Can Road Traffic Volume Information Improve Partitioning for Distributed SUMO?

Ulrich Dangel, Quentin Bragard, Patrick McDonagh, Anthony Ventresque and Liam Murphy

Abstract Microscopic vehicular simulations can be computationally intensive due to the sheer size of the road network and number of vehicles. One solution is to parallelize the simulation through distribution and concurrent execution of the scenario being simulated. To enable distributed simulation of an area, the partitioning of the map into different areas for parallel execution on different nodes is required. How the map is partitioned is also a critical factor for distributed simulation, as a poor partitioning can lead to a communication overhead and/or an imbalance of workload among the computing nodes. In this paper, we ask: Can traffic volume information improve the classical structural partitioning algorithms? In the context of improving distributed simulation in SUMO, we propose a modification to three existing mechanisms for road network partitioning, SParTSim, Smart Quadtrees and Quadtrees, with the aim of creating more balanced partitions (in terms of workload) derived from traffic volume data.

Keywords Distributed simulation · Road partitioning · Graph partitioning · SUMO

U. Dangel · Q. Bragard (✉) · A. Ventresque · L. Murphy
Lero@UCD, School of Computer Science and Informatics,
University College Dublin, Dublin, Ireland
e-mail: quentin.bragard@ucdconnect.ie

U. Dangel
e-mail: ulrich.dangel@ucdconnect.ie

A. Ventresque
e-mail: anthony.ventresque@ucd.ie

L. Murphy
e-mail: liam.murphy@ucd.ie

P. McDonagh
Lero@DCU, School of Electronic Engineering, Dublin City University, Dublin, Ireland
e-mail: patrick.mcdonagh@dcu.ie

M. Behrisch and M. Weber (eds.), *Modeling Mobility with Open Data*,
Lecture Notes in Mobility, DOI 10.1007/978-3-319-15024-6_5

1 Introduction

Urban populations are growing dramatically: for instance, the aggregated annual population increase of six major developing-country cities is already higher than Europe's total population [1]. With the increase in the size of cities, traffic simulation requires more computation time in order to simulate more individual vehicles. This is particularly the case for microscopic traffic simulation, which can offer interesting insights to its users, but has a high computation time. Microscopic traffic simulation can accurately model urban traffic patterns and evaluate different scenarios and their impact on traffic, e.g. placement of additional bus stops at a route, traffic light sequencing, etc. By using microscopic simulation, stakeholders can directly observe the impact of their potential decisions on the traffic. As stated above, microscopic simulation models are generally slow, as they need to process a large number of elements (e.g., individual cars). A standard solution to reduce the overall required computation time is to parallelize and distribute the simulation.

Classically, parallel or distributed systems split the problem space into different partitions, i.e., sub problems for concurrent execution, this may involve synchronisation between nodes if data from one partition needs to be moved to another partition. For vehicular simulation, these partitions are typically based on the road network or the spatial map—we call this style of partitioning, structural. The partitioning algorithms are evaluated using two main metrics [2]: (i) the balancing of computational workload across the nodes that run the partitions; (ii) the communication overhead generated by the distribution.

Distributed simulations are currently an active area of research interest within the SUMO community. There has been recent work to provide a multi-agent system on top of SUMO [3] by combining it with an existing multi-agent development framework [4]. Another approach for distributed SUMO simulation is dSUMO [5], a framework that interconnects SUMO instances, each running separate, but spatially connected areas of a map. Both solutions require mechanisms to divide the road-network into different areas for parallel processing on their respective nodes.

In this paper, we propose an enhancement for distributed simulation using SUMO by using traffic volume data to improve the load balancing of the individual partitions and minimizing the communication overhead, in order to reduce the overall required computation time of the distributed simulation. We evaluate this idea by comparing results against those obtained for SParTSim [6], Quadtrees [7] and Smart Quadtrees [8].

2 Related Work

Partitioning in general is a key concept in distributed and parallel computing. In MapReduce [9] the mapping is a partitioning which is responsible for distributing the input data to different processes. This partitioning step enables the distributed and parallel execution of the work.

