Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. D. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

An evaluation of space partitioning methods and meta-heuristics based graph partitioning methods for partitioning road network simulations

Aravind Vasudevan[1], Quentin Bragard[2], Anthony Ventresque[2] and David Gregg[1]

[1] School of Computer Science and Statistics, Trinity College Dublin
[2] Computer Science Department, University College Dublin

ABSTRACT

Abstract goes here.

1 MOTIVATION

Introduction goes here.

Our **key contributions** in this paper are:

-
-
-
-

The rest of the paper is arranged as follows.

2 RELATED WORK

3 FORMALIZATION OF THE PROBLEM STATEMENT

In this paper we conduct a comparison of different methods of partitioning a road network graph based on some metrics. This section defines what a road network graph is and presents the formal definition of the metrics.

3.1 The Road Network Graph

The road network of a city can be represented by a directed cyclic graph (Holden and Risebro 1995) $G(V,E)$ where V denotes the vertex set and E denotes the edge set. Every edge $e_{ij} \in E$ in the graph represents a unidirectional road in the city that connects intersection v_i to intersection v_j. Every vertex $v_i \in V$ denotes an intersection of two or more roads. A weight w_{ij} is associated with the edge e_{ij} that is representative of the traffic that flows through that road. As discussed in Section 6, we intend to increase the number of weights that can be associated with every edge to be able to represent the number of lanes, length of the road, importance of the road etc. as part of our future work.

3.2 Partitioning a graph

For the sake of completeness of the graph definition, we also assign weights to the vertices denoted by W_i, which is defined as follows :

$$W_i = \sum_{\forall j \in neighbours(i)} w_{ij} \tag{1}$$

where $neighbours(i)$ is the set of all nodes in V that receive an outgoing edge from v_i.

The problem of partitioning a graph has a 2 dimensional space, where one axis represents the node ID(representing the intersection) and the other represents the partition ID. Each point in this 2-D space represents an $\{intersection,partition\}$ pair which implies this intersection is mapped onto this partition. We define a "state" to be a collection of $|V|$ points such that each intersection is mapped to exactly one partition. The total number of possible "states" in this discrete space is $|V|^p$, where p is the number of required partitions.

3.3 Formalizing the objective function

To effectively perform a distributed road network simulation, one must fully understand the effects of dividing the workload(the road network graph to be simulated) into the required number of partitions. To this end, we define three heuristics{cite papers} that are widely accepted in the community, to measure the effectiveness of a "partitioning scheme".

We define a partitioning scheme on the given road network G, $\lambda_G \in \Lambda_G$(where Λ_G is the set of all possible partitioning schemes of the given graph G onto p partitions) as a function that maps every node v_i from the road network graph to a partition. Please note that we will use the terms "node", "vertex" and "intersection" interchangeably.

$$\lambda_G(v_i) \in \{P_0, \ldots, P_{p-1}\}, \forall v_i \in V \tag{2}$$

where P_i denotes the i^{th} partition. For the following discussion we denote the weight of the i^{th} partition(P_i) as $w(P_i)$. One of the primary objectives of road network partitioning for distributed simulation is to minimize the execution time of the simulation time. To this end, we aim to minimize the weight of the **heaviest partition(λ_G^{max})** as described by the following equation,

$$\lambda_G^{max} = max(w(P_i)), \forall P_i \in \{P_0, \ldots, P_{p-1}\} \tag{3}$$

The execution time of a distributed simulation also depends on the amount of **inter-partition communication(λ_G^{comm})**. In order to reduce this inter-partition communication, we minimize the following metric,

$$\lambda_G^{comm} = \sum_{i \in \{v_0,\ldots,v_{|V|-1}\}} \sum_{j \in \{v_0,\ldots,v_{|V|-1}\}} w(e_{ij}), s.t. \lambda_G(i) \neq \lambda_G(j) \tag{4}$$

The final metric that we consider is the **evenness(λ_G^{σ})** of the partitions. In a scenario in which the underlying architecture consists of homogeneous execution units(i.e. a vertex of the road network graph takes equally long to simulate on any machine in the underlying architecture) and where communication plays a less significant role(i.e. powerful communication backbone), this metric plays a very important role as it defines how varied each machine is loaded.

$$\lambda_G^{\mu} = \frac{\sum_{P_i \in \{P_0,\ldots,P_{p-1}\}} w(P_i)}{p} \tag{5}$$

$$\lambda_G^{\sigma} = \frac{\sqrt{\sum_{P_i \in \{P_0,\ldots,P_{p-1}\}} (w(P_i) - \lambda_G^{\mu})^2}}{p} \tag{6}$$

The importance of these metrics vary hugely depending on the underlying architecture onto which these partitions are mapped onto and hence we present the metrics without probing into their relative importance. In the next section, we present an overview of the methods that are compared under the purview of road network partitioning.

