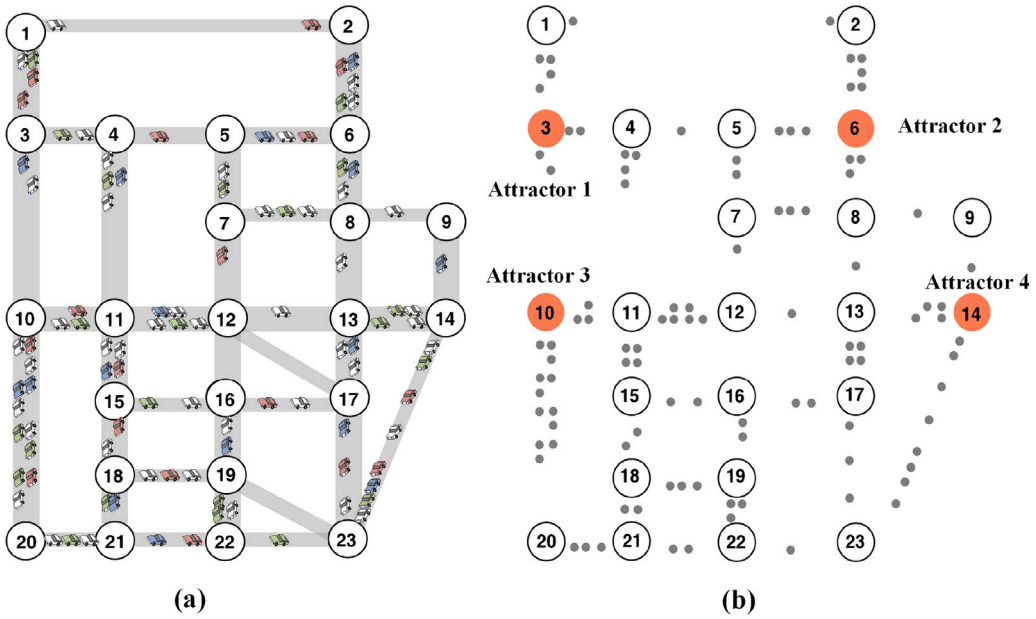


Fig. 6. An example of TC attractors search algorithm: (a) Road nodes with different local traffic density in an urban traffic network, (b) Four attractors are found based on traffic flow densities of road nodes.

Algorithm 2 TC-based partition-growing algorithm.

Input: road-network nodes, $\mathcal{V}$; set of node-traffic, $\mathcal{T}$;
set of TC attractors, $\mathcal{A}$
Output: initial partitions, $\mathcal{C}$

$\mathcal{C} \leftarrow \varnothing$
$\mathcal{R}(\mathcal{A}_i) \leftarrow \mathcal{A}_i$, mark $\mathcal{A}_i$ as *visited*, for all $i = 1\ to\ |\mathcal{A}|$
$\bar{P} \leftarrow \sum_{i=1}^{|\mathcal{V}|} \mathcal{T}_i / |\mathcal{A}|$ // Average load per partition
for each processor $k = 1, \ldots, |\mathcal{A}|$ **do**
  $P \leftarrow 0$
  $\mathcal{L} \leftarrow \mathcal{N}(\mathcal{A}_k)$ // Neighbour node set of $\mathcal{A}_k$
  while $\mathcal{L}$ is not empty and $P \leq \bar{P}$ **do**
    $v^* \leftarrow argmax_v \{T_v | v \in \mathcal{L}\}$
    // Find the maximal traffic node v^* in $\mathcal{L}$
    $\mathcal{L} \leftarrow \mathcal{L} \setminus \{v^*\}$
    if v^* has not been *visited* **then**
      $\mathcal{R}(\mathcal{A}_k) \leftarrow \mathcal{R}(\mathcal{A}_k) \cup \{v^*\}$
      mark v^* as *visited*
    else if $v^* \in \mathcal{R}(\mathcal{A}_j)$ and $j \neq k$, *such that*
        $flow_{v^*,\ \mathcal{R}(\mathcal{A}_k)} > flow_{v^*,\mathcal{R}(\mathcal{A}_j)}$ **then**
      $\mathcal{R}(\mathcal{A}_k) \leftarrow \mathcal{R}(\mathcal{A}_k) \cup \{v^*\}$
      $\mathcal{R}(\mathcal{A}_j) \leftarrow \mathcal{R}(\mathcal{A}_j) \setminus \{v^*\}$
    else continue
    end if
    $P \leftarrow P + flow_{v^*,\ \mathcal{R}(\mathcal{A}_k)}$
    for each $v' \in \mathcal{N}(v^*)$, *such that*
        v' has not been *visited* **do**
      $\mathcal{L} \leftarrow \mathcal{L} \cup \{v'\}$
    end for
  end while
  $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{R}(\mathcal{A}_k)$
end for
return $\mathcal{C}$

$$t_w = p_w \cdot (\max(F) - \min(F)) \tag{6}$$

where p_c is the time of transferring a vehicle through an edge-cut e between CPNs; $flow_e$ denotes the traffic flow on e; p_w is the computation time for processing a vehicle on a CPN; $F = \{F_1, F_2, \ldots, F_n\}$ denotes a set of traffic flows in each component. When the number of vehicles on roads dramatically increases, intensive traffic among different components might produce excessive synchronization messages, incurring a heavy communication cost. The communication overhead t_c can be measured by the transmission time of cross-CPN vehicles. The surging workload may spread within a couple of components, possibly causing severe load imbalance as some idle nodes would wait heavily-loaded ones to complete their tasks for a very long time. The waiting time t_w can be estimated by calculating the difference of processing time between the largest and the smallest CPN flows. A higher f_t indicates existing partition scheme cannot adapt well to the traffic change at time t, which brings large computational overhead, cutting down parallel simulation performance. The lower the f_t is, the more current partition can fit the variation in traffic.

In the simulation process, we continuously monitor the changes of f_t during the simulation and make the decision whether current partition scheme should be updated by the following condition:

$$f_t - f_l > \alpha \cdot f_l \tag{7}$$

When the difference between current f_t and the value after the last partitions update (i.e., f_l) is above the threshold defined by $\alpha \cdot f_l$ (α is a scale factor, empirically set to 0.15 through performance analysis), the parallel framework needs to update parallel partitions to rebalance the workloads between CPNs and reduce the increased synchronization operations brought by TC changes. If t_c is higher than t_w, the communication overhead has a greater impact on f_t. When updating partitions, we give priority to merging cross-CPN TCs as much as possible. Likewise, when t_w has a bigger effect on f_t, rebalancing is prioritized for lessening the idle time of low-loaded CPNs. It is worth noting that regardless of which factor is prioritized, the treatment to one would also affect the other. For instance, merging TCs is likely to incur workload imbalance that increases some CPNs' waiting time, while rebalancing operation might bring more synchronization traffic among CPNs. Hence, it is unviable to achieve desired efficiency gains by separately dealing with load imbalance and excessive communication overhead when performing partition updates.