Other, more domain-specific partitioning schemes, provide guidance how to select and choose appropriate partitioning algorithms.

Space partitioning, for example, is often used in computer graphics [10–12] and visualisation. An overview about different space partitioning algorithms was provided by the authors in [13]. Here the authors discussed Quadtrees, unconstrained k-d trees, constrained k-d trees and region growing with region growing performing best for their simulation. Space partitioning is widely used in distributed or parallel computation, such as Massively Multiplayer Online Role-Playing Games (MMORPG) or Raytracing [14]. Employing a binary space partitioning mechanism, such as Quadtrees, will lead to the creation of a spatial hierarchy. This hierarchy can be used to divide a city, and assign pieces of it (partitions) to different nodes. Another approach for the space partitioning of cities is to reuse existing boundaries such as postal districts. The problem with both approaches is that they typically do not use the road-network for the partitioning but only spatial information. With regards to a distributed vehicular simulation, this can lead to uneven distribution of workload. This in turn, will lead to decreased simulation performance as a result of uneven processing times for simulation steps, resulting in some nodes waiting for others when synchronisation is required.

Graph-partitioning on the other hand, does not consider the space but uses the graph-structure of the problem. Graph partitioning has been used to parallelize clustering of documents [15], parallel factorisation of sparse matrices [16] as well as workload distribution [17]. Graph partitioning has been originally implemented with heuristics [18] and was later extended to utilize genetic-algorithms [19]. Graph-growing, is a refinement and extension [2] of classical graph partitioning and expands individual partitions in each step. Region growing, similar to graph-growing, has been shown to be best solution for crowd simulations [13]. Graph partitioning is widely used [20, 21] in different domains such as workload distribution, task scheduling and in the VLSI [22] domain. Using graph partitioning for vehicular simulations solves multiple issues encountered with space partitioning, such as uneven distribution of roads in a partition as graph partitioning works on the street level and not on the map. Taking road properties into account can further refine graph partitioning, i.e. edges provide attributes about the significance of a particular street. By using additional attributes of street-data, a graph partitioning targeted for road networks can be derived, such as SParTSim.

3 Experimental Evaluation

In order to use input data for the different partitioning algorithms, we have to extract volume data to provide the partitioning. In real-world scenarios, such data can be extracted from existing Traffic Management Systems, such as SCATS [23] or

IRIS.[1] In this work, we use the dataset provided by TAPAS Cologne [24] with SUMO to extract the volume data. Below, we describe the formula used for providing a weight for nodes in the road graph based on traffic data, as well as modifications to the existing algorithms.

3.1 Volume Extraction

As some of the algorithms used are graph based, we provide a weight per node instead of a weight per edge. This allows us to use the same weighting for all algorithms, whether they are graph or space-based. We use a weighted sum as shown in (1), to calculate the weight of a node, N_w, with c_t being the total number of cars present at step t, c_{tn} the number of cars at node n at step t.

$$N_w = \sum w_t \frac{c_{tn}}{c_t}. \tag{1}$$

where the weight w_t is defined as the number of cars in this step over the maximum number of observed cars (in any one step), as shown in (2).

$$w_t = \frac{c_{tn}}{c_{max}}. \tag{2}$$

By using (2) we ensure that steps with a low traffic volume have a lower impact on the overall weight of a node.

3.2 Modification of Quadtrees

Quadtrees are a space-dividing partitioning method, often used to divide two-dimensional spaces. Quadtrees divide a space recursively into sub-regions, until a specific stop condition is met, e.g., the space is divided evenly or, into the required number of partitions.

The original version of the used Quadtree algorithm uses the sum of the street size (street length * number of lanes) to select the partition to divide. We modify Quadtrees to not use the sum of the street size but the sum of the volume data from Sect. 3.1 above, in order to select the partition to divide further. By using the sum of the volume data for each partition, we choose the area with the highest weight to divide further.

[1] http://iris.dot.state.mn.us/.

