SParTSim: A Space Partitioning Guided by Road Network for Distributed Traffic Simulations

Anthony Ventresque*, Quentin Bragard*†, Elvis S. Liu*‡, Dawid Nowak*,
Liam Murphy*, Georgios Theodoropoulos‡ and Qi Liu§
*Lero@UCD, School of Computer Science and Informatics, University College Dublin, Ireland
Email: {anthony.ventresque,dawid.nowak,liam.murphy}@ucd.ie
†Institut National des Sciences Appliquées de Lyon, Villeurbanne, France, Email: quentin.bragard@insa-lyon.fr
‡IBM Research, Dublin Research Lab, Ireland, Email: {elvisliu,georgetheo}@ie.ibm.com
§IBM T. J. Watson Research Center, NY, USA, Email: liuqi@us.ibm.com

***Abstract*—Traffic simulation can be very computationally intensive, especially for microscopic simulations of large urban areas (tens of thousands of road segments, hundreds of thousands of agents) and when real-time or better than real-time simulation is required. For instance, running a couple of what-if scenarios for road management authorities/police during a road incident: time is a hard constraint and the size of the simulation is relatively high. Hence the need for distributed simulations and for optimal space partitioning algorithms, ensuring an even distribution of the load and minimal communication between computing nodes. In this paper we describe a distributed version of SUMO, a simulator of urban mobility, and SParTSim, a space partitioning algorithm guided by road network for distributed simulations. It outperforms classical uniform space partitioning in terms of road segment cuts and load-balancing.**

***Keywords*-Parallel and Distributed Simulation; Space Partitioning; Graph Partitioning; Road Network; Traffic Simulation**

I. Introduction

It is estimated that 70% of world population will reside in cities by 2050, and even likely 86% in developed countries [1]. This situation has led to severe difficulties for municipalities, governments, citizens and businesses for decades, or more. For traffic management, the consequences are clear: in Dublin, between 1994 and 2002 journey time of car commuters increased by 44% (21.24 to 30.65 mins) and average speed decreased by 31% (16.71 to 11.58 km/hr) [2]; by 2000, the number of people driving to work in Dublin was 50% [3]. Dublin City Council made lot of efforts to reduce car utilisation: more bus lines, cycle lanes in all major routes, etc. and the number of car only commuters has decreased to 34.6% in 2008 [2]. But entering the city or passing through bridges over the river Liffey is still difficult, while things that worked in the 2000s cannot be applied easily any more (many buses are overloaded, it is difficult to add new cycle lanes, etc.). An approach could be to leverage on the data avalanche (city data from sensors, governmental applications, social network feeds, mobile applications data, etc.) to design a precise and relevant prediction tool for traffic management. For instance, road authorities could be keen to see the impact of their possible decisions on traffic conditions, after simulating what-if scenarios for near future (eg. next 30 minutes). Obviously this is critical in terms of performance: the scale of the simulation is important (large urban area), execution-time has to be short (a quick decision has to be made) and number of parameters to process is large (microscopic simulations of individual agents). Even though there exist some very interesting and promising high performance computing systems and projects[1], we doubt being able to run a simulation with these requirements on a single server. Hence the need for distributed simulation, ie. a distributed system where each server runs a division of the space and the servers synchronise their runs. The question is now: *How can we best partition a road network for a distributed simulation?*

This question has been addressed in 'open' space partitioning, for instance in massively multiplayer online role-playing game. In such games a large 'virtual world' is managed by several servers, each of them being in charge of a region. When players and non-player characters reach the border between regions and either need information about contiguous regions, or cross the borders, servers exchange messages. Classical solutions for space partitioning are uniform partitioning (eg. regular grid layout) and non-uniform (with more complex shapes). Usually, distributed vehicular mobility simulations use the former: they apply a grid layout (tiles, hexagons, etc.) on a road map and assign each partition to a server. This is more efficient, in terms of complexity of look-up computation, and easier to implement. However this does not take into consideration the characteristics (eg. shape, length, etc.) of road networks. We claim that a space partitioning for distributed traffic simulation needs to leverage on the graph-like structure of the road network. In fact, a solution to this problem has more to deal with graph partitioning than space partitioning, while

[1]For instance IBM's exascale project: http://researcher.watson.ibm.com/researcher/view_project.php?id=2564

road network is a very unique kind of graph. We propose in this paper SParTSim, a space partitioning that uses the graph network to get better load balancing with minimal inter partition communication.

The rest of this paper is structured as follows. Section II defines the problem of road network partitioning and the metrics that describe the quality of a partitioning. Section III presents the various classical techniques for space partitioning (the main problem we address) and graph partitioning (the inspiration for our solution). Section IV introduces our road network partitioning and section V the implementation of this algorithm for a distributed microscopic traffic simulator. We show some evaluations in section VI and finally we discuss our approach and conclude in section VII.

