Scaling Up Transit Priority Modelling Using High-Throughput Computing

Mahmoud Mesbah
School of Civil Engineering, University of Queensland, Australia
Mahmoud.Mesbah@uq.edu.au

Majid Sarvi
Dept. of Civil Engineering, Monash University, Australia
Majid.Sarvi@monash.edu

Jefferson Tan
Faculty of I.T., Monash University, Australia
Jefferson.Tan@monash.edu

Fateme Karimirad
Dept. of Mechanical and Aerospace Engineering, Monash University, Australia
Fatemeh.Karimirad@monash.edu

Abstract

The optimization of Road Space Allocation (RSA) from a network perspective is computationally challenging. An analogue to the Network Design Problem (NDP), RSA can be classified NP-hard. In large-scale networks when the number of alternatives increases exponentially, there is a need for an efficient method to reduce the number of alternatives while keeping computer execution time of the analysis at practical levels. A heuristic based on genetic algorithms (GAs) is proposed to efficiently select Transit Priority Alternatives (TPAs). The proposed framework allows for a TPA to be analysed by a commercial package that is a significant provision for large-scale networks in practice. We explore alterative parallel processing techniques to reduce execution time: multithreading and High-Throughput Computing (HTC). Speedup and efficiency are compared with that of traditional sequential GA, and we discuss both advantages and limitations. We find that multithreading is better when using the same number of processors, but HTC provides expandability.

Keywords: transport modelling, genetic algorithm, high-throughput computing, high-performance computing

1 Introduction

With ever-increasing travel demands, traffic congestion has become a challenge for many cities around the world. Construction of new roads or mass transit is not always possible, and reallocation of road space between transit vehicles and cars has emerged as a solution. Mesbah et al. (2011a, 2011b) proposed a bi-level optimization program for road space allocation (RSA). The objective was to identify the roads on which a bus lane should be introduced. The authors showed that this Mixed Integer Non-Linear (MINL) formulation is an NP-hard problem and is therefore computationally challenging. For large-scale networks, a heuristic approach is adapted to find reasonable solutions. This problem can be classified under the umbrella of Network Design Problems (NDP) that has a wide range of applications in Engineering. The network can be for roads, communication, power, water, or any network with a set of connected nodes and links. The goal is to find the optimal combination of links to be added/modified to minimize a certain objective function.

The RSA problem is NP-hard, so the proposed optimization methods to large-scale problems requires extensive computational power, feasible with advanced techniques such as *High-Performance Computing* (HPC) (Strohmaier et al., 2005). While the term was applied broadly at first (Dongarra et al., 2005), HPC today typically applies to a tightly coupled system of many shared memory processors, particularly important when jobs must communicate among themselves. An alternative is *High-Throughput Computing* (HTC), aimed at providing large amounts of processing capacity taken together over a long period of time (Thain et al, 2005). *Many Task Computing* bridges the gap between HPC and HTC (Raicu et al., 2010), whether or not there are many long duration tasks, and regardless of the number of processors per computer. The common goal is to support simultaneous computations, where a long process is divided into small tasks, which are distributed across a set of interconnected processors to execute separately, simultaneously. Results are then gathered and combined. While HPC taken broadly may apply, the work described in this paper focuses on the HTC approach to distinguish the use of several independent computers on a network, as against our previous work using a single multiprocessor (Mesbah et al., 2011a). We demonstrate the application of HTC to solve a large-scale optimization problem in Transportation Engineering.