3.3 Modification of Smart Quadtrees

Smart Quadtrees, also referred as grid based partitioning, are an extension to Quadtrees where the map is initially divided into small, independent grids. These grids are then merged together according to some heuristic, based on the value of an individual region. This differs to Quadtrees as Quadtree divides a map into 4 similar regions, while Smart Quadtrees divides a map into small grids and merges them until all grids are merged (Fig. 1).

The unmodified version of the Smart Quadtree implementation uses the street size (street lengths * number of lanes) as a heuristic. We modify Smart Quadtrees by changing the heuristic to use the sum of the volume data, as described in Sect. 3.1, for each grid. The difference between the two implementations is shown in Fig. 2.

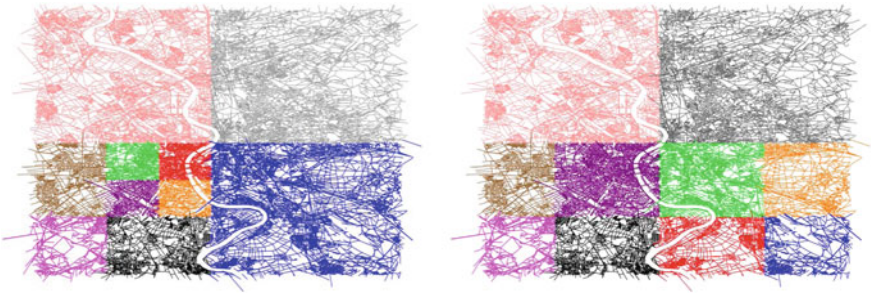

Fig. 1 Output of the modified quadtree algorithm (*left*) and unmodified quadtree (*right*) with ten partitions or three divisions

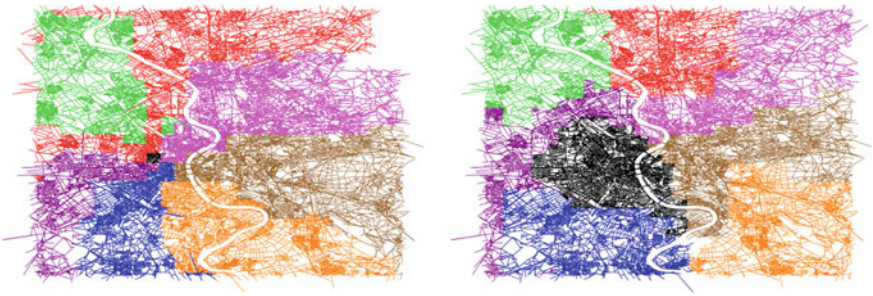

Fig. 2 Output of the modified smart quadtree algorithm (*left*) and unmodified smart quadtree (*right*) with eight partitions

3.4 Modification of SParTSim

SParTSim uses the concept of creating a domain-specific partitioning algorithm for road networks by combining space partitioning (region-growing) with graph partitioning. By utilizing both space-, and graph-partitioning methodologies, SParTSim aims to produce better partitions for vehicular distributed simulations. SParTSim determines the starting point of each partition by choosing the node with the highest degree. After the starting point for the individual partitions is selected, each partition grows, starting from the starting point. SParTSim grows the partitions based on road-network attributes, such as number of lanes.

As unmodified version of SParTSim determines the starting point of a partition by choosing the nodes with the highest degrees, the starting point of a partition impacts the shape of an individual partition as the partition starts to grow around this point until it can't grow any more as a result of all areas now belonging to other partitions, i.e. all areas on the map are covered. SParTSim then trades road segments between partitions to minimize the road cuts between partitions and to achieve load-balanced partitions. The SParTSim algorithm only uses static graph properties to achieve evenness of road topology between partitions. In order to do this it uses a heuristic to determine the workload for the individual partitions. However, SParTSim considers that if the road topology is balanced between partitions, then the workload will be similar, irrespective of the actual traffic volume. Therefore, we modified the starting point selection in SParTSim to use the nodes with the highest traffic volume (as determined in Sect. 3.1) instead of using the nodes with the highest degree. Figure 3 shows the partitioning result of both the unmodified and modified version of SParTSim.