II. Problem Definition

Our problem of space partitioning for urban mobility simulation needs a precise definition of the object under consideration: road network. It is indeed a particular kind of graph and the partitioning is a graph function aiming at splitting it by delimiting subsets of vertices and arcs. We also define the cutting cost, which represents the number of arcs that join the partitions, and the load-balancing, which regards the uniformity of the partitioning.

Definition 1 (Road Network): A *road network* is a weighted directed multigraph $G = (V, A, w)$ where $V = \{v_i, \ldots, v_n\}$, $n \in \mathbb{N}$, is a set of vertices denoting notable road elements: junctions, ends, etc., $A = \{a_1, \ldots, a_m\}$, $m \in \mathbb{N}, \forall a_i, \exists (v_j, v_k), a_i = (v_j, v_k)$ is a set of arcs (directed edges) representing road segments between vertices, and w is a function assigning a weight to each arc and denoting the length of this segment of road.

A partitioning of this road network is a set of subsets of the original network: every vertex of the original road network is allocated to one partition, and arcs between two partitions become inter partition links. We also make sure that partitions are connected subgraphs of the original road network.

Definition 2 (Road Network Partitioning): Let $G = (V, A, w)$ be a road network. A set function: $\pi(G, n) : G \times \mathbb{N} \mapsto (\mathcal{P}(G))^{\mathbb{N}}$ is a partitioning of G such that $\pi(G, n) \to \{\gamma_i\}$, $|\{\gamma_i\}| = n$, $\forall \gamma_i, \gamma_i[A] \subseteq G[A]$ and $V = \bigcup_{i=1}^{n} \gamma_i[V]$. $\forall v_j, v_k \in \gamma_i[V]$ we have a path between v_j and v_k: $\exists \{a_o, a_1, \ldots, a_l, a_f\}$, $a_o, a_1, \ldots, a_l, a_f \in \gamma_i[A]$, $\exists v_o, v_f \in \gamma_i[V], a_o = (v_j, v_o), a_f = (v_f, v_k)$ and $a_p[V] \cap a_{p+1}[V] \neq \emptyset$, $p \in [1..l-1]$. That ensures us that the graph of each partition is connected. We also introduce new kind of arcs that replace the ones cut by the partitioning and keep track of the links between partitions: $\forall a_i = (v_m, v_m), a_i \in G[A] \wedge a_i \notin \bigcup \pi(G, n)$ we have a new link l_i, $l_i = (v_n, v_m) \wedge w(l_i) = w(a_i)$, and we have $\pi(G, n)[L] = \{l_i\}$.

Once the partitions are made, they are distributed to various computing nodes. The border between any two partitions of the network is then at the junction between two computing nodes and they have to exchange messages to make it a one-piece, seamless, simulation. This has a cost, the cost associated with the number of messages that need to be sent over the borders. This cost can only be computed at execution time, as it implies to run the simulation and monitor the number of messages between computing nodes. However, a good approximation of this quantity of messages, is the number of lanes crossing the borders.

Definition 3 (RNP Splitting Cost): We define the splitting cost of partitioning a road network as the sum of segments that link partitions. Given a road network $G = (V, A, w)$ and a RNP function π, we have:
$cut(\pi(G, n)) = \sum_{i=1}^{|\pi(G,n)[L]|} w(l_i)$

The previous cost is not the only quantity measuring the performance of the partitioning, as it does not tell how much work each computing node needs to process. In fact, the best partitioning is probably the one that makes sure all nodes finish their computation at the same time, and then minimise the delay needed for the synchronisation. Actually a distributed simulation system must be *balanced*, i.e. it feeds each computing node with the adequate work quantity it can process, to make sure all the nodes do their job at the same pace, and do not retard the others. Note that in this paper we do not address the computing nodes heterogeneity, and assume each computing node to be equal to the others, in terms of capacity. We also focus on an a priori partitioning, not considering traffic density. Balancing the partitions in this context consists in making sure there is a similar road segment length for every partition, $epsilon$ being an acceptable difference with average load.

Definition 4 (ϵ-Balancing): A RNP is *ϵ-balanced* iff $\forall \gamma_i \in \pi(G, n)$, $\frac{|G[A]|}{n} - \epsilon \leq |\gamma_i[A]| \leq \frac{|G[A]|}{n} + \epsilon$, with ϵ an arbitrarily positive or null quantity.

The best road network partitioning is obviously a RNP, such that the RNP cut cost is minimal and the balancing is perfect: $\epsilon = 0$. Our objective is to minimise the cuts cost and maximise the evenness.

III. Space and Graph Partitioning

This section explores two related fields: space partitioning and graph partitioning. We show that both are somehow relevant to our purpose, although we have a different perspective and in a way combine both.