Here we propose an adaptive update algorithm (APU) to automatically adjust the parallel partitions to suit traffic flow changes in road networks, which considers not just the volume reduction in messages exchanged by cross-CPN agents, but the workload

Table 1
Details of Pingshan road network used for simulations.

Scenario	Network	Nodes	Links	Agents	Scale
a	Pingshan	35 K	67.4 K	154 K	Large

balance among CPNs. As shown in Algorithm 3, APU updates partitions via two key steps: *candidates generation* and *optimal partition selection*. No matter which factor we prioritize, partition update always starts from the alteration of components that edge-cut nodes belong to. When reducing communication overhead preferentially (i.e., $t_c > t_w$), for each edge-cut node v_e on a border link with high communication traffic, we first calculate the total edge-cut traffic $flow_e$, and then re-assign v_e to each neighbour partition, respectively. If the substitution could reduce the total traffic flows on v_e's edge-cuts, the partition is added into the candidate partition set C^* (line 12). Similarly, when the system performance is seriously affected by unbalanced load, we check whether the workload of v_e's CPN is above average (line 15), and if so, underloaded adjacent partitions are selected to generate C^* for v_e (line 16–18). Notably, consider the interplay between balance-first and TC-first re-partition strategies mentioned above, we do not separately adopt either of them to conduct a further screening on C^*. Instead, we propose to determine the optimal partition according to its ability to reduce the overall computational overhead. Specifically, a f_t based partition search strategy (line 24) is advanced to attain the updated partition C_e for v_e. It first calculates f_t^c for each candidate c in C^*, respectively. From partitions that satisfy the following condition: $f_t^c < f_t$, we choose the one with a minimum of f_t as the re-assigned partition of v_e. In this way, all nodes that need to be re-partitioned would be assigned to locally optimal components. Note that apart from updated partitions, each run of APU also updates the edge-cut node set. We iteratively refine the updated partitions by conducting the algorithm on the updated node set, until the partitioning can minimize the total computation overhead.

Here we propose two stop conditions for the iteration. If system overhead is dominated by excessive synchronization messages, we use the *synchronization time reduction ratio* φ to decide whether to terminate the loop, which can be phrased as follows

$$\varphi = 1 - \frac{t_c}{t'_c} < \varepsilon \tag{8}$$

where t'_c is the total communication cost at time t, t_c is the cost after several rounds of iterations. When φ is lower than a certain threshold ε, the iteration terminates. Likewise, if the simulation performance is largely degraded by severe unbalanced loads among CPNs, we use the *degree of load balance* ρ for current partitioning to decide whether to stop the iteration, which is given as

$$\rho = \frac{\max(F)}{\bar{F}} \leq \lambda \tag{9}$$

where $\bar{F}$ denotes the average traffic flow of each component. When ρ is less than or equal to a partition capacity parameter λ, partition updating stops. In our work, ε and λ are empirically set to 0.0025 and 1.05, respectively.

5. Experiment and results

We implement PATRIC in Java and have integrated it into the MATSim transport simulator [37]. The communication among CPNs is realized using the MPJ Express library [84]. During synchronization, when the agent reaches the boundary of a partition, it triggers the message sending, and the CPN of the targeted partition will receive the message and update the state every 2 seconds. To evaluate our proposed approach, we compare PATRIC with three mainstream state-of-the-art methods: METIS [71], Stripe [58], and NRGG [10]. The experiments are conducted on a computing workstation that has an Intel Xeon Gold 6230 2.10 GHz CPU with 20 cores and 640 GB memory.

5.1. Study area

We conduct experiments on the real-world road network of Pingshan, a district of Shenzhen that is one of the most rapidly developed cities in China [85]. Pingshan has comprehensive commercial facilities, residential areas, and industrial zones. In addition, it possesses a typical dense road network with 35 K road nodes and 67.4 K road links (Table 1). Besides, it is estimated that about 154 K residents travel in Pingshan in November 4th 2019 (Monday) from 6 a.m. to 24 p.m. Fig. 7 depicts the location and road network of the study area.

5.2. Results analysis

5.2.1. Traffic clusters

Fig. 8 describes 4 locations identified by the TC attractors search algorithm and traffic flow around them. These attractors are the places with high local traffic density in Pingshan, including Pingshan railway station, Shenzhen No.9 People's Hospital, New Bridge Industrial Zone, and Universiade Shenzhen Gymnasium.

After identifying attractors, the partition-growing algorithm mentioned in Section 4.2.2 is used to generate the initial partition. The resulting partition is shown in Fig. 9.

Algorithm 3 The partition updating algorithm.

Input: edge-cut node to be updated, v_e;
current total communication overhead, t_c;
current waiting time of all CPNs, t_w
Output: re-assigned partition for v_e,C_e

$\mathcal{L} \leftarrow \mathcal{N}(v_e)$ // Neighbour node set of v_e
$flow_e \leftarrow 0, C^* \leftarrow \varnothing$
if $t_c > t_w$ **then**
  for each $v \in \mathcal{L}$, *such that* $C(v) \neq C(v_e)$ **do**
    $flow_e \leftarrow flow_e + flow_{v,v_e}$
  end for
  for each $c \in C(\mathcal{L})$ **do**
    $flow'_e \leftarrow 0$
    for each $v \in \mathcal{L}$, *such that* $C(v) \neq c$ **do**
      $flow'_e \leftarrow flow'_e + flow_{v,v_e}$
    end for
    if $flow'_e < flow_e$ **then** $C^* = C^* \cup \{c\}$
    end if
  end for
else
  if $F_{C(v_e)} > \bar{F}$ **then**
    for each $c \in C(\mathcal{L})$, *such that* $F_c < \bar{F}$ **do**
      $C^* = C^* \cup \{c\}$
    end for
  end if
end if
if $C^* = \varnothing$ **then**
  $C_e = C(v_e)$
else
  $C_e = argmin_c \{f_t^c | c \in C^*,\ f_t^c < f_t\}$
end if
return C_e