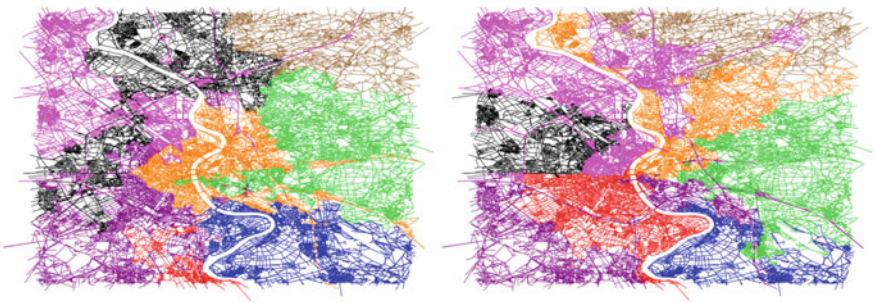

Fig. 3 Output of the modified SParTSim algorithm (*left*) and unmodified SParTSim (*right*) with eight partitions

3.5 Simulation of a Distributed Simulation

In this paper we are only interested in the impact of urban data on the quality of the partitioning, not on the distributed simulation itself. For more details on the latter, we refer the reader to our previous work on a distributed version of SUMO [5]. We have then decided to focus on the partitioning and only "simulate" the distributed simulation: TAPAS Cologne gives us the position of every vehicle at any given time, so that we are able to know precisely when a vehicle crosses the border of a partition for every partitioning schemes; this allows us to run SUMO for each partition and to manage the passing of vehicles in an ad hoc manner, without using the communication and synchronisation mechanisms of a real distributed simulation. It is important to tell again that while we avoid here all the characteristics of a distributed simulation (e.g., communication time and synchronisation mechanism), this does not have any impact on the focus of our study, i.e., an evaluation of the improvement of using urban traffic data for the partitioning in a distributed simulation.

4 Evaluation

In this section, we evaluate the use of traffic volume data for Quadtrees, Smart Quadtrees and SParTSim. We divide the city for both Smart Quadtrees and SParTSim into four and eight partitions while we use for Quadtrees four and ten partitions. The visual partitioning outputs for Quadtrees with 10 partitions is shown in Fig. 1, with the outputs for Smart Quadtrees with eight partitions is shown in Fig. 2 and those for SParTSim are shown in Fig. 3.

4.1 Metrics

In the first part of our evaluation we focus on two metrics, communication overhead and workload balance. For communication overhead, we calculate the number of messages sent between partitions in each step. These messages represent the movement of a vehicle on a road segment, which is divided across partitions. We can calculate this with SUMO by extracting the position of each vehicle in each step. If a street intersects or touches a partition border, it is part of multiple partitions. This ensures that states are shared between different nodes. If a vehicle is on a road segment, which is divided across partitions, a message has to be sent to the neighbouring partition to transfer the state of the vehicle across to the new partition. As each message has to be communicated and processed by dSUMO, the lower the number of messages, the better. The results for this metric are provided below.

To evaluate the workload balance between partitions, we calculate the Simpson Diversity Index [25], as shown in (3), with C_p being the cars in partition p, c_t the total number of cars in step t and P the number of partitions. The result is between 0 and 1 with 1 being a perfectly load balanced system and 0 being the opposite for unbalanced workloads between partitions.

$$D_t = \frac{1}{\sum_{p=0} (C_p/C_t)^2 P}. \tag{3}$$

In the second part of our evaluation we simulate the use of a distributed simulation to measure the time required to process each simulation and the real time factor. In a conservative distributed simulation, the slowest node determines the time required to run the whole simulation. The more the simulation is load-balanced, the smaller the difference will be between the slowest and the fastest node. The second metric used is the average time required by a mock distributed simulation to be executed. The third metric is the number of Vehicle Per Second (VPS) which is measured by adding the number of vehicles processed at each step divided by the runtime. The last metric is the real time factor correspond to how much faster the simulation is compared to the real time. For instance, a real time factor of 10 means that the simulation execute 10 s of simulated time in 1 s of real time.