A. Space Partitioning

Space partitioning is the most widely used interest management approach for distributed simulations. This approach limits the participants' interactions and communications within a small number of space partitions, or zones. Participants are connected to these zones in order to receive events and updates that are generated from them. A typical partitioning scheme allows participants to specify an area of interest (AOI), which consists of a radius of zones where

the participant is joining new ones at the leading edge and leaving old ones at the trailing edge as his avatars moves around the virtual space. Many systems that are compliant to HLA DDM [4] also adopt this type of schemes.

1) Uniform Partitioning: Schemes adopting uniform partitioning divide the virtual space into zones that are static, regular, with a uniform orientation, and a uniform adjacency. The two most common shapes adopted by the existing approaches are rectangles [5] and hexagons [6]. For different simulations, the reasons for adopting one of these shapes are varied, and are usually application dependent.

[7] presents an approach which defines static multicast groups that are overlaid on square zones. The authors asserted that using regular overlays might be wasteful when multicast groups are being allocated to zones where little or no activity will take place. On the other hand, irregular overlays can be arbitrarily shaped, which allows the developers to define multicast groups according to the terrain information. However, this also requires relatively more computation overhead, since there is no trivial solution for the overlap test between the circular AOI and zones of arbitrary shape. The developers of NPSNET prefer hexagonal zones over square [6]. The authors pointed out that using hexagons can approximate more closely a round shape AOI than rectangles, and therefore can increase the filtering precision. However, [8] argues that the interest matching process with hexagons is more complicated than squares, since it usually involves point-to-line distance formula. In MOPAR [9], a partitioning scheme for Massively Multiplayer Online Games (MMOGs) is proposed, which combines a Distributed Hash Table (DHT) overlay and peer-to-peer connections. This scheme divides the virtual space into hexagonal zones, while each zone has a corresponding home node via the DHT mapping. An approach which minimises the use of the DHT is also proposed in the paper.

2) Non Uniform Partitioning: Apart from partitioning the virtual space by some regular pattern, some schemes defines zones with different shape, size, and relative orientation. These approaches often employ a hierarchy data structure for space partitioning.

Spline [10] decomposes the virtual space into various subdivisions called 'locales', which may have an arbitrary shape and may be linked together by arbitrary transformations. Each entity resides in exactly one locale, where the participant's interactions are limited to the current locale and its immediate neighbors. A binary space partitioning tree (BSP tree) is employed to describe the boundary of a partition. [11] present four partition schemes, including: quadtree, k-dimensional tree (k-d tree), constrained k-d tree, and region growing. Both BSP tree and k-d tree share similar properties, except the partition planes of the latter must be axis-aligned. Although k-d trees are less flexible and may be harder to keep balanced than BSP trees, the query process can be faster since comparing an AOI with axis-aligned rectangles is simpler than with unregulated shapes. The region growing approach constructs the virtual space from bottom-up, which is based on a standard image processing algorithm for image partitioning. The choice of partitions is done according to the actual structure of the virtual space, which is of irregular pattern and arbitrary shape. [12] present a clustering approach that employs quadtrees to reduce the computational cost of the interest matching process. A quadtree is used to partition two-dimensional space by recursively subdividing it into four equal-size, axis-aligned subdivisions. Since the partition planes are fixed, partitioning by quadtree is less flexible than the BSP tree and k-d tree. In [13], an approach using Voronoi diagram is proposed. Given a number of points (sites) in a two-dimensional space, a Voronoi diagram partitions the space into the same number of zones (Voronoi regions), such that each Voronoi region contains all the points closer to the region's site than to any other site. No individual participant need to be aware of the entire diagram but only the zone covered by his boundary and enclosing Voronoi regions. This scheme, however, does come at the significant overhead of continual re-computation of the Voronoi diagram itself.

3) Granularity: Choosing a proper granularity is one of the major considerations for all partitioning schemes. For a static partitioning of the virtual space, a trade-off must be made. If the zones are large, each zone would contain a large number of virtual entities and thus the participants might receive a large amount of irrelevant data. On the other hand, if the zones are small, the number of zones as well as the number of multicast groups would become large, and therefore it increases the chance of subscribing and unsubscribing from multicast groups as the participants move around the virtual space, resulting in an increase in management overheads.

The hierarchical structures also suffer from the same problem but in a different form - a trade-off must be made when choosing a proper granularity of the leaf nodes or a proper height of the hierarchy. [12] and [11] tried to maintain a balanced hierarchy by setting a maximum height or a population threshold.

B. Graph Partitioning

Partitioning a graph is one of the most fundamental problems of computer science, with applications in many domains such as telephone network design, very large scale integration for circuit layout composition, computer vision, physical mapping of DNA, distributed computing, clustering, route planning, etc. (see for example [14], [15]) We proposed a specific description of this problem in section . In short, it consists in splitting a set of vertex such as 'load' is even among the clusters generated and 'communication' between clusters minimised. Most instances of this general problem are reputed to be NP-hard [16] and given its criticality in so many areas, it has garnered a lot of attention

and a rich literature. Most techniques and heuristics use a geometric [14] or spectral [17] partitioning, and one of the most popular transversal idea is to use multilevel partitioning [18]. Here the graph is step by step decomposed into coarser, smaller versions. Smaller graphs are easier to partition and the first cuts are refined while the graph is recomposed.