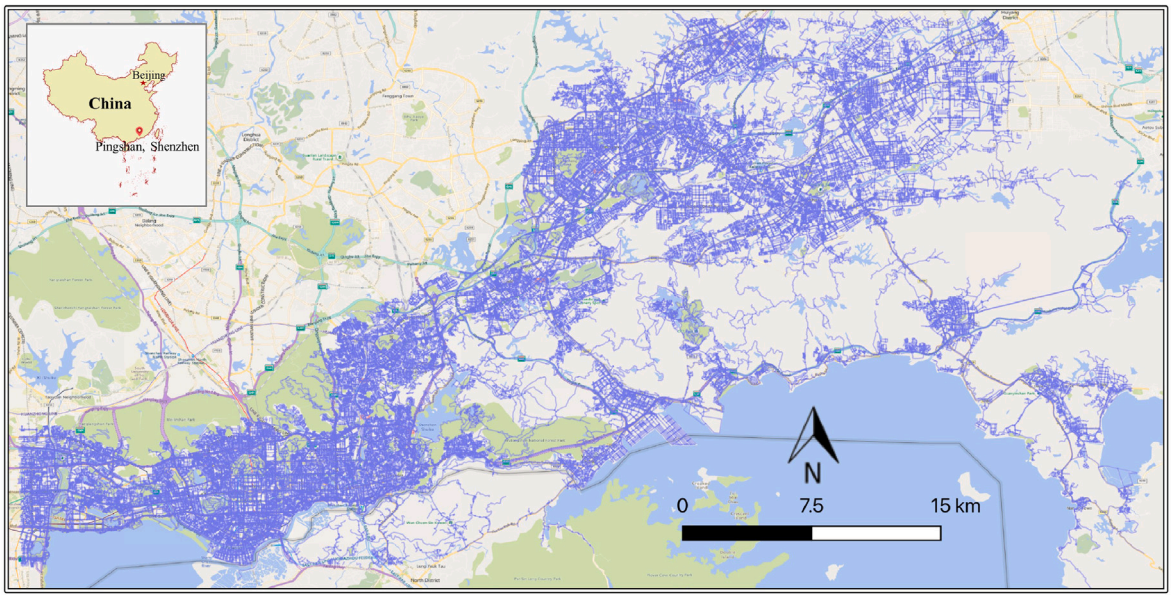


Fig. 7. Location and road network of the study area.

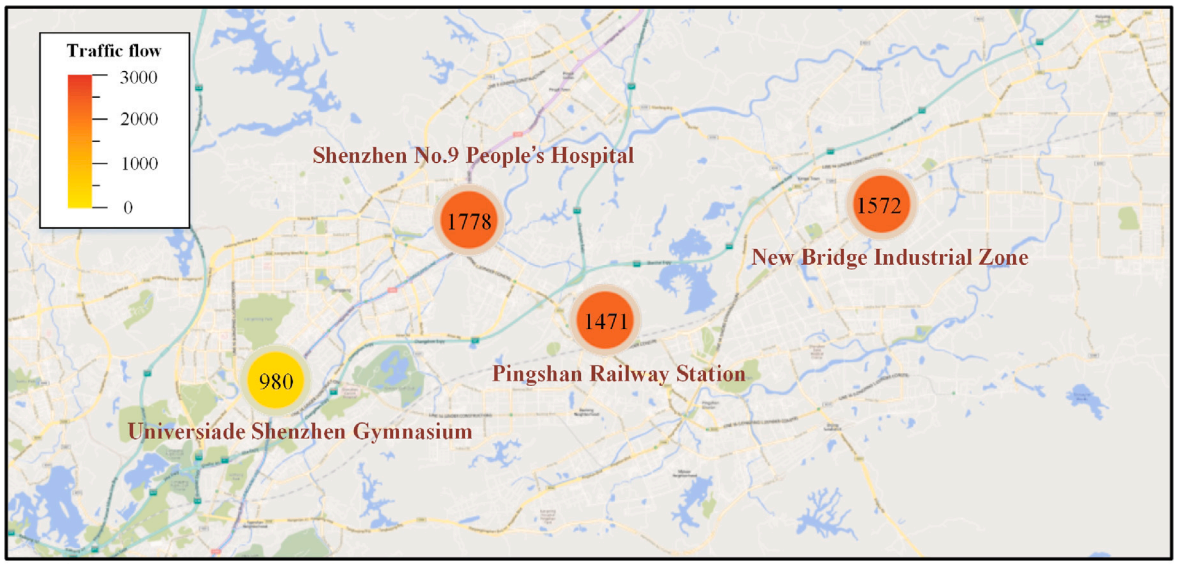


Fig. 8. Four TC attractors and the traffic flow around them.

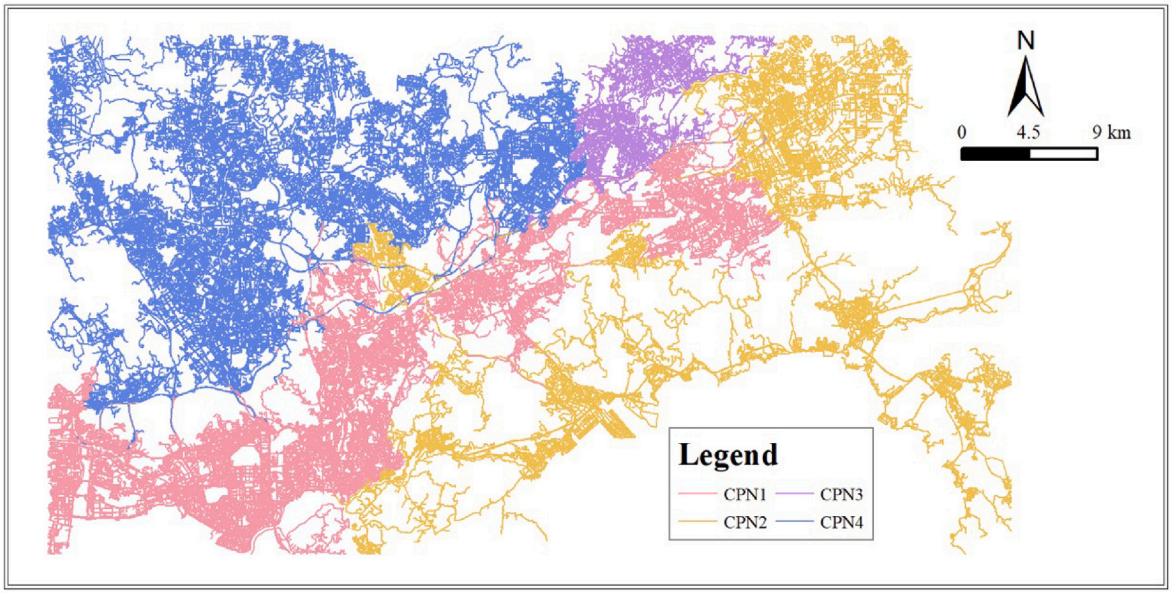
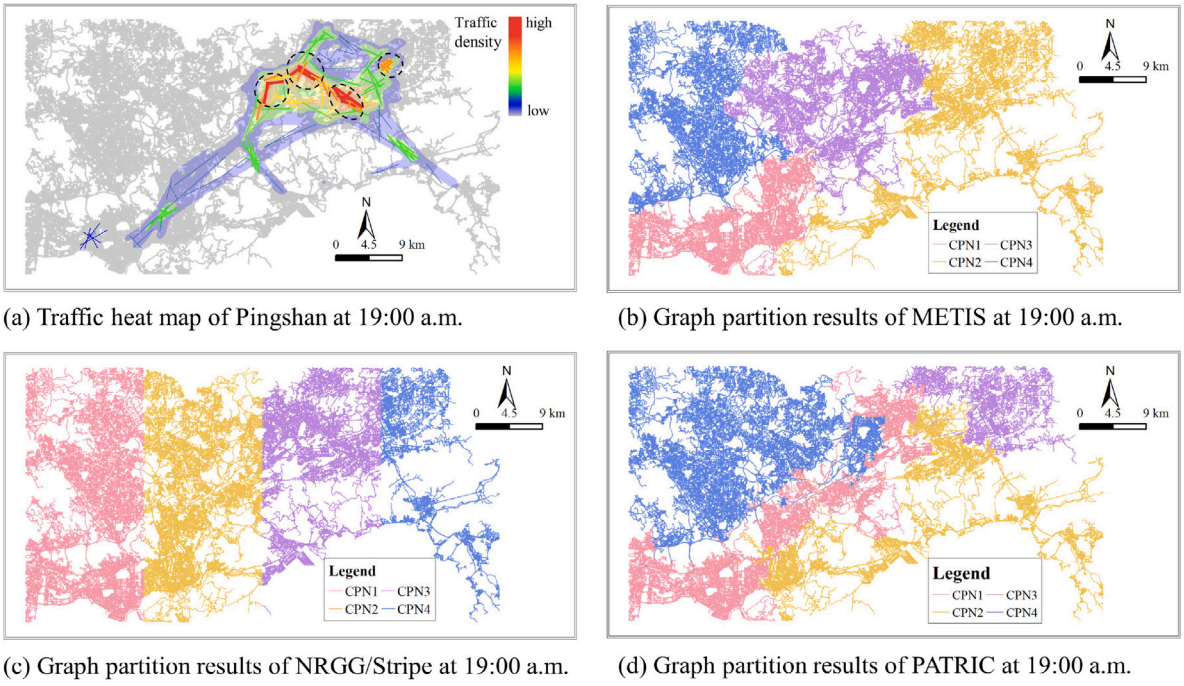