5 Results

We use the TAPAS Cologne [24] 0.17 scenario to evaluate our result. TAPAS Cologne is a simulation describing the traffic of Cologne on a workday between 06:00 and 08:00 am. The data was captured as part of the TAPAS project [26] and has been refined multiple times. The scenario consists of 7,200 steps, with one step representing one second in real-time. TAPAS Cologne contains more than 250,000 vehicles traces for the 2 h period.

Figures 4 and 5 display the number of messages between partitions per simulation step for Quadtrees, Smart Quadtree and SParTSim. We don't distinguish between the modified and unmodified Quadtree for four partitions, as both results are exactly the same.

Due to the regular, rectangular shape of the partitions the Quadtree shows the best communication properties. In all cases, though, the modified versions of the algorithms show increased levels of communication, compared to the unmodified versions. The modified Quadtree algorithm selects the city centre (Fig. 1) for further partitioning, resulting in additional communication overhead. For the Smart Quadtree algorithm, our modified version created some small partitions (Fig. 2), causing additional communication overhead. Our modified algorithms show higher communication overhead compared to the unmodified versions. This is expected as our modifications focus on load balanced partitions and does not optimize with regard to communication. However, as can be seen below (in terms of workload

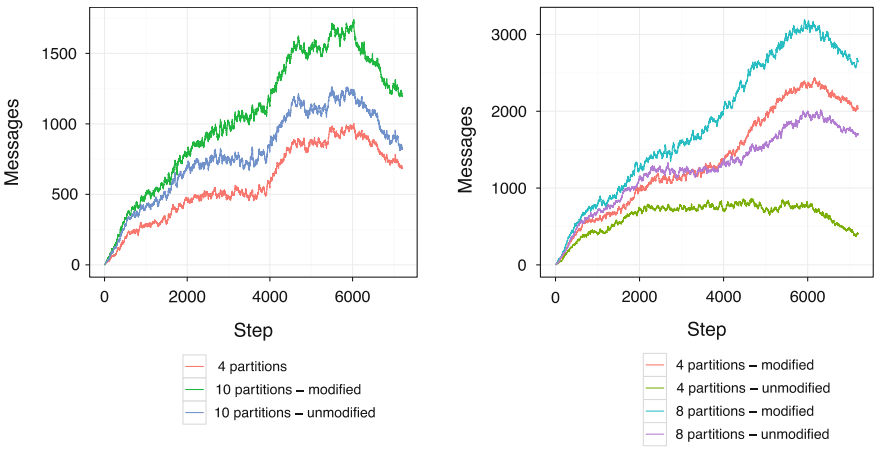

Fig. 4 Number of messages sent per simulation step for quadtrees (*left*) and smart quadtrees (*right*). Modified and unmodified versions are both shown

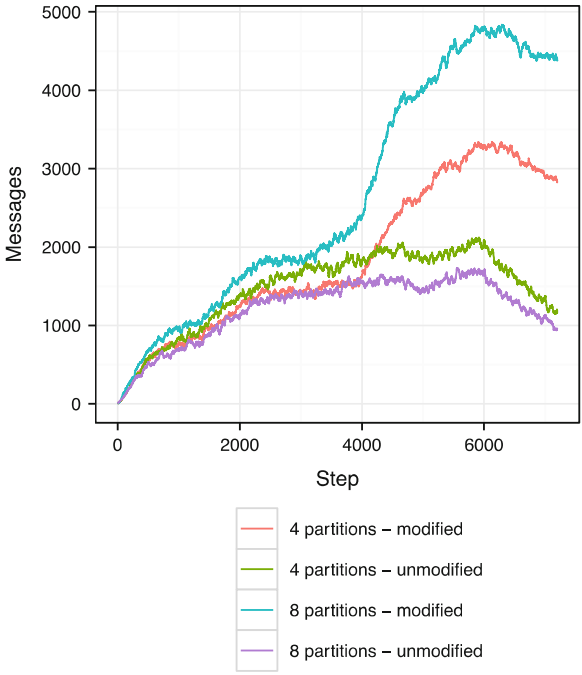

Fig. 5 Number of messages sent per simulation step for SParTSim, both modified and unmodified for 4 and 8 partitions

balance) our algorithms achieve a higher level of balancing between partitions, which should provide higher utilisation across all compute nodes as delays incurred by waiting for simulation should be decreased.