We think that the previous approaches lack in their own way the particularities of road networks. Space partitioning hardly captures the fact that vehicles move on a network, ie. a graph structure, and it seems tricky and inappropriate to use the genuine space partitioning techniques here. On the other hand, pure graph partitioning techniques are very powerful (and we are not competing with them) but a little too complex for 'simple' graphs like road networks where a lot of information is not hidden but explicit: what are the important roads, where are some structural elements such as bridges, etc.

IV. SParTSim Algorithm

In this section, we present our intuitions and the implementation of a region growing algorithm. Those intuitions are that roads form a *hierarchy* in the infrastructure, with some roads more important than others. While this is rather basic, it helps to easily define a hierarchical partitioning. Another simple idea is that some elements of the road network have a specific impact on the traffic as they have a high centrality degree. Those are important elements that bridge different areas in the road network.

A. Road Hierarchy

Roads are organised in hierarchies: each country has implemented a structure of motorways, national roads, highways, large roads, smaller multi-lane roads, small one-way roads, etc. In the Republic of Ireland, there are six levels in the road hierarchy: National Primary Roads and Motorways, National Secondary Roads, Regional Roads and three levels of Local Roads.We assume in this paper that using this hierarchy is helpful for a partitioning of urban area, as commuting (up to half of the number of trips in a urban area [1]) generally links low levels: typically residential areas to industrial and commercial estates, through high levels: highways, motorways, national roads, etc. [2]. Partitioning the urban network graph with a bit of the highest levels (eg. in Dublin, M50 and main N itineraries) and the incident smaller roads would then make sense.

Our algorithm uses the explicit road hierarchy and generates a first partitioning at the highest level, then refining at lower levels. This heuristic has proven to be very efficient in Karypis and Kumar [18].

B. Explicit Natural Bridges

Other important structural elements that can easily be collected and contain useful information are the *natural bridges* of the road network. Indeed, some elements have a very high centrality value: bridges, mountain passes, borders between countries, etc. Those are structural elements through which most of the traffic has to flow and that are important in the road network (hence the high centrality, ie. the fact that many routes go through them). Several works have addressed this problem with graphs, sometimes applying their solutions to road networks [19]. But in all fairness, we do not need to compute the natural cuts/bridges as the vast majority of them (we use OpenStreetMap, see Section V, and we usually find this information in OSM files) are explicitly encoded in the map. So our algorithm aims at using this information to direct the road network partitioning, trying to put natural bridges at the border between partitions (we discuss this in conclusion as well).

C. SParTSim

SParTSim is a multi-level region growing road network partitioning. We define a region growing as an algorithm that grows regions in all directions from an initial vertex. In our case, we have various regions growing concurrently and competing for the vertices and arcs. A region tries to incorporate gradually all vertices and arcs around it, until it reaches the boundaries of other regions.

Algorithm 1 shows in lines 6-11 the growing process: every region grows and the process stops when none can increase its size. Note that the function `BestCandidateVertex` presented line 4 picks the best starting point for each region according to some predefined heuristics: that could be an homogeneous repartition over the map, or driven by some map elements (natural bridges, highest road hierarchy), etc. In this paper, we consider nodes with the highest degree as they are likely to see huge traffic passing through them. Function `Grow` (line 9) manages the growing part of SParTSim. Every vertex in the partition γ_i gets its neighbors and γ_i ranks them according to the level of the road segment, the length of the segment, and whether this neighbor is in this partition, in no partition or in another partition. The algorithm always picks the road segment with the highest level in the road hierarchy, but when there are several possible nodes linked to the partition with the top level road segments, they are ranked according to other elements: length and status of the node. We have tried different weighting schemes to rate the importance of these characteristics but (i) have not found any clear evidence that having different non uniform weights has an impact on the performance of the algorithm (ii) do not have enough space here to present them. Note that `Grow` returns a boolean value, the meaning of false being 'not able to grow'. When no partition is able to grow, this first processing step of SParTSim stops.