Fig. 9. Initial partition result of Pingshan road network.

As the initial partition shows, the southwest components are relatively sparse while the northeast part is denser. This aligns well with the fact that there is more traffic in the northeast region than in the southwest. The northeast of Pingshan is a gathering place for a large number of industrial parks with many nearby residential areas. Therefore, commuting flows are mainly concentrated in the northeast, causing higher traffic volumes and denser partitions. As for the southwestern part of Pingshan, it mainly contains tourist attractions, of which traffic density is relatively low during weekdays.

5.2.2. Parallel partitions

As described in Section 4.2.3, the initial partition can be updated to fit the traffic dynamics during the day, keeping components' load and a total number of messages at an appropriate level. Taking the traffic situation of Pingshan at 19:00 p.m. as an example, the partitions produced by METIS, NRGG/Stripe, and PATRIC are depicted in Fig. 10 (NRGG and Stripe are mentioned together because they generate similar partitioning shapes).



(a) Traffic heat map of Pingshan at 19:00 a.m.

(b) Graph partition results of METIS at 19:00 a.m.

(c) Graph partition results of NRGG/Stripe at 19:00 a.m.

(d) Graph partition results of PATRIC at 19:00 a.m.

Fig. 10. Traffic heat map and partitioning of Pingshan District at 19:00 p.m. on Nov 4, 2019.

Table 2
Comparation with other methods. The message volume (θ), workload imbalance degree (σ) and the total overhead (f_t) are reported.

Methods	Message volume (θ)	Imbalance degree (σ)	Overhead (f_t)
METIS	56780	126.3047	419.7606
Stripe	55547	122.5509	248.0940
NRGG	55527	121.5	230.1773
PATRIC	**41616**	**97.2943**	**200.594**

It can be seen from Fig. 10(a) that there are 4 TCs (dotted circles) in Pingshan during the evening peak. The traffic within each TC is much denser than the transit traffic between TCs. Fig. 10(b) and Fig. 10(c) draw the partitions generated by METIS and NRGG/Stripe, respectively. METIS creates components of irregular shapes based on a coarse-grained load balancing, whereas NRGG/Stripe creates roughly striped shapes. Both methods split TCs and are insufficiently adaptive to the traffic changes. Particularly, the latter leads to heavy inter-component messaging and slows down the traffic simulation. As for the partition produced by PATRIC (Fig. 10(d)), it is consistent with the location and distribution of TCs, achieving a fewer number of inter-component messages. Consider, for example, the component assigned to CPN3. Comparison with the initial partition in Fig. 9 demonstrates that PATRIC correctly extended this component to accommodate traffic changes in TCs.

5.2.3. Load balancing and inter-CPN communication

After the partitioning of the road network in the Pingshan District, we conduct the parallel simulation with PATRIC and measure its performance. Workload imbalance degree, the number of synchronization messages among CPNs (adding up the number of messages sent and received during each synchronization cycle), and overhead (i.e., f_t described in Section 4.2.3) act as three important factors of simulation efficiency. During the simulation, loads of each partition are recorded every 10 min, and the CPN's load variance σ is computed to estimate the imbalance degree:

$$\sigma = \frac{1}{N}\sum_{i=1}^{N}\left(F_i - \bar{F}\right)^2 \tag{10}$$

where N is the total number of partitions, $\bar{F}$ denotes the average traffic flow of each component. The lower σ is, the workload can be more evenly distributed among each CPN, since the computing resources of each CPN are roughly equal. The total amount of messages, overhead and workload imbalance accumulated by different approaches in the simulation process under the same CPN number are counted and shown in Table 2.

As shown in Table 2, we compare PATRIC with three other methods in terms of message volume, workload imbalance degree and total overhead. The number of synchronization messages decreased by PATRIC is 25%, 25%, and 27% less than NRGG, Strip,

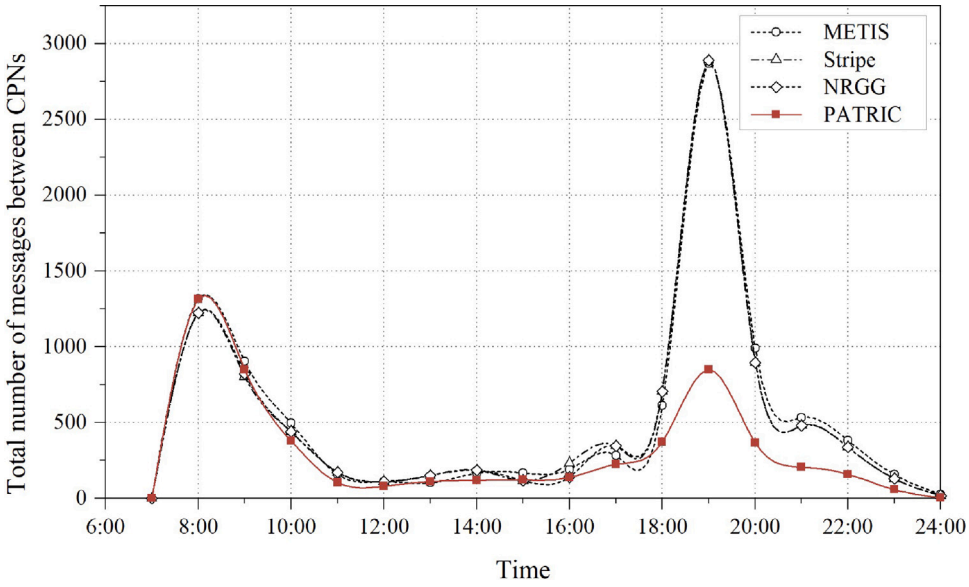


Fig. 11. Sum of synchronized messages between CPNs on 4 approaches during 17 hours.