The proposed RSA is formulated as bi-level optimization. The upper level formulates an objective function and a set of constraints from the system managers' perspective. The lower level consists of user behavioural models, which requires a complex optimization program on its own. A number of commercial packages are available in order to analyse the user behaviour at the lower level, one of which is employed in this research. Many transport networks are already modelled in commercial packages, so there are benefits to sticking with them. Transport authorities have invested heavily in developing these models and already have confidence in their performance. Moreover, many transport planners are already trained to work with them. However, there are certain challenges in dealing with commercial applications such as we have had to do. We use a package called **Visum**. It requires Microsoft Windows, and uses a dongle for license management. The installer is more than 700MB, and requires interactive installation. While it can use multithreading on a machine with many processors (cores) and lots of memory, the

cost of such a machine can be prohibitive, and there are physical upper limits on cores and memory on any given machine. On the other hand, like other packages, it is not designed for HTC environments. Apart from a cluster, HTC can also be through a *computational grid*. This is an extensible aggregation of computational *resources*, such as clusters, belonging to independent organizations (Foster et al., 2001). Grids traditionally consist of Linux resources, while many engineering applications run on the Windows platform. Grids commonly support non-commercial applications with standard libraries provided almost out of the box, so a distributed execution of such applications is normally straightforward. RSA computations speed up if workload is distributed across such environments, but the nature of grids conflicts with the conditions for commercially licensed software. Licenses are typically limited to individual organizations while grids span across a *virtual organization* (VO) of several member organizations that remain autonomous. One cannot install or execute on just any resource, and such resources are normally not uniform anyway. We therefore have these three interesting challenges:

1. We use Visum, which requires Windows.
2. This is a commercial package and the source code is not accessible for reprogramming.
3. It must be pre-installed on each compute node with a large installer of over 700MB.

The proposed method can apply to many engineering applications where an iterative procedure is carried out using a commercial software package. A point we wish to make is that, despite the challenges, HTC can make many engineering applications scalable for large problems, even where the long runtime used to be a limiting factor.

The next section starts with a limited literature review on transit priority and continues with the bi-level optimization formulation. Then a solution algorithm is presented, based on a *genetic algorithm* (GA). It is implemented for (1) a single CPU on one machine, (2) multiple CPUs on one machine, and (3) multiple CPUs on multiple machines. Details are discussed subsequently, as is an example. In the last section, the results are discussed and the major findings are summarized.

2 Research Background

2.1 Road Space Allocation

The introduction of exclusive lanes to transit vehicles is one way to prioritize transit, an approach known as Road Space Allocation (RSA) (Black 1991, Currie et al., 2004). The literature on RSA can be classified into evaluation studies and optimization studies (see Figure 1).

Some evaluation studies focus on the local level, i.e. a link or corridor, e.g., Black (1991) presented a model on an urban corridor, evaluating several predefined scenarios based on total user travel time. Jepson and Ferreira (2000) assessed different road space priority treatments such as bus lane and setbacks based on delays in two consecutive links. Currie et al. (2007) considered a comprehensive list of impacts of RSA including travel time, travel time variability, initial and maintenance costs in a local priority project.

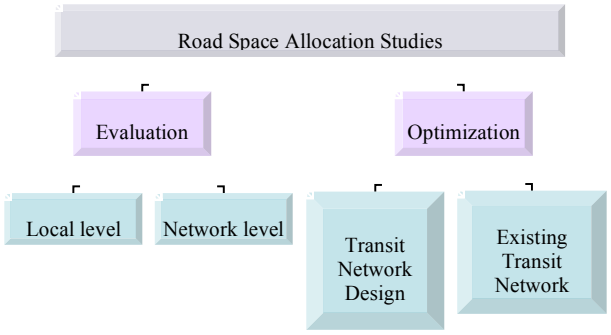

Figure 1. Classification of RSA studies.

Having compared performance measures in the literature, they proposed an approach to evaluate transit priority projects. Using the concept of intermittent bus lanes (Viegas 1996, Viegas and Lu 2004), Eichler and Daganzo (2006) suggested a new analysis method based on kinematic wave theory, which can be applied to a long arterial. At the network level, Bly et al. (1978) explored exclusive bus lanes to a link in different conditions, and the impact on the network was assessed using sensitivity analysis. Waterson et al. (2003) presented a macro-simulation approach which evaluates a given priority scenario at the network level. This approach considered rerouting, retiming, modal change, and trip suppression. Liu et al. (2006) proposed a similar approach with micro-simulation. Stirzaker and Dia (2007) applied micro-simulation to evaluate a major bus lane project in Brisbane. These studies evaluated a limited number of alternatives that do not necessarily include the best possible RSA over the network, and do not propose an optimization method to find the best set of bus lanes.