Once every region has grown to its maximum (ie. reaching the border of the map or being in contact with other partitions), it is unlikely to be balanced, and algorithm 1 then initiates a trading process, each region trying to get closer to

Algorithm 1: SParTSim

input : A road network $G = (V, A, w)$; n the number of partitions wanted; ϵ an acceptable unbalance of the partitioning.
output: $\pi(G, n)$ a road network partitioning.

```
1  begin
       // Initialisation
2      stop⃗ ← new array of size n;
3      for i ← 1 to n do
4          γ_i ← BestCandidateVertex();
5          stop⃗[i] ← 1;
       // Region growing
6      while stop⃗ ≠ ∅⃗ do
7          for i ← 1 to n do
8              if stop⃗[i] ≠ 0 then
9                  hasGrown ← Grow(γ_i);
10                 if hasGrown then
11                     stop⃗[i] ← 0
       // Balance partitioning
12     balanced← false;
13     while ¬ balanced ∨ enough iterations do
14         γ_i = max π(G, n);
15         γ_j = min π(G, n);
16         if |γ_i[V]| − ε < |G[V]|/n < |γ_j[V]| + ε then
17             balanced ← true;
18         else
19             Trade(γ_i, γ_j);
       // Ensure connectivity
20     foreach γ_i do
21         ComputeConnectedSubgraphs(γ_i);
22     foreach connected component cc^j_{γ_i} do
23         Attach(cc^j_{γ_i});
24 end
```

an average value of segment length (lines 12-19). SParTSim takes the partitions with the maximum (γ_i) and minimum (γ_j) segment length from the set of partitions (lines 14 and 15). If they both have an acceptable unbalance (ie. between $\frac{|G[V]|}{n} + \epsilon$ and $\frac{|G[V]|}{n} - \epsilon$), then the load-balancing process is finished. Otherwise, SParTSim tries to move a certain quantity (difference in segment length between the two partitions divided by 2) from γ_i to γ_j. The trading itself consists in finding the shortest path between γ_i and γ_j and to trade connected subsets of partitions along this path. In short, a wave of trades propagates the excess of arcs from γ_i to γ_j, ensuring that there is no major impact on the overall structure of partitioning.

After `Trade`, partitions are well-balanced, but their graph may not be connected. This is obviously not desirable and the next steps of the algorithm aims at solving this issue. First, `ComputeConnectedSubgraphs` (line 21) computes all the connected components of each partition. The idea is then to cluster those components with two requirements and one objective: (r1) two components can be aggregated only if they have a common segment; (r2) there must be n partitions at the end of the process; (o1) SParTSim has to keep the partitions balanced. `Attach` (line 23) finds all the possible aggregations of connected components and makes the best decision according to these ideas.

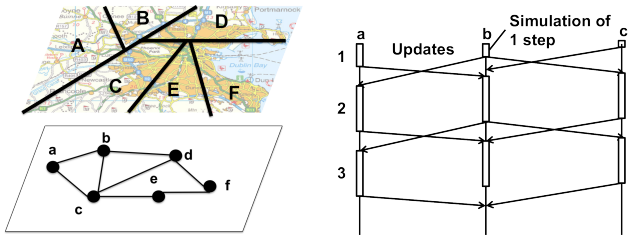
V. A Distributed Simulator

We propose a rather simple yet efficient solution for distributed simulation. Although this may not be the best possible architecture, it addresses the various problems related to the definition of an architecture for distributed simulation.

We use SUMO [20] for our individual computing nodes as it is a very efficient, portable and microscopic simulator that can scale up to millions of entities and hundreds of thousands of road segments. There is no distributed version of this tool to our knowledge, and our objective here is to provide a scalable one to the community. SUMO runs in discrete time and continuous space (hence the impression that cars leap between two simulated steps) and computes turn-by-turn next position for each vehicle according to their characteristics (speed, acceleration, position, other vehicles, infrastructure, driver's behavior, etc.).

Figure 1 (a) shows our general goal: a map (of Dublin here) is partitioned by an algorithm (see section IV above) and each partition is assigned to a computing node that is responsible for it. This node simulates traffic in its region and sends vehicles to its neighbors when they cross borders. Our synchronisation protocol provides useful properties (see Figure 1 (b)): at the end of each turn, each computing node sends update messages to each of its neighbors. The messages are empty if no car crossed the border. Otherwise, the messages describe the characteristics and IDs of the vehicles that are passing from this node to its neighbor. This protocol has useful properties: (i) no central entity is required to pass messages: updates are only sent to neighbors. Systems that assume or need such a central entity see a decrease of their performance [21]. Therefore, we expect a better scalability of our solution; (ii) no central clock or any complex synchronisation mechanism is needed: each computing node waits until it receives update messages from all its neighbors before starting the next turn; (iii) any node failure, message loss, etc. is detected by all nodes in the system after only a few turns (the maximum being the diameter of the system). Stopping a simulation in such a distributed system is then easy and getting it to restart is likely to be as simple.We call our protocol semi-optimistic, as every node runs as long as it receives updates from

neighbors: no node matters about other long distance nodes, while some sort of synchronisation is ensured.



(a) (b)

Figure 1. (a) A map is partitioned and each region is assigned to a computing node (simulator) (b) Synchronisation protocol.