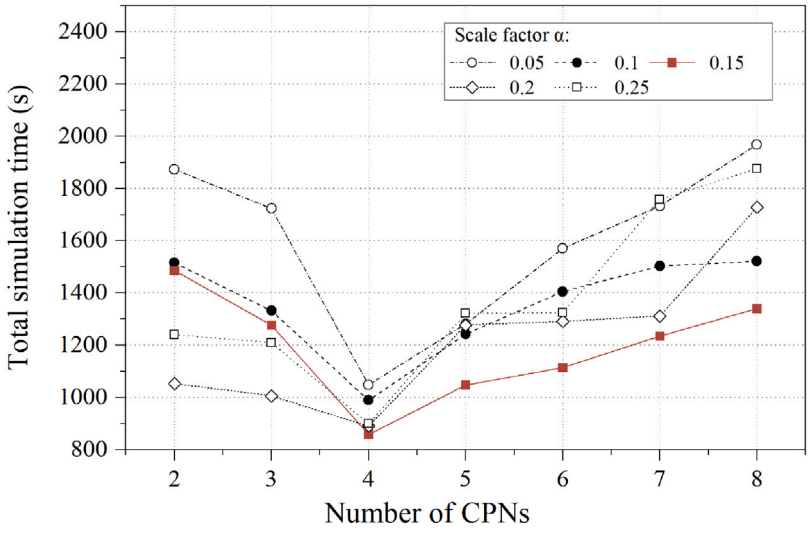


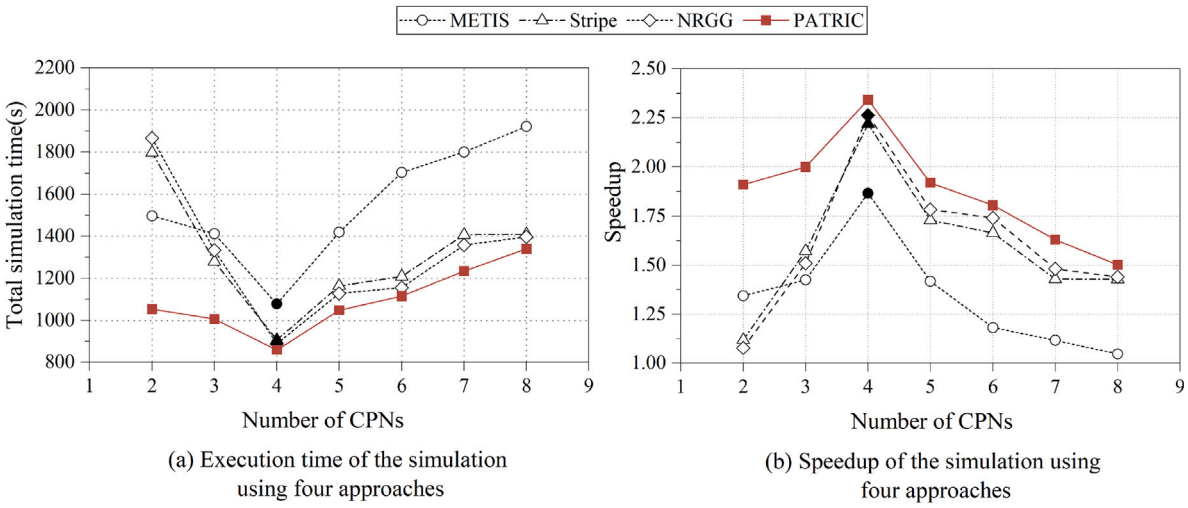
Fig. 12. The effect of the number of CPNs and the scale factor α on the total simulation time for PATRIC.

and METIS, which is benefit from the reduction of traffic flow on edge-cuts. Compared to other methods, PATRIC also achieves better load balance and reduces the workload imbalance degree by 20%, 21%, and 23%, respectively. Moreover, on the whole, PATRIC has the least overhead time. These results show that PATRIC outperforms state-of-the-art methods.

To prove that PATRIC can adapt to traffic changes and keep the number of synchronization messages between CPNs at a stable level, we plotted the number of messages during the simulation depends on the time of the day (Fig. 11). Expectedly, PATRIC reached the lowest number of messages exchanged during the simulation compared to other typical methods. Furthermore, our method was also relatively stable during the entire simulation period. Taking the Stripe method as an example, the difference between maximum and minimum exchanged messages of our method is 55% lower than Stripe. PATRIC succeeds in keeping the number of messages far below other methods during evening rush hours. As described in Fig. 11, the number of messages of our method is 65% less than METIS during evening rush hours, and 66% smaller than Stripe or NRGG.

5.2.4. Performance analysis

In Section 4.2.3, we introduce the scale factor α to determine whether to update the partitions or not. To optimize the acceleration performance of PATRIC, we adjust the scale factor α and the number of CPNs through a series of experiments. The average simulation time under various parameter settings is illustrated in Fig. 12. The simulation time of all algorithms decreases first and then increases



(a) Execution time of the simulation using four approaches

(b) Speedup of the simulation using four approaches

Fig. 13. Total simulation time and speedup of four approaches under different number of CPNs.

with the increase of the number of CPNs and α. PATRIC achieves the best performance when 4 CPNs are used for parallel simulation and α is set to 0.15.

Finally, to verify the simulation acceleration effect of PATRIC, we record and compare the simulation time of different accelerating approaches under 7 different CPN quantity conditions. The speedup which refers to the accelerating ratio of the simulation time using multiple CPNs to a single CPN is also calculated. Fig. 13(a) shows that no matter how many CPNs are used for the parallel simulation, PATRIC can complete the simulation the quickest. Without acceleration, it takes 2010 sec to complete the all-day traffic simulation of Pingshan. However, with the use of PATRIC, the simulation only requires 860 sec, achieving considerable improvement in simulation efficiency. Besides, all methods achieve their optimal results when using 4 CPNs for Pingshan traffic simulation, which may be related to the road structure and traffic distribution in Pingshan, and excessive CPNs will significantly increase the communication messages and load imbalance, thereby increasing the simulation time. Compared to other approaches, PATRIC can reduce simulation time by up to 21.4%, which sharply improves simulation efficiency. According to Fig. 13(b), in the case of 4 CPNs, PATRIC's speedup is 1.03 times faster compared to NRGG, 1.05 times faster than Strip, and 1.26 times faster than METIS. Moreover, the simulation time of PATRIC with only 2 CPNs is less than that of METIS with 4 CPNs, demonstrating the superiority of PATRIC in terms of simulation efficiency.

6. Conclusions and future work

Urban traffic is a complex system which varies a lot across time and space. During the morning and evening rush hours, the number of vehicles on the road network always surges in reality. The urban traffic flows concentrate within the business areas to form traffic clusters with numerous agents. To improve the traffic simulation performances, we propose a novel parallel framework named PATRIC based on data-driven traffic clustering.

The proposed approach can automatically partition road networks based on traffic clustering, and dynamically update partition to adapt to traffic changes, thus minimizing synchronization overhead and waiting time overhead among CPNs. Experiments on large-scale scenarios using a real-world road network show that PATRIC enables generating an optimal partitioning scheme for the parallel traffic simulation. The findings of this study confirm that existing traffic clusters in large cities can be used to improve the efficiency of the parallel agent-based urban simulation.

In the future, we will try to test our method in more cities to explore how the algorithm performs in different road structures and traffic distributions. We also plan to optimize other modules of parallel simulation such as message passing. Besides, further trials of PATRIC by combining other parallelization techniques, such as GPU computing technology is also to be performed.

CRediT authorship contribution statement

Lin Wan: Conceptualization, Methodology, Supervision, Writing – review & editing. **Ganmin Yin:** Software, Writing – original draft. Writing – review & editing. **Jiahao Wang:** Methodology, Software, Writing – original draft. **Golan Ben-Dor:** Writing – review & editing, Funding. **Aleksey Ogulenko:** Writing – review & editing, Funding. **Zhou Huang:** Supervision, Writing – review & editing, Funding.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared influence to the work reported in this paper.

Data availability

The authors do not have permission to share data.

Acknowledgements

This work is supported by grants from the National Key Research and Development Program of China (2017YFE0196100). The research of the Tel-Aviv team was funded by the Ministry of Science and Technology, Israel, Israel, and The Ministry of Science and Technology of the People's Republic of China, Grant No. 3-15741.

References

[1] I. Benenson, P.M. Torrens, Geosimulation: object-based modeling of urban phenomena, Comput. Environ. Urban Syst. 28 (2004) 1–8.
[2] D. Helbing, S. Balietti, How to do agent-based simulations in the future: From modeling social mechanisms to emergent phenomena and interactive systems design, Urban Econ. Reg. Stud. E J. (2011).
[3] H.R. Parry, M. Bithell, Large scale agent-based modelling: A review and guidelines for model scaling, Agent-Based Models Geogr. Syst. (2012) 271–308.
[4] L. Gasser, K. Kakugawa, B. Chee, M. Esteva, Smooth scaling ahead: Progressive MAS simulation from single PCs to grids, in: International Workshop on Multi-Agent Systems and Agent-Based Simulation, Springer, 2004, pp. 1–10.
[5] N. Cetin, A. Burri, K. Nagel, A large-scale agent-based traffic microsimulation based on queue model, in: Proceedings of Swiss Transport Research Conference, Strc, Monte Verita, Ch, Citeseer, 2003.
[6] P. Andelfinger, Y. Xu, D. Eckhoff, W. Cai, A. Knoll, Fidelity and performance of state fast-forwarding in microscopic traffic simulations, ACM Trans. Model. Comput. Simul. (TOMACS) 30 (2) (2020) 1–26.
[7] X. Gong, Z. Huang, Y. Wang, L. Wu, Y. Liu, High-performance spatiotemporal trajectory matching across heterogeneous data sources, Future Gener. Comput. Syst. 105 (2020) 148–161.
[8] Y. Bao, Z. Huang, X. Gong, Y. Zhang, G. Yin, H. Wang, Optimizing segmented trajectory data storage with hbase for improved spatio-temporal query efficiency, Int. J. Digit Earth. 16 (1) (2023) 1124–1143.
[9] J. Xiao, P. Andelfinger, D. Eckhoff, W. Cai, A. Knoll, A survey on agent-based simulation using hardware accelerators, ACM Comput. Surv. 51 (6) (2019) 1–35.
[10] Y. Xu, W. Cai, D. Eckhoff, S. Nair, A. Knoll, A graph partitioning algorithm for parallel agent-based road traffic simulation, in: Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, 2017, pp. 209–219.
[11] D. Yin, S. Wang, Y. Ouyang, ViCTS: a novel network partition algorithm for scalable agent-based modeling of mass evacuation, Comput. Environ. Urban Syst. 80 (2020) 101452.
[12] T. Potuzak, Distributed/parallel genetic algorithm for road traffic network division using a hybrid island model/step parallelization approach, in: 2016 IEEE/ACM 20th International Symposium on Distributed Simulation and Real Time Applications, DS-RT, IEEE, 2016, pp. 170–177.
[13] P. Sanders, C. Schulz, Distributed evolutionary graph partitioning, in: 2012 Proceedings of the Fourteenth Workshop on Algorithm Engineering and Experiments, ALENEX, SIAM, 2012, pp. 16–29.
[14] T. Potuzak, Distributed-parallel road traffic simulator for clusters of multi-core computers, in: 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications, IEEE, 2012, pp. 195–201.
[15] H. Chen, T. Cheng, X. Ye, Designing efficient and balanced police patrol districts on an urban street network, Int. J. Geogr. Inf. Sci. 33 (2) (2019) 269–290.
[16] Y. Leng, H. Wang, F. Lu, Artificial intelligence knowledge graph for dynamic networks: An incremental partition algorithm, IEEE Access 8 (2020) 63434–63442.
[17] C. Withanage, D. Lakmal, M. Hansini, K. Kankanamge, Y. Witharanage, U. Thayasivam, A modified multilevel k-way partitioning algorithm for trip-based road networks, in: MATEC Web of Conferences, Vol. 272, EDP Sciences, 2019, p. 1038.
[18] Q.-S. Hua, Y. Li, D. Yu, H. Jin, Quasi-streaming graph partitioning: A game theoretical approach, IEEE Trans. Parallel Distrib. Syst. 30 (7) (2019) 1643–1656.
[19] Y. Wang, Z. Huang, G. Yin, H. Li, L. Yang, Y. Su, Y. Liu, X. Shan, Applying Ollivier-Ricci curvature to indicate the mismatch of travel demand and supply in urban transit network, Int. J. Appl. Earth Obs. Geoinf. 106 (2022) 102666.
[20] G. Yin, Z. Huang, Y. Bao, H. Wang, L. Li, X. Ma, Y. Zhang, ConvGCN-RF: A hybrid learning model for commuting flow prediction considering geographical semantics and neighborhood effects, GeoInformatica (2022) 1–21.
[21] G. Yin, Z. Huang, L. Yang, E. Ben-Elia, L. Xu, B. Scheuer, Y. Liu, How to quantify the travel ratio of urban public transport at a high spatial resolution? a novel computational framework with geospatial big data, Int. J. Appl. Earth Obs. Geoinf. 118 (2023) 103245.
[22] Z. Yang, R. Zheng, Y. Ma, Parallel heuristics for balanced graph partitioning based on richness of implicit knowledge, IEEE Access 7 (2019) 96444–96454.
[23] S. Gong, Y. Zhang, G. Yu, HBP: Hotness balanced partition for prioritized iterative graph computations, in: 2020 IEEE 36th International Conference on Data Engineering, ICDE, IEEE, 2020, pp. 1942–1945.
[24] K. Ramamohanarao, H. Xie, L. Kulik, S. Karunasekera, E. Tanin, R. Zhang, E.B. Khunayn, Smarts: Scalable microscopic adaptive road traffic simulator, ACM Trans. Intell. Syst. Technol. 8 (2) (2016) 1–22.
[25] Y. Qu, X. Zhou, Large-scale dynamic transportation network simulation: A space-time-event parallel computing approach, Transp. Res. C 75 (2017) 1–16.
[26] H. Wang, Z. Huang, X. Zhou, G. Yin, Y. Bao, Y. Zhang, Doufu: a double fusion joint learning method for driving trajectory representation, Knowl.-Based Syst. 258 (2022) 110035.
[27] H.B. Celikoglu, M. Dell'Orco, Mesoscopic simulation of a dynamic link loading process, Transp. Res. C 15 (5) (2007) 329–344.
[28] B. Chen, H.H. Cheng, J. Palen, Integrating mobile agent technology with multi-agent systems for distributed traffic detection and management systems, Transp. Res. C 17 (1) (2009) 1–10.
[29] M. Dell'Orco, M. Marinelli, M.A. Silgu, Bee colony optimization for innovative travel time estimation, based on a mesoscopic traffic assignment model, Transp. Res. C 66 (2016) 48–60.
[30] M. Di Gangi, G.E. Cantarella, R. Di Pace, S. Memoli, Network traffic control based on a mesoscopic dynamic flow model, Transp. Res. C 66 (2016) 3–26.
[31] Ł. Kamiński Bogumił and Kraiński, A. Mashatan, P. Paweł, P. Szufel, Multiagent routing simulation with partial smart vehicles penetration, J. Adv. Transp. 2020 (2020).