SParTSim has a trading phase, which aims to reduce the communication overhead. This behaviour can be observed in Fig. 5 for the unmodified versions, which perform better than the Smart Quadtree. Our modified initial starting point selection for SParTSim caused the increased communication overhead. This shows, that even though SParTSim has a trading mechanism to reduce the communication overhead, the initial point selection has a large impact on the resulting partition.

For the case of workload balance between partitions, Table 1 shows the properties of the Simpson diversity index (the higher the number, the better) over the complete simulation for all 3 algorithms. For both Quadtree and Smart Quadtree our modification provides better load-balanced partitions compared to the unmodified versions of the same algorithm, e.g. for the Smart Quadtree our modifications are twice as good as the unmodified versions. Our modifications to SParTSim on the other hand, provide slightly worse results compared to the unmodified algorithms. This is due to the trading phase of SParTSim, as we did not adjust the trading phase but only the initial starting point selection.

Comparing the different algorithms to each other shows that our modified Smart Quadtree produces more even partitions than the other partitioning algorithm. Our modified version of the Quadtree took 1 h 13 min to compute 10 partitions, Smart Quadtree took 1 h 22 min to compute eight partitions while SParTSim took 5 h 14 min for eight partitions on a 4 Core I 7-2,600 with 16 GB of memory. As shown in [27, 28] load balanced simulations are a required to optimize the overall computation time.

The modified Smart Quadtree provides more balanced partitions compared to the other algorithms. Furthermore, the modification to Quadtree and Smart Quadtree provide more balanced partitions compared to unmodified versions of their algorithm. In addition to providing more balanced partitions, we can observe that for Smart Quadtree, the time taken to compute these partitions is significantly lower, compared to SParTSim. In the case of Quadtree, the time taken to compute the partitions is significantly lower than SParTSim (and Smart Quadtree) but at the

Table 1 Simpson diversity index for the different partitioning algorithms over the simulation

Name		Min	Median	Mean	Max
Quadtree	4 partitions	0.3680	0.7190	0.7318	0.8540
	10 partitions—modified	0.3570	0.6070	0.6021	0.6330
	10—unmodified	0.2270	0.3530	0.3674	0.5540
Smart quadtree	4 partitions—modified	0.5610	0.9000	0.9157	0.9940
	4 partitions—unmodified	0.358	0.431	0.427	0.568
	8 partitions—modified	0.4460	0.7760	0.7798	0.8550
	8 partitions—unmodified	0.2840	0.3890	0.3889	0.4940
SParTSim	4 partitions—modified	0.4810	0.6650	0.6718	0.7340
	4 partitions—unmodified	0.7210	0.7920	0.7854	0.8450
	8 partitions—modified	0.4060	0.4540	0.4642	0.6430
	8 partitions—unmodified	0.4710	0.6850	0.6573	0.7780

expense of workload balance. Our modification to SParTSim on the other hand, did not provide better results, due to the unmodified trading phase. We expect that by modifying the trading phase, the result for SParTSim will improve as well.

While Table 1 shows the theoretical benefits of using urban data for the partitioning in terms of load balancing (i.e., Simpson Index), Table 2 presents some experimental results from the simulation of the distributed simulation. The results for Quadtree show that while the average time and the VPS stay stable for both configurations, the runtime is divided by more than 2 for the modified methods, using urban data, and the Real time factor is proportionally twice higher. Regarding Smart QuadTree, the modified versions for 4 and 8 partitions get a runtime improvement of respectively 13 and 22 %. While the average time and the VPS just slightly vary, Real time factor is improved in the same way than runtime. As we previously observed for the load balancing with the Simpson index, using urban data to optimise SParTSim does not seem to improve the simulation time.