4 META HEURISTICS BASED GRAPH PARTITIONING ALGORITHMS

4.1 Simulated Annealing

Input: Initial Mapping λ_0 and Starting and Final Temperatures $\mathscr{T}_0$, $\mathscr{T}_f$
Output: Best Mapping λ_{best}
$\lambda_{current} \leftarrow \lambda_0$;
$C_{current} \leftarrow objective\_function(\lambda_0)$; //calculate initial objective function value;
$\lambda_{best} \leftarrow \lambda_{current}$;
$C_{best} \leftarrow C_{current}$;
for $i \leftarrow 0$ *to* ∞ **do**
  $\mathscr{T}_{current} \leftarrow next\_temperature(\mathscr{T}_0, i)$;
  $\lambda_{new} \leftarrow move(\lambda_{current}, \mathscr{T})$;
  $C_{new} \leftarrow objective\_function(\lambda_{new})$;
  $\Delta C \leftarrow C_{new} - C_{current}$;
  $p \leftarrow acceptance\_probability(\Delta C, \mathscr{T}_{current})$;
  if $\Delta C < 0$ *or* $r < p$ **then**
    if $C_{new} < C_{best}$ **then**
      $\lambda_{best} \leftarrow \lambda_{new}$; $C_{best} \leftarrow C_{new}$; $\lambda_{current} \leftarrow \lambda_{new}$; $C_{current} \leftarrow C_{new}$;
    end
  end
  else
    if $\mathscr{T}_{current} \leq \mathscr{T}_f \| executionTime \geq maxTime$ **then**
      break
    end
  end
end
return λ_{best}

Algorithm 1: The Simulated Annealing Algorithm

Simulated Annealing (Kirkpatrick, Vecchi, et al. 1983) is an adaptation of the Metropolis-Hastings algorithm for solving the problem of locating a good approximation of the global optimum of a given function, $\mathscr{F} : \mathbb{R} \rightarrow \mathbb{R}$, which has a large search space. This large number of states, as discussed in Section 3.2, makes exhaustive enumeration to find optimal solutions, not feasible.

SA is a heuristic algorithm that explores the search space by inspecting one valid state at each iteration. Each of these inspected states are evaluated by an "objective function" which tells us how "good" or "bad" this state is. The "goodness" in an SA algorithm is problem dependent and in our case it is given by the metrics defined in Section 3.3. The algorithm progresses by inspecting a candidate state at each iteration and it either accepts it as its current state or discards the state and "moves" on to another state. We define a move as the generation of the next candidate states and this progress is governed by a global time-varying parameter called the "temperature" which changes based on an "annealing schedule".

The algorithm always accepts a move to a better solution, i.e. a new state which has a "better" objective function value than the current state. When this value is worse however, the SA algorithm accepts this move with a certain "acceptance probability", that changes with the current temperature. When the temperature is high, the algorithm accepts moves to a worse solution with a higher probability; as the temperature reduces over time, this probability decreases as well.

Every Simulated Annealing method is characterized by the definition of its core functions and parameters. One such core function is the "move" function which defines how the next candidate state is generated. We formally define a this function as,

$$move(\lambda_G, \mathscr{T}) = \lambda_G^{'} \tag{7}$$

where $\lambda_G^{'}$ is a new partitioning scheme which also maps each intersection to exactly one partition. The move function takes as input the current partitioning scheme λ_G and the current temperature and describes how the generation of $\lambda_G^{'}$ is done. In this paper we present a few variations of the SA algorithm, in terms of the "move" as follows.

4.1.1 Standard Move

Conventional wisdom (Orsila, Kangas, Salminen, and Hamalainen 2006) suggests that the move function has to be random to avoid local minimums. Hence, our baseline is this conventionally accepted notion of moving about the search space through random neighbours. For each intersection in the road network graph, we choose a random partition to assign it to as follows,

$$\lambda_G^{'}(v_i) = rand(0, p-1), \forall v_i \in V \tag{8}$$

The results of this method is labelled as **SA-standard** in the graphs in the results section.

4.1.2 Edge Labelling

As mentioned previously in section 3.3, the execution time of a distributed simulation depends on the amount of inter-partition communication. To address this, we label edges as **heavy** or **light** by virtue of their edge weights as compared to the "rest of the edges' weights". Given a subset of edges $E_S, S \subseteq G$ we define the following metrices that we will use in our two edge labelling methods.

$$E_S^{\mu} = \frac{\sum\limits_{e_i \in \{E_S\}} w(e_i)}{|E_S|} \tag{9}$$

$$E_S^{\sigma} = \frac{\sqrt{\sum\limits_{P_i \in \{P_0, \ldots, P_{p-1}\}} (w(P_i) - E_S^{\mu})^2}}{p} \tag{10}$$

Calculate E_S^{μ} and E_S^{σ} for the given E_S
for $e_i \in E_S$ **do**
  if $w(e_i) - E_S^{\mu} \geq k * E_S^{\sigma}$ **then**
    label(e_i) = heavy
  end
  else
    label(e_i) = light
  end
end

Algorithm 2: Edge labelling

For every edge in the edge set, we check if it is "k" sigmas away from the mean. If it is, then we label it as a heavy edge and do not allow it cross across partitions. For instance, if the edge weight of edge e_{ij}(between vertices v_i and v_j) is "k" sigmas away from the mean E_S^{μ}, then we move vertex j to the partition to which i belongs to or more formally we denote it by $\lambda_G^{'}(v_j) = \lambda_G{'}(v_i)$ thereby minimizing the inter-partition communication. Since this edge labelling algorithm works for any subset $S \in G$, we define the following two flavours of the move function.

- **Global Edge Labelling** - In this method, we set the subset S to be the entire graph G itself. This gives an idea of how varied the edges are in a global setting and helps to identify the critical pieces of roads that contain a lot of traffic flowing through them. The results of this method is labelled as **SA-stats** in the graphs in the results section.
- **Local Edge Labelling** - In contrast to the global edge labelling method, we define S as the subset of vertices from the graph that are connected to some vertex v_i. This identifies what the most critical edges are for any given intersection and we perturb the current solution only on vertex at a time. The results of this method is labelled as **SA-all-optimizations** in the graphs in the results section.

4.2 Genetic Algorithm

Genetic Algorithm is a heuristic search{cite core paper} algorithm that mimics the natural selection process. This heuristic algorithm is often used{cite usage in general context} for solving optimization problems with large search spaces and it belongs to a larger class called Evolutionary Algorithms.

Choose an initial random population of individuals
Evaluate the fitness of the individuals
while *Termination criteria not met* **do**
  Select the *best* "n" individuals to be used by the genetic operators
  Generate new offsprings by using the genetic operators
  Evaluate the objective function value for these offsprings
  Replace the *worst* "k" individuals of the current population with the best "k" individuals from the offsprings
end

Algorithm 3: The Genetic Algorithm

In the context of the problem at hand, we follow the definition for the search space from Section 3.2, as a 2-dimensional space with one axis representing the intersections and the other the partition IDs. Hence each point in the space becomes a $\{intersection,partition\}$ pair. Under this definition of the search space, to characterize a genetic algorithm, we require two things :

- A **genetic representation** of the solution space as shown in Figure **??**
- An **objective function** to evaluate solutions as discussed in Equation **??** from Section 3.3

Once this is fixed, we initialize the algorithm by generating a set of random solution vectors and assigning that as the initial population. The initial population size is problem specific and in our case it is 400 which is sufficiently a large sample and small enough to ensure fast generation of offsprings. We have chosen to generate the initial population randomly as opposed to generating it with some seed, to cover a wider range of candidates from our large search space. We then apply a set of genetic operators as discussed in Section 4.2.1 to generate the next population. This process is repeated and the "n" best solutions are retained at every step. We terminate the algorithm once a fixed time has passed by. Algorithm 3 gives an algorithmic overview of the above mentioned process.

4.2.1 Genetic Operators

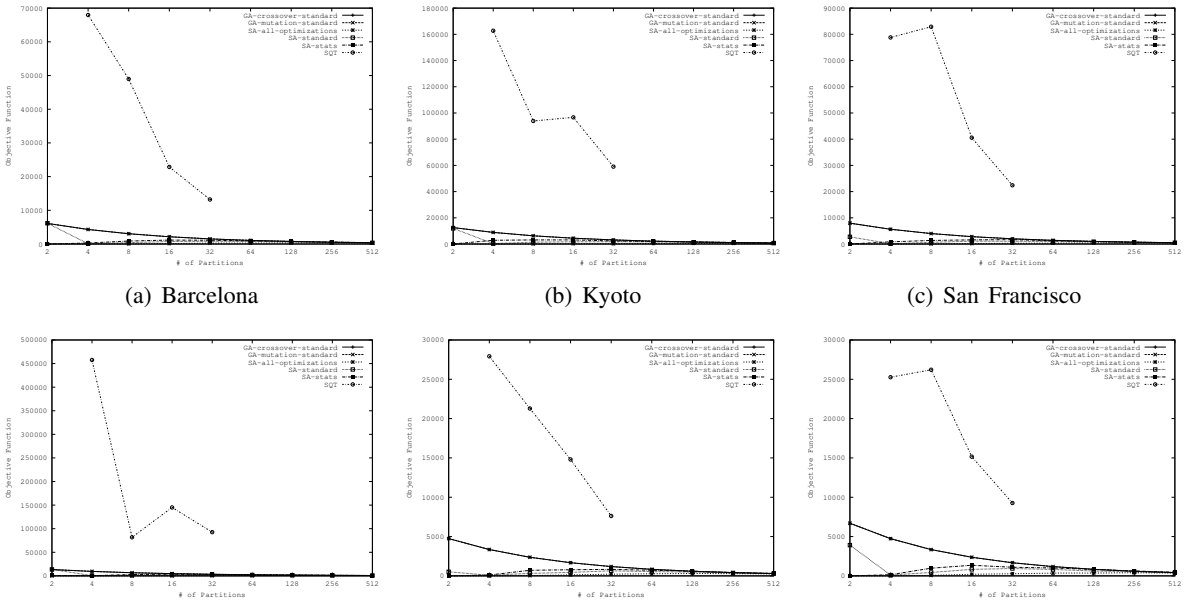
Although there are a lot of genetic operators to choose from{cite core paper and some new papers}, we employ the **mutation** and **crossover** operators for the purpose of this comparison

- **mutation** -
- **crossover** -

5 EXPERIMENTS AND RESULTS

	Applications					
	Barcelona	Kyoto	San Francisco	Cologne	Lyon	Miami
$\|V\|$	13476	42456	15436	56548	8174	8141
$\|E\|$	25658	93722	35092	115483	15586	21856

Table 1: The task graph setup



(a) Barcelona (b) Kyoto (c) San Francisco

(d) Cologne (e) Lyon (f) Miami

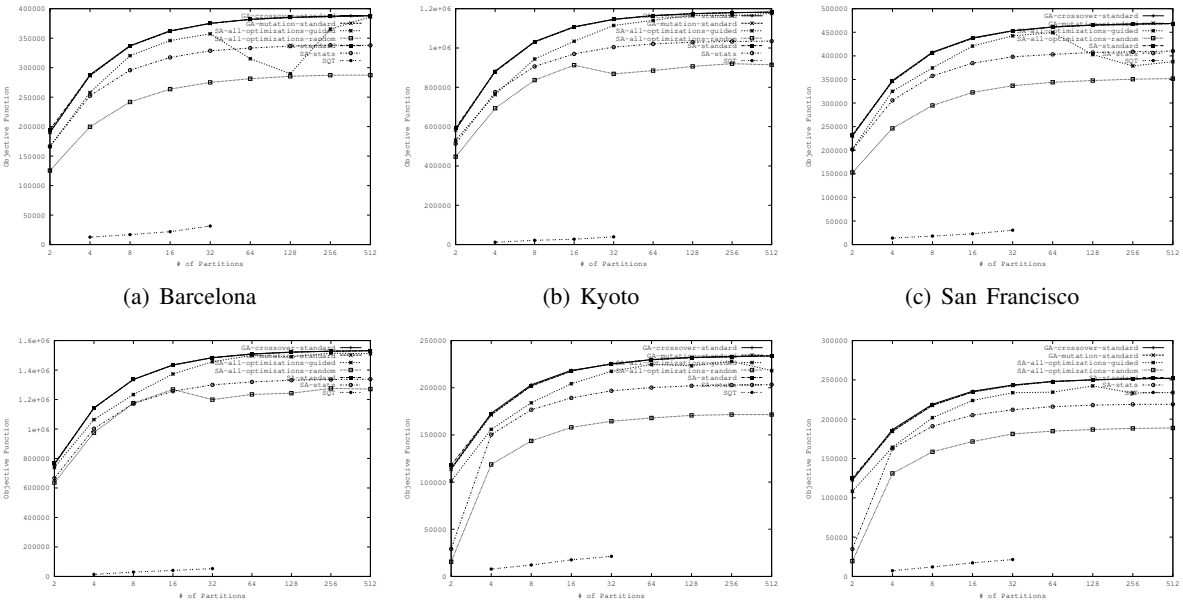
Figure 1: Comparison of the partioning algorithms based on the first metric(Variance) given by Eq. 6

6 FUTURE WORK

- One of the limiting factors in this comparison is the limited representation of the road network. We plan to extend our model by allowing edges and vertices in our road network graph to have more than one weight associated with them. In doing so, we can allow for a finer and more accurate representation of the road network which in turn allows us to partition the graph better.
- In the modified move function, we currently move edges(the neighbour correspoding to the edge) if it has a weight which is 2σ away from μ. This is under the assumption that traffic is normally distributed across the edges. One of the improvements that we could do is to fit the distribution of traffic across edges and move edges according to this regressively found curve that fits the traffic distribution.

7 CONCLUSION

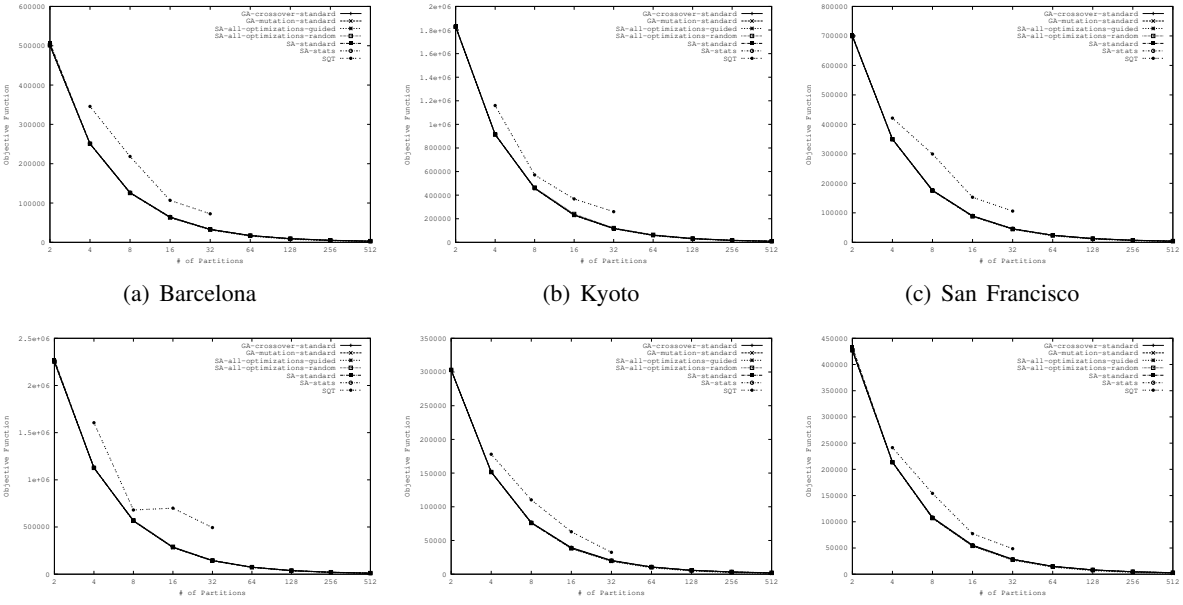
In this paper we have outlined the comparison of space and graph partitioning techniques in the context of partitioning road network graphs for simulation. We have discussed



(a) Barcelona (b) Kyoto (c) San Francisco

(d) Cologne (e) Lyon (f) Miami

Figure 2: Comparison of the partitioning algorithms based on the second metric(Inter-Partition Communication) given by Eq. 4



(a) Barcelona (b) Kyoto (c) San Francisco

(d) Cologne (e) Lyon (f) Miami

Figure 3: Comparison of the partitioning algorithms based on the third metric(Max(PartitionSizes)) given by Eq. 3

ACKNOWLEDGMENTS

This work is partly funded by the IRC Enterprise Partnership Scheme in collaboration with IBM Research, Dublin, Ireland.

A APPENDICES

Place any appendices after the acknowledgments and label them **A**, **B**, **C**, and so forth.

REFERENCES

Holden, H., and N. H. Risebro. 1995. "A mathematical model of traffic flow on a network of unidirectional roads". *SIAM Journal on Mathematical Analysis* 26 (4): 999–1017.

Kirkpatrick, S., M. Vecchi et al. 1983. "Optimization by simmulated annealing". *science* 220 (4598): 671–680.

Orsila, H., T. Kangas, E. Salminen, and T. D. Hamalainen. 2006. "Parameterizing simulated annealing for distributing task graphs on multiprocessor SoCs". In *System-on-Chip, 2006. International Symposium on*, 1–4. IEEE.

AUTHOR BIOGRAPHIES

ARAVIND VASUDEVAN
QUENTIN BRAGARD
ANTHONY VENTRESQUE
DAVID GREGG.