A number of studies have approached the problem using combined RSA optimization in Transit Network Design Problem (TNDP). Duff-Riddle and Bester (2005) applied a *trip focusing* process to design transit routes. The iterative method was able to put transit routes on the shortest travel time and shortest distance. The issue of express buses was also included with minute changes in the model. Chen et al. (2007) presented a design method in the form of a mathematical programming model. However, similar to Duff-Riddle and Bester (2005), the aim of their method was to design a new bus route.

Having first explored optimal TPAs in an existing transit network (Mesbah et al., 2008) with a general framework to find the optimal TPA at the network level, we have since then introduced a decomposition approach and a GA approach (Mesbah et al., 2011a, 2011b). This paper extends our work by employing HTC to reduce the runtime for large-scale transit networks.

2.2 High-Throughput Computing

HPC is a broad umbrella for a number of different environments (Strohmaier et al., 2005), but when performance is measured for many tasks across long periods of time, we may speak of high-throughput computing (HTC) (Thain et al., 2005). A neutral term bridging HPC and HTC is *many task computing* (MTC), with little distinction about the size of tasks (Raicu et al., 2010). Commodity computers can also be organized on high-speed networks. They are relatively low expenditure resources, compared to supercomputing facilities.

Beowulf-class clusters were probably the first (Sterling et al., 1998) of such environments, providing a queuing system for submitting and managing computational jobs. Another environment is the **Sun Grid Engine** (Gentzsch, 2001), and there are others, which uniformly share a preference for the UNIX or Linux environment.

Condor (Thain et al., 2005) uses computers that are normally used for other purposes, e.g., a desktop, and supports Windows nodes. Condor was originally dubbed "hunter for idle workstations" (Litzkow et al., 1988), i.e., when the user leaves the console for extended periods, e.g., after hours. This is the case for Monash University's SPONGE resource, with up to 1000 cores running on computer laboratories across campuses during lean periods and after hours. While most nodes have two cores with modest memory, SPONGE collectively provides a considerable HTC resource.

3 Transit Priority Optimization

The RSA problem can be modelled as a 'Stackelberg competition' in which the system manager is the leader and transport users are followers (Simaan 1977, Bard and Falks 1982, Yang and Bell 1998, Liu et al., 2008). The system manager chooses a TPA, and in the subsequent system, users would choose their mode of travel and a path in order to maximize their own benefit.

The above design approach is formulated in this paper as a bi-level optimization program (Shimizu et al., 1997, Bard, 1998) (see Figure 2). At the upper level are the objective function and constraints from the system manager perspective. The upper level determines the TPA or the links on which priority would be provided for transit vehicles (decision variables). The aim of the upper level is to achieve System Optimal (SO) (Sheffi, 1984), thus the objective function includes a combination of network performance measures. The corresponding constraints are included in the upper level constraints.

The upper level can be formulated as follows:

$$MinZ = \alpha \sum_{a \in A} x_a^c t_a^c(x) + \beta \left(\sum_{a \in B} x_a^b t_a^b(x) + \sum_{i \in I} w_i^b \right) + \gamma \sum_{a \in A} \frac{x_a^c}{Occ^c} l_a Imp^c + \eta \sum_{a \in B} f_a s_a Imp^b \quad (1)$$

s.t.,

$$\sum_{a \in A_2} Exc_a \phi_a \le Bdg \quad (2)$$

$$\phi_a = 0 \text{ or } 1 \quad \forall a \in A_2 \quad (3)$$

Variable definitions can be found in the annotation section. Note that $f_a = \sum_{p \in L} f_p \xi_{p,a}$, where $\xi_{p,a}$ is an element of the bus line-link incident matrix with $\xi_{p,a}=1$ if bus line p travels on link a and $\xi_{p,a}=0$ otherwise. The in-vehicle travel time is $t_a^b(x)$.