Regarding the city network and its graph representation, we use Open Street Map[2], a collaborative open data project for geographical data collection and mapping. Registered users are able to add (upload), edit, remove, etc., worldwide geographical information, like what Wikipedia users can do. We extract OSM files for urban areas (eg. Dublin), clean it a little (OSM data can be very dirty) and upload it in Neo4J [3], a graph database. That is more handy and efficient for our many graph processing needs (search, intense neighbors finding, etc.).

VI. Analysis

This section aims at showing that our solution performs better than some classical space partitioning techniques.

A. Presentation of the Testing System

The techniques we compare our algorithm to are:

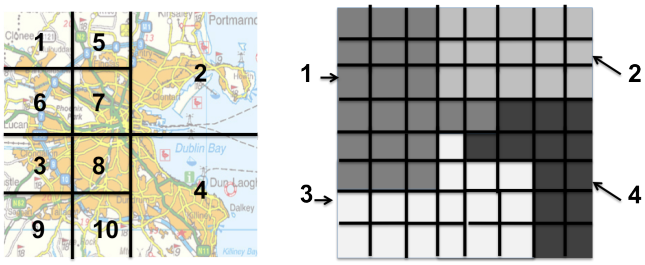
- A simple quad-tree solution (also called QT1, see Figure 2). This solution is very simple, map can be split in 4 identical tiles, and any tile can be further split, etc. We direct the decomposition using a density-aware heuristic, trying to always split denser regions.
- QT2, our implementation of a 'smart' quad-tree (see Figure 2 (b)). This solution uses a divide-and-cluster approach, consisting in splitting the plan in small tiles (with a predefined threshold parameter) and then regrouping some of them to form complex shapes, 'a la Tetris'. Figure 2 (b) for instance shows 4 partitions having rather complex shapes.

All our experiments run on Windows 7 on an Intel Pentium P6200 Dualcore 2.13GHz laptop with 4GB RAM.

It is worth noting that QT1 cannot split the space in any random number of partitions. Due to its rather fixed 'quad' nature, the map can only be decomposed in $3n+1$ regions. On the contrary QT2, because it tessellates the map in a multitude of small tiles and then cluster them, can generate

[2]http://www.openstreetmap.org/

[3]http://neo4j.org



(a) (b)

Figure 2. (a) Simple quad-tree partitioning and (b) our 'smart quad-tree'

any number of partitions. It is the same for SParTSim. Our experiments then consider few servers $(1, 4, 7, 10, 13, 16)$ for QT1 and all the natural numbers between 1 and 16 for the other two techniques.

B. Metrics

We have presented in section III-B the two main perspectives we have on partitioning algorithms performance: links between computing nodes and load-balancing.

First, we compute the number of relations between servers, ie. how many neighbors a single computing node has. This is an important element as it shows how many connections need to be maintained by servers. The lower this quantity is, the better. Then we plot the number of links between servers, which corresponds to the number of road segments and their size. Crossing the borders between partitions, they are likely to generate messages between nodes. The lower this quantity is, the better.

We consider that, given the lack of traffic density information, load is represented by the length and size of road segments (see Section III-B). We chose the Simpson diversity index [22], which is highly regarded when one needs to check the balance between populations (partitions in our case): $DI(\pi(G,n)) = \frac{1}{length(G)^2} \sum_{\gamma_i \in \pi(G,n)} |length(\gamma_i)|^2$. This formula ensures that the index is in $[0..1]$. $length$ is a function giving the length and size of the roads in a graph.

C. Results

Figure 3 presents the number of relations between partitions (ie. active connections) and shows that QT1 generates a very predictable number of relations between partitions, as any partition has always a few but known number of neighbors. For instance when 10 partitions are created (see Figure 2), we know that the maximum is 4 and the minimum 2 (corner partitions). QT2, and more importantly SParTSim, create less inter-partition relations, and that is an advantage. It also seems that those techniques diverge more and more with QT1 (the more servers, the bigger the difference) and that SParTSim always outperforms both.

Figure 4 plots the number of links (road segments and their weight) between partitions. It is quite unexpected, but

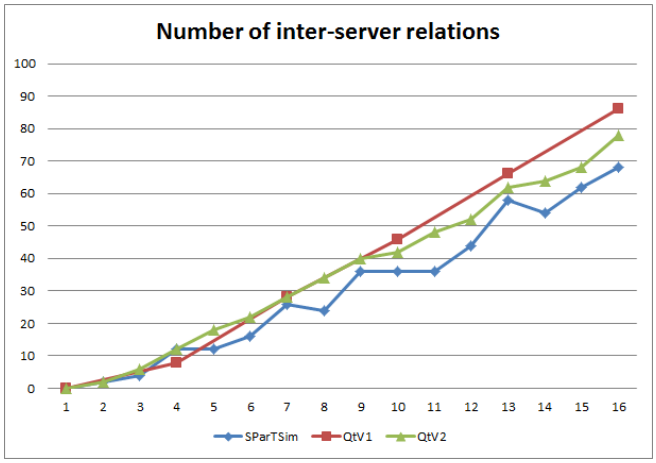


Figure 3. Number of relations between servers.