[32] J. Liu, P. Mirchandani, X. Zhou, Integrated vehicle assignment and routing for system-optimal shared mobility planning with endogenous road congestion, Transp. Res. C 117 (2020) 102675.
[33] P.A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, E. Wießner, Microscopic traffic simulation using sumo, in: 2018 21st International Conference on Intelligent Transportation Systems, ITSC, IEEE, 2018, pp. 2575–2582.
[34] A. Ehlert, A. Schneck, N. Chanchareon, Junction parameter calibration for mesoscopic simulation in Vissim, Transp. Res. Procedia 21 (2017) 216–226.
[35] M. Certickỳ, M. Jakob, R. Píbil, Z. Moler, Agent-based simulation testbed for on-demand mobility services, Procedia Comput. Sci. 32 (2014).
[36] J. Auld, M. Hope, H. Ley, V. Sokolov, B. Xu, K. Zhang, POLARIS: Agent-based modeling framework development and implementation for integrated travel demand and network and operations simulations, Transp. Res. C 64 (2016) 101–116.
[37] A. Horni, K. Nagel, K. Axhausen, The multi-agent transport simulation title of book: The multi-agent transport simulation MATSim subtitle positioned below, 2016.
[38] H.W.Y. Adoni, T. Nahhal, M. Krichen, B. Aghezzaf, A. Elbyed, A survey of current challenges in partitioning and processing of graph-structured data in parallel and distributed systems, Distrib. Parallel Databases (2019) 1–36.
[39] G.B. Dor, E. Ben-Elia, I. Benenson, Population downscaling in multi-agent transportation simulations: A review and case study, Simul. Model. Pract. Theory (2020) 102233.
[40] S. Abar, G.K. Theodoropoulos, P. Lemarinier, G.M. O'Hare, Agent based modelling and simulation tools: A review of the state-of-art software, Comp. Sci. Rev. 24 (2017) 13–33.
[41] W. Gao, M. Balmer, E.J. Miller, Comparison of MATSim and EMME/2 on greater toronto and hamilton area network, Canada, Transp. Res. Rec. 2197 (1) (2010) 118–128.
[42] M.A. Ji, P.E.I. Yu-long, Development and application of TransCAD for urban traffic planning, J. Harb. Univ. Civ. Eng. Archit. 5 (2002).
[43] T. Vorraa, Transport modelling supported by GIS–an overview of GIS features now within cube, Urban Transp. XV Urban Transp. Environ. 15 (2009) 235.
[44] D. Graur, R. Bruno, J. Bischoff, M. Rieser, W. Scherr, T. Hoefler, G. Alonso, Hermes: Enabling efficient large-scale simulation in MATSim, Procedia Comput. Sci. 184 (2021) 635–641.
[45] T. Arentze, F. Hofman, H. van Mourik, H. Timmermans, ALBATROSS: multiagent, rule-based model of activity pattern decisions, Transp. Res. Rec. 1706 (1) (2000) 136–144.
[46] T. Bellemans, B. Kochan, D. Janssens, G. Wets, T. Arentze, H. Timmermans, Implementation framework and development trajectory of FEATHERS activity-based simulation platform, Transp. Res. Rec. 2175 (1) (2010) 111–119.
[47] I.N. Sener, C.R. Bhat, R. Copperman, S. Srinivasan, J.Y. Guo, A. Pinjari, N. Eluru, Activity-Based Travel-Demand Analysis for Metropolitan Areas in Texas: CEMDAP Models, Framework, Software Architecture and Application Results, Tech. Rep., Midwest Regional University Transportation Center, 2006.
[48] C. Dobler, Implementation of a time step based parallel queue simulation in MATSim, in: 10th Swiss Transport Research Conference, Monte Verita, Ascona, 2010.
[49] A. Saprykin, N. Chokani, R.S. Abhari, GEMSim: A GPU-accelerated multi-modal mobility simulator for large-scale scenarios, Simul. Model. Pract. Theory 94 (2019) 199–214.
[50] M.J. Quinn, Parallel Computing Theory and Practice, McGraw-Hill, Inc., 1994.
[51] Q. Bragard, A. Ventresque, L. Murphy, Self-balancing decentralized distributed platform for urban traffic simulation, IEEE Trans. Intell. Transp. Syst. 18 (5) (2016) 1190–1197.
[52] C. Chan, B. Wang, J. Bachan, J. Macfarlane, Mobiliti: scalable transportation simulation using high-performance parallel computing, in: 2018 21st International Conference on Intelligent Transportation Systems, ITSC, IEEE, 2018, pp. 634–641.
[53] G. Cordasco, R. De Chiara, A. Mancuso, D. Mazzeo, V. Scarano, C. Spagnuolo, Bringing together efficiency and effectiveness in distributed simulations: the experience with D-MASON, Simulation 89 (10) (2013) 1236–1253.
[54] Z. Liu, Q. Meng, Distributed computing approaches for large-scale probit-based stochastic user equilibrium problems, J. Adv. Transp. 47 (6) (2013) 553–571.
[55] D. Fu, M. O'Connor, M. Becker, H. Szczerbicka, Approximate distributed discrete event simulation using semi-conservative look-ahead estimation, in: 2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications, DS-RT, IEEE, 2019, pp. 1–8.
[56] A. Pellegrini, F. Quaglia, Cross-state events: A new approach to parallel discrete event simulation and its speculative runtime support, J. Parallel Distrib. Comput. 132 (2019) 48–68.
[57] Y. Xu, W. Cai, H. Aydt, M. Lees, D. Zehe, Relaxing synchronization in parallel agent-based road traffic simulation, ACM Trans. Model. Comput. Simul. (TOMACS) 27 (2) (2017) 1–24.
[58] R. Klefstad, Y. Zhang, M. Lai, R. Jayakrishnan, R. Lavanya, A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation, in: Proceedings. 2005 IEEE Intelligent Transportation Systems, 2005, IEEE, 2005, pp. 813–818.
[59] D.-H. Lee, P. Chandrasekar, A framework for parallel traffic simulation using multiple instancing of a simulation program, ITS J. 7 (3–4) (2002) 279–294.
[60] K. Nagel, M. Rickert, Parallel implementation of the TRANSIMS micro-simulation, Parallel Comput. 27 (12) (2001) 1611–1639.
[61] N.M. Soudani, A. Fatemi, M. Nematbakhsh, An investigation of big graph partitioning methods for distribution of graphs in vertex-centric systems, Distrib. Parallel Databases 38 (1) (2020) 1–29.
[62] G. Gomes, J. Ugirumurera, X. Li, Distributed macroscopic traffic simulation with open traffic models, in: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems, ITSC, 2020, pp. 1–6.
[63] P. Sanders, C. Schulz, Distributed evolutionary graph partitioning, in: Proceedings of the Fourteenth Workshop on Algorithm Engineering and Experiments, ALENEX, SIAM, 2012, pp. 16–29.
[64] I. Stanton, Streaming balanced graph partitioning algorithms for random graphs, in: Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2014, pp. 1287–1301.
[65] X. Zhu, W. Chen, W. Zheng, X. Ma, Gemini: A computation-centric distributed graph processing system, in: 12th {USENIX} Symposium on Operating Systems Design and Implementation, {OSDI} 16, 2016, pp. 301–316.
[66] N. Xu, L. Chen, B. Cui, LogGP: a log-based dynamic graph partitioning method, Proc. VLDB Endow. 7 (14) (2014) 1917–1928.
[67] G. Echbarthi, H. Kheddouci, Fractional greedy and partial restreaming partitioning: New methods for massive graph partitioning, in: 2014 IEEE International Conference on Big Data, Big Data, IEEE, 2014, pp. 25–32.
[68] M.A.K. Patwary, S. Garg, B. Kang, Window-based streaming graph partitioning algorithm, in: Proceedings of the Australasian Computer Science Week Multiconference, 2019, pp. 1–10.
[69] F. Rahimian, A.H. Payberah, S. Girdzijauskas, M. Jelasity, S. Haridi, Ja-be-ja: A distributed algorithm for balanced graph partitioning, in: 2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems, IEEE, 2013, pp. 51–60.
[70] T. Chen, B. Li, A distributed graph partitioning algorithm for processing large graphs, in: 2016 IEEE Symposium on Service-Oriented System Engineering, SOSE, IEEE, 2016, pp. 53–59.
[71] G. Karypis, V. Kumar, Multilevelk-way partitioning scheme for irregular graphs, J. Parallel Distrib. Comput. 48 (1) (1998) 96–129.
[72] A. Attanasi, E. Silvestri, P. Meschini, G. Gentile, Real world applications using parallel computing techniques in dynamic traffic assignment and shortest path search, in: 2015 IEEE 18th International Conference on Intelligent Transportation Systems, IEEE, 2015, pp. 316–321.
[73] S. Chokri, S. Baroud, S. Belhaous, M. Bentaleb, M. Mestari, M. El Youssfi, Heuristics for dynamic load balancing in parallel computing, in: 2018 4th International Conference on Optimization and Applications, ICOA, IEEE, 2018, pp. 1–5.

[74] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, P. Kalnis, Mizan: a system for dynamic load balancing in large-scale graph processing, in: Proceedings of the 8th ACM European Conference on Computer Systems, 2013, pp. 169–182.
[75] A. Leivadeas, G. Kesidis, M. Falkner, I. Lambadaris, A graph partitioning game theoretical approach for the VNF service chaining problem, IEEE Trans. Netw. Serv. Manag. 14 (4) (2017) 890–903.
[76] A. McCrabb, E. Winsor, V. Bertacco, Dredge: Dynamic repartitioning during dynamic graph execution, in: 2019 56th ACM/IEEE Design Automation Conference, DAC, IEEE, 2019, pp. 1–6.
[77] M. Onizuka, T. Fujimori, H. Shiokawa, Graph partitioning for distributed graph processing, Data Sci. Eng. 2 (1) (2017) 94–105.
[78] S. Salihoglu, J. Widom, Gps: A graph processing system, in: Proceedings of the 25th International Conference on Scientific and Statistical Database Management, 2013, pp. 1–12.
[79] L. Vaquero, F. Cuadrado, D. Logothetis, C. Martella, xDGP: A dynamic graph processing system with adaptive partitioning, 2013, arXiv preprint arXiv:1309.1049.
[80] N.T. Bao, T. Suzumura, Towards highly scalable pregel-based graph processing platform with x10, in: Proceedings of the 22nd International Conference on World Wide Web, 2013, pp. 501–508.
[81] Z. Shang, J.X. Yu, Catch the wind: Graph workload balancing on cloud, in: 2013 IEEE 29th International Conference on Data Engineering, ICDE, IEEE, 2013, pp. 553–564.
[82] Y. Bai, J. Yuan, S. Liu, K. Yin, Variational community partition with novel network structure centrality prior, Appl. Math. Model. 75 (2019) 333–348.
[83] D. Wei, F. Chen, X. Sun, An improved road network partition algorithm for parallel microscopic traffic simulation, in: 2010 International Conference on Mechanic Automation and Control Engineering, IEEE, 2010, pp. 2777–2782.
[84] A. Shafi, B. Carpenter, M. Baker, Nested parallelism for multi-core HPC systems using Java, J. Parallel Distrib. Comput. 69 (6) (2009) 532–545.
[85] X. Zhou, H. Wang, Z. Huang, Y. Bao, G. Zhou, Y. Liu, Identifying spatiotemporal characteristics and driving factors for road traffic CO2 emissions, Sci. Total Environ. 834 (2022) 155270.