The first two terms in the objective function are the total travel time by car and bus. The next two terms represent the various other impacts of these two modes including emission, noise, accident, and reliability of travel time. The factors α, β, γ, and η not only convert the units, but also enable the formulation to attribute different relative weights to the components of the objective function (Mesbah et al., 2010). Equation (2) states that the cost of the implementation should be less than or equal to the budget. The decision variable is ϕ_a by which the system managers try to minimize their objective function (Z). If $\phi_a=1$, then a bus lane is introduced on link a and buses can speed up to free flow speed, while the capacity of the link for cars is reduced from $Cpc_{0,a}^c$ to $Cpc_{1,a}^c$. If $\phi_a=0$, then buses will travel in the mixed traffic on a link with a capacity of $Cpc_{0,a}^c$. They are users who determine the link flows (x). Link flows are related to the decision variables by the lower level models.

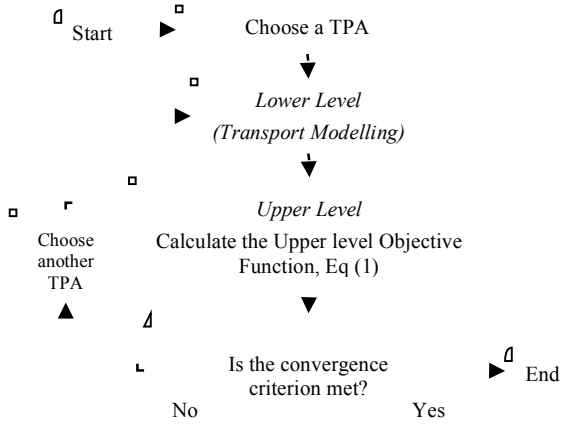

Figure 2. Outline of the proposed methodology.

At the lower level, it is the users' turn to maximize their benefit. Based on the decision variables determined at the upper level, users make their trips. The traditional four-step method (Ortúzar and Willumsen, 2001) is adapted in this paper for transport modelling. It is assumed that the travel demand and the distribution of demand are not affected by the location of bus lanes (these conditions can be relaxed in future studies). Therefore, the origin-destination matrix remains constant. The lower level consists of three models: (1) modal split model, (2) traffic assignment model (car demand), and (3) transit assignment model (bus demand). Once the demand is determined, users choose their travel mode. Then, the car demand segment of the total demand is assigned to the network. The last step at the lower level formulation is the assignment of transit demand. Without loss of generality, in this study, a Logit model is used for the mode choice (Papacostas and Prevedouros, 1993), a User Equilibrium (UE) model is adapted for traffic assignment (Sheffi, 1984), and frequency-based assignment is applied to transit assignment (PTV AG, 2009). While these models are used for mode choice and assignment steps, the proposed HTC framework can be implemented by many other transport planning models. The lower level calculations are performed in Visum (PTV AG, 2009). As previously stated, many cities already use commercial packages. The proposed framework incorporates them instead of having to convert the models to other formats.

The bi-level structure, with a linear objective function and constraints, is NP-hard (Ben-Ayed and Blair, 1990). To complicate things further, the upper level objective function and the UE traffic assignment are non-linear. We employ a GA to find an approximate solution. The output

of the model is the combination of transit exclusive lanes which minimizes the proposed objective function.