QT1 performs better, SParTSim being very close to it. It seems that the regular, square shape of QT1's partitioning is the most efficient anyway and that more complex shapes never achieve as well. QT2 for instance generates 2 to 3 times more road cuts than QT1. SParTSim, while not being able to have better score than QT1, is very close, and in some occasions gets the exact same values.

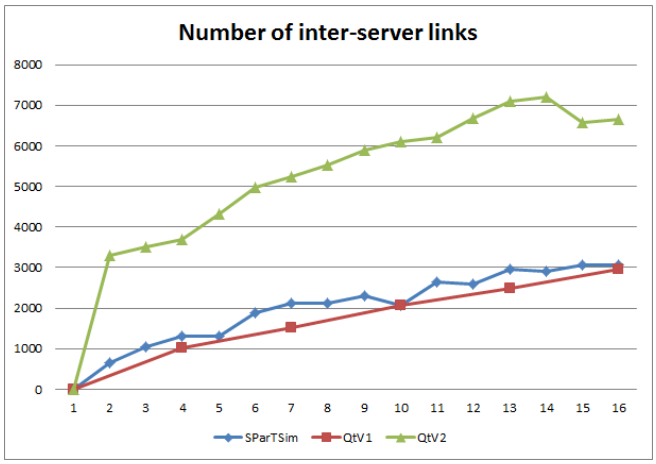


Figure 4. Number of road segments cuts.

Regarding load-balancing, QT1 has very poor results compared with both QT2 and SPartSim, those having a very high evenness (see Figure 5). It is worth noting that evenness for QT1 is quickly stable at a low value (0.7) which is likely to tell something about Dublin's road structure, more than about the algorithm itself. QT2 and SParTSim have better scores, and the latter seems to outperform for high number of partitions.

Regarding execution time (see Figure 6), QT1 and QT2 run in constant time (100 and 150 seconds respectively),

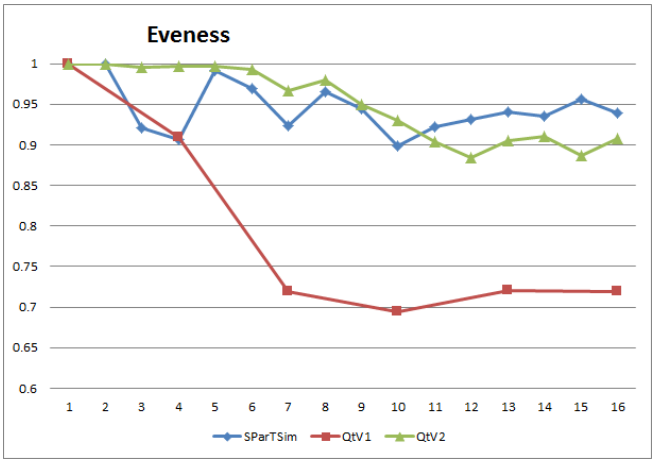


Figure 5. Evenness of the partitions.

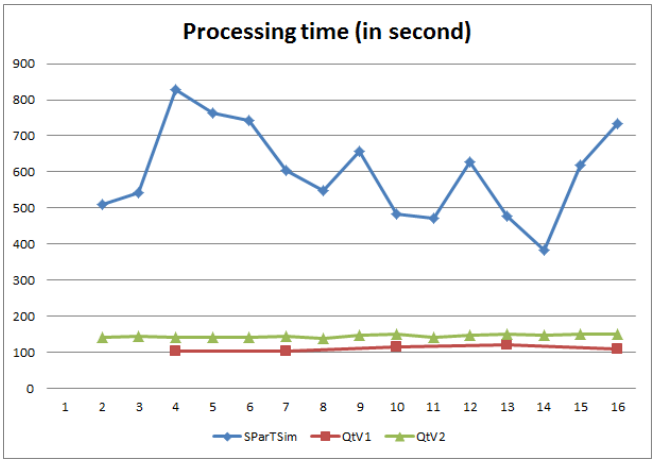


Figure 6. Execution time.

while SParTSim is very dependant on the network structure and its execution time is 600 seconds on average (min. is 400 seconds for 14 partitions, max. is 820 for 4 partitions). This is very good as (i) QT1 and QT2 do not need a lot of computation and SParTSim is definitely not far from them (ii) its execution time is not increasing.