When we increase the number of partitions to 16, the results seem to show a limited improvement. Despite increasing the number of partitions, QuadTree presents no improvement between 10 and 16 partitions for the modified version. Again the modified SParTSim shows results a little bit worse than the original SParTSim but the improvement due to the increase of the number of partitions goes from 6 to 15 %. On the other side, while the original Smart QuadTree achieves good results, the modified version shows a slower runtime. This decrease in performance comes from a bad choice of seeds. It appears that some seeds generated regions inside

Table 2 Execution time for every SUMO instances in the simulated version of a distributed SUMO

Name		Time(s)	Avg time(s)	VPS	Real time factor
Quadtree	4 partitions	247.46	107.75	78830.22	30.31
	10 partitions—modified	104.78	47.388	71834.39	71.58
	10—unmodified	223.52	44.12	75072.70	33.55
	16 partitions—modified	103.1	30.08	35056.79	72.75
	16—unmodified	198.11	27.86	83255.98	37.86
Smart quadtree	4 partitions—modified	115.78	97.13	112691.76	64.78
	4 partitions—unmodified	133.33	92.14	108292.90	56.26
	8 partitions—modified	93.62	51.44	59541.65	80.12
	8 partitions—unmodified	120.61	45.34	72850.18	34.80
	16 partitions—modified	197.79	27.87	83389.63	37.92
	16—unmodified	87.08	23.79	107091.92	86.13
SParTSim	4 partitions—modified	179.76	84.60	115305.59	41.72
	4 partitions—unmodified	176.34	82.42	110353.29	42.53
	8 partitions—modified	100.34	44.43	92839.74	74.75
	8 partitions—unmodified	88.29	42.71	97958.16	84.95
	16 partitions—modified	94.09	22.44	86193.98	79.72
	16—unmodified	75.45	23.84	86593.48	99.41

others and could not fully grow to form proper regions. As Smart QuadTree does not have any trading phase, it creates in this case tiny regions inside others, processing only few vehicles and spending most of their time communicating vehicles with other partitions.

These results lead to two observations. Firstly, we can see that the runtime and the Simpson Index match in most cases. The higher the Simpson Index is, the smaller the simulation time will be. This observation reinforce us in thinking that the Simpson Index is a good indicator of the load balanced state of a distributed simulation. Moreover, as expected, the average time and the number of vehicles per second prove to give no indication on the load balancing. Secondly, it looks like there is a direct correlation between the smartness of an algorithm and the improvement obtained with urban data: the smarter the algorithm, the smaller the impact of the optimisation. Optimised Quadtree, Optimised Smart Quadtree and SParTSim provide close results for 8–10 partitions.

6 Conclusion

In this paper we propose the of use volume data to improve road partitioning for distributed simulations using SUMO. We modify three existing partitioning algorithms to take volume data into account. In general, the volume data can be extracted by a Transportation Management System for a city or by examining results from previous simulations. We show the impact of volume data on the individual partitioning algorithms for the partition topology, as well as the impact on the distributed simulation by comparing communication overhead and workload balance between the different algorithms.

We show that partition algorithms have a large impact for distributed simulation, either providing workload balanced partitions or reducing the overall communication overhead. SParTSim, the algorithm trying to optimize for both cases, has a long runtime making it impractical for dynamic load balancing. By using traffic volume, we can improve the workload balance of simple spatial partitioning algorithms, which could make them useful for dynamic repartitioning of large simulations. This means that in order to be able to scale and distribute large-scale simulations with dSUMO, the focus for dSUMO should be on the communication overhead with external systems, as balanced partitioning has been shown reduces the overall computation time. On the other side, we also show that the optimization using traffic volume has its limitation and cannot make a simple algorithm such as Smart QuadTree as reliable as a urban traffic dedicated partitioning algorithm such as SParTSim.