4 The Genetic Algorithm Solution

A Genetic Algorithm (GA) is an iterative search method in which new answers are produced by combining two predecessor answers (Russell and Norvig 2003). Inspired from evolutionary theory in nature, the GA starts with a set of answers referred to as the *population*. Each individual answer in the population, a *chromosome*, is assigned a survival probability, based on the value of the objective function. The algorithm *selects* individual chromosomes based on this probability to breed the next generation of the population. GA uses *crossover* and *mutation* operators to breed the next generation, which replaces the predecessor generation. The algorithm is repeated with the new generation until a convergence criterion is satisfied. A number of studies applied GA to transit networks. Two recent examples are a transit network design problem considering variable demand (Fan and Machemehl, 2006) and minimization of transfer time by shifting time tables (Cevallos and Zhao, 2006).

In applying GA to the RSA problem, we define a gene to represent the binary variable ϕ_a, and a chromosome is the vector of genes (ϕ) which represents a TPA. A chromosome (TPA) contains a combination of links on which an exclusive lane may be introduced (set A_2). Therefore, the length of the chromosome is equal to the size of A_2. The algorithm starts with an initial population with n chromosomes. The chromosomes of the initial population are produced randomly. When an initial chromosome population is produced, they are evaluated using the lower level models, i.e. the transport planning models of mode split, traffic assignment, and transit assignment. This evaluation is the time consuming component in the GA. Using the flow and travel time from the lower level, the values of the upper level objective function (Z) for all chromosomes are determined. Once the evaluated, the chromosomes are ranked from the lowest Z value to the highest. The fitness function, which determines the probability of a chromosome selection for breeding, is assumed to be an arithmetic series with the highest probability assigned to the top chromosome. The probability of the top ranked chromosome is assumed to be $P(1) = a_0 + 1/n$ where a_0 is a constant and n is the population size. Subsequently, other terms can be calculated using $P(i>1) = P(1) - \gamma \times i$ where γ is the reduction factor so that $\sum_{i=1}^{n} P(i) = 1$.

$$\sum_{i=1}^{n} P(i) = P(1) + \sum_{i=2}^{n} (P(1) - \gamma \times i) = 1$$

$$\Rightarrow \gamma = \frac{n \times P(1) - 1}{n-1} = \frac{n \times a_0}{n-1}$$

A one point crossover is used in all experiments. The mutation involves flipping the value of a gene from 0 to 1 or vice versa. When a chromosome is selected for mutation, one gene from each set of 5 to 8 genes are flipped. That is about 12 to 20 flips for a chromosome 100 genes long. A common convergence criterion adapted here is to terminate if the number of iterations exceeds a predetermined value (*maxg*) or if the best objective function value found remains constant for a number of generations (m). The process above is summarized in this algorithm:

0. Initialization: Set iteration number (n) to 1, best solution value or *upper bound* (UBD) to ∞. Set max generations (*maxg*), and number of generations with same UBD, m.
1. Generate initial population.
2. Evaluation: Calculate the objective function value for all chromosomes (or TPAs) in the population, using the transport planning models at the lower level.
3. Fitness: Determine survival probabilities (fitness) and update UBD.
4. Convergence: If $n>maxg$ or UBD is constant for m generations, then stop.
5. Reproduction: Breed a new generation by performing selection, crossover, and mutation. Go to Step 2.

5 Implementation of the Genetic Algorithm

The most computationally intensive part of the GA is Step 2 where TPAs are evaluated. One evaluation involves running the four-step modelling for a network, which may take as long as a few hours on a typical desktop. Furthermore, the GA requires a large number of TPA evaluations, depending on the number of decision variables and attributes, e.g., probabilities of crossover and mutation. At this point, we decompose the processes in order to execute them in distributed fashion. This approach significantly reduces execution time.

The steps of Genetic algorithm in terms of dependency of processes are of two types. First is the evaluation step (Step 2). The evaluation of an individual chromosome (or TPA) is independent of other chromosomes (or TPAs) in a generation, which gives us a number of processes that can be executed independently. The second part of the GA involves fitness, convergence, and reproduction (Steps 3 to 5). These steps integrate the individual evaluations of Step 2 where the processes are interdependent. On the basis of the dependency attribute, two variants of the GA are proposed in the literature (Haupt et al., 2004, Goldberg, 2002, Cantú-Paz, 2000): *serial* (SGA) and *parallel* (PGA). Figure 3 illustrates these two variants. In SGA, all processes are carried out in a sequence, which means that, in Step 2, evaluation of a chromosome is completed before the evaluation of another chromosome is started. Then Steps 3, 4, and 5 are completed to produce another generation and then we cycle back to Step 2 (Figure 3 (a)). However, in PGA, evaluations are performed simultaneously. Therefore, Step 2 is executed in parallel, which is then followed by Steps 3, 4, and 5 in a sequence (see Figure 3(b)). SGA is simpler to implement, and details are explained in the next section. For PGA, we use two techniques of implementation: multithreading with multiple cores on one machine or HTC over several machines in a network.

5.1 Parallel GA - Multithreading (MT)

An operating system (OS) creates *threads* to run software. To run multiple applications simultaneously, multiple threads can be processed at a time, i.e.,

multithreading, if the machine supports multiple cores (Akhter and Roberts 2006, Evjen, 2004). To implement PGA by multithreading, the architecture of Figure 3(b) is used. The number of threads is selected equal to the number of processing cores on a machine (say p) plus a main thread. The main thread is reserved to control the flow of the GA from the start to the end. The main thread performs the fitness, convergence, and reproduction steps. The remaining p threads are used to execute TPA evaluations (objective function). When a generation is produced (see Figure 3(b)), n TPAs are queued for evaluation. The first p jobs in the queue are assigned p available threads. Once these p TPAs are evaluated, the next p TPAs are assigned. The next generation is produced when all TPAs are evaluated.

The speedup achieved depends on the number of cores on a machine and the efficiency of the OS in supporting multithreading. We implemented multithreading in Windows since the TPAs are evaluated by Visum, which requires Windows. The latter is commonly criticized for its performance, but there will always be cases where performance declines when the number of threads exceeds the number of cores (Akhter and Roberts, 2006), regardless of the OS. In that case, the OS must time-share the limited cores among so many executing threads, and we incur "time slicing" overhead. Moreover, the maximum number of cores that can go into one machine is subject to space and temperature constraints. There can also be a limit to gains due to memory latency and cache conflicts (Athanasaki et al., 2008). There is thus a cap on the speedup in multithreading, and the cost of purchasing many cores and supporting hardware can be high.

However, with TPA evaluations performed with commercial software, multithreading saves considerably on *license* costs for some packages. For example, one Visum license is sufficient for one multithreading execution of the entire model on one machine, but performance will be constrained to what that machine can deliver. The next section discusses our HTC approach to avoid some of the limits of multithreading, although it requires multiple licenses. Our implementations are in Visual Basic .NET environment in this study.

A distributed computing approach such as HTC schedules TPA evaluations to several nodes on a network, each node having its own set of cores and local memory. Therefore, there is less of a limit on the number of tasks that can be executed simultaneously, as the number of computers in a network is not so tightly bounded. The trade-off is the complexity of distributing the task to available computers in the network, manage the queue, data transfers, provide an inter-process message-passing system in some cases, then collect and integrate the results.

5.2 Parallel GA –HTC with Condor

In Figure 3(b), p out of n evaluations in a population can be run in parallel. The ideal case is when p is equal to n, which means all n evaluations are done at the same time. However, as mentioned earlier, the number of threads supported on a given machine is limited. There are a number of existing systems for these, such as Condor. It was originally developed to use computers during idle periods (Thain et al., 2005), but is now one of the most flexible and powerful HTC platforms. Computers participate within a Condor *pool*. Owners can configure nodes to donate only some of their time. For example, as in the case of Monash University, the SPONGE pool