VII. Conclusion

This paper presents a space partitioning algorithm for road networks and the implementation of a distributed microscopic traffic simulator. Our algorithm differs from traditional space partitioning approaches as it tries to use the road network graph-like structure; and also from the 'abstract' graph partitioning solutions as we leverage on the characteristics of a road network. We show that a hierarchical partitioning can easily be implemented, all the structural information being available freely on Open Street Map. It is worth noticing that we do not challenge the very

efficient space and graph partitioning solutions, as they have been studied thoroughly for many years and garnered a lot of attention. SParTSim is more a smart and well-adapted implementation of a partitioning heuristic, aiming at having a quick and good partitioning given a road network and all the information associated with it.

Our validations show that our algorithm is better than state-of-the-art space partitioning solutions, both in terms of number of road cuts and load-balancing. SParTSim is a very efficient solution (almost constant computation time, only about five times the very simple geometrical partitioning QT1 and QT2).

We have several objectives for the future: (i) compare our approach to graph partitioning and not only to space partitioning; (ii) apply the same algorithm to other urban areas to check whether Dublin city has some specificities and whether our algorithm still performs better than the baselines; (iii) improve some parameters through more experimental study: choice of the first node of the region growing, impact of the natural bridges, etc. (iv) apply it in a large scale distributed simulation to add and test other parameters for SParTSim (eg. density) and see whether we have even better performance; (v) implement a dynamic version of the partitioning, by adding/removing servers during the simulation.

ACKNOWLEDGEMENT

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] U. Habitat, "State of the world's cities 2010/2011 - cities for all: Bridging the urban divide," http://www.unhabitat.org/pmss/listItemDetails.aspx?publicationID=2917.

[2] "Dublin city council workplace travel plan," http://www.dublincity.ie/RoadsandTraffic/Documents/DCC_Travel_Plan_v2_1[1].doc.

[3] E. L. W. Morgenroth, "Analysis of the economic, employment and social profile of the greater dublin region," http://www.dublinpact.ie/pdfs/profile.pdf, 2001.

[4] DMSO, "High Level Architecture Interface Specification Version 1.3," 1998. [Online]. Available: http://hla.dmso.mil

[5] D. J. Van Hook, S. J. Rak, and J. O. Calvin, "Approaches to Relevance Filtering," in *Workshop on Standards for the Interoperability of Distributed Simulations*, 1994, pp. 26–30.

[6] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham, "Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments," in *VRAIS*, 1995, pp. 2–10.

[7] S. Srinivasan and B. R. D. Supinski, "Multicasting in DIS: A Unified Solution," in *Electronic Conference on Scalability in Training Simulation*, April-June 1995.

[8] K. Prasetya and Z. D. Wu, "Performance Analysis of Game world Partitioning Methods for Multiplayer Mobile Gaming," in *ACM SIGCOMM Workshop on Network and System Support for Games*, 2008, pp. 72–77.

[9] A. P. Yu and S. T. Vuong, "MOPAR: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games," in *workshop on Network and operating systems support for digital audio and video*, 2005, pp. 99–104.

[10] J. W. Barrus, R. C. Waters, and D. B. Anderson, "Locales and Beacons: Efficient and Precise Support For Large Multi-User Virtual Environments," in *VRAIS*, 1996, pp. 204–213.

[11] A. Steed and R. Abou-Haidar, "Partitioning Crowded Virtual Environments," in *VRST*, 2003, pp. 7–14.

[12] D. J. Van Hook, S. J. Rak, and J. Calvin, "Approaches to RTI Implementation of HLA Data Distribution Management Services," in *Workshop on Standards for the Interoperability of Distributed Simulations*, 1997.

[13] S.-Y. Hu, J.-F. Chen, and T.-H. Chen, "VON: A Scalable Peer-to-Peer Network for Virtual Environments," *IEEE Network*, vol. 20, pp. 22–31, 2006.

[14] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970.

[15] S. N. Bhatt and F. T. Leighton, "A framework for solving VLSI graph layout problems," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 300–343, 1984.

[16] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[17] A. Pothen, H. D. Simon, and K.-P. Liou, "Partitioning sparse matrices with eigenvectors of graphs," *SIAM Journal on Matrix Analysis and Applicationa*, vol. 11, no. 3, pp. 430–452, 1990.

[18] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, Dec. 1998.

[19] D. Delling, A. V. Goldberg, I. Razenshteyn, and R. F. F. Werneck, "Graph partitioning with natural cuts," in *IPDPS*, 2011, pp. 1135–1146.

[20] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo - simulation of urban mobility: An overview," in *SIMUL*, Barcelona, Spain, 2011.

[21] T. Suzumura, S. Kato, T. Imamichi, M. Takeuchi, H. Kanezashi, T. Ide, and T. Onodera, "X10-based massive parallel large-scale traffic flow simulation," in *ACM SIGPLAN X10 Workshop*. New York, NY, USA: ACM, 2012, pp. 31–34.

[22] E. H. Simpson, "Measurement of diversity," *Nature*, vol. 163, no. 4148, 1949.