Acknowledgment This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie)

References

1. Abbott J (2013) State of the world's cities: prosperity of cities, Australian Planner, pp 1–2
2. Karypis G, Kumar V (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J Sci Comput 20:359–392
3. Soares G, Macedo J, Kokkinogenis Z, Rossetti RJ (2013) An integrated framework for multi-agent traffic simulation using SUMO and JADE. In: SUMO2013, The first SUMO user conference, 15–17 May 2013—Berlin-Adlershof, Germany, pp 125–131
4. Bellifemine F, Bergenti F, Caire G, Poggi A (eds) (2005) JADE—a java agent development framework. Multi-agent programming, Springer, pp 125–147
5. Bragard Q, Ventresque A, Murphy L (2013) dSUMO: towards a distributed SUMO. In: SUMO2013, The first SUMO user conference, 15–17 May 2013—Berlin-Adlershof, Germany
6. Ventresque A, Bragard Q, Liu ES, Nowak D, Murphy L, Theodoropoulos G et al (2012) SParTSim: a space partitioning guided by road network for distributed traffic simulations. In: Proceedings of the 2012 IEEE/ACM 16th international symposium on distributed simulation and real time applications, pp 202–209
7. Finkel RA, Bentley JL (1974) Quad trees a data structure for retrieval on composite keys. Acta Informatica 4:1–9
8. Wang Y, Lees M, Cai W (2012) Grid-based partitioning for large-scale distributed agent-based crowd simulation. In: Proceedings of the winter simulation conference, p 241
9. Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51:107–113
10. Amanatides J Woo A (1987) A fast voxel traversal algorithm for ray tracing. In: proceedings of EUROGRAPHICS, pp 3–10
11. Radha H, Vetterli M, Leonardi R (1996) Image compression using binary space partitioning trees. Image Process IEEE Trans 5:1610–1624
12. Torres E (1990) Optimization of the binary space partition algorithm (BSP) for the visualization of dynamic scenes. In: Eurographics, pp 507–518
13. Steed A, Abou-Haidar R (2003) Partitioning crowded virtual environments. In: Proceedings of the ACM symposium on virtual reality software and technology, pp 7–14
14. Freisleben B, Hartmann D, Kielmann T (1997) Parallel raytracing: a case study on partitioning and scheduling on workstation clusters. In: Proceedings of the thirtieth hawaii international conference on system sciences, 1997, pp 596–605
15. Dhillon IS (2001) Co-clustering documents and words using bipartite spectral graph partitioning. In: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining, pp 269–274
16. Pothen A, Simon HD, Liou K-P (1990) Partitioning sparse matrices with eigenvectors of graphs. SIAM J Matrix Anal Appl 11:430–452
17. Hendrickson B, Leland R (1995) An improved spectral graph partitioning algorithm for mapping parallel computations. SIAM J Sci Comput 16:452–469
18. Kernighan BW, Lin S (1970) An efficient heuristic procedure for partitioning graphs. Bell Syst Tech J 49:291–307
19. Fjällström PO (1998) Algorithms for graph partitioning: a survey. linköping electron art comput inf sci 3(10):1–37
20. Hendrickson B, Leland RW (1995) A multi-level algorithm for partitioning graphs. SC 95:28
21. Andreev K, Racke H (2006) Balanced graph partitioning. Theor Comput Syst 39:929–939
22. Karypis G, Aggarwal R, Kumar V, Shekhar S (1999) Multilevel hypergraph partitioning: applications in VLSI domain. IEEE Trans Very Large Scale Integr VLSI Syst 7:69–79
23. Lowrie P (1990) Scats, sydney co-ordinated adaptive traffic system: a traffic responsive method of controlling urban traffic
24. SUMO (2014) TAPAS-Cologne dataset. http://sourceforge.net/apps/mediawiki/sumo/index.php?title=Data/Scenarios/TAPASCologne
25. Simpson EH (1949) Measurement of diversity. Nature 163:688

26. Varschen C, Wagner P (2006) Mikroskopische modellierung der personenverkehrsnachfrage auf basis von zeitverwendungstagebüchern. Stadt Reg Land 81:63–69
27. Boukerche A, Das SK (1997) Dynamic load balancing strategies for conservative parallel simulations. In: Proceedings of 11th workshop on parallel and distributed simulation, 1997, pp 20–28
28. Devine KD, Boman EG, Heaphy RT, Hendrickson BA, Teresco JD, Faik J et al (2005) New challenges in dynamic load balancing. Appl Numer Math 52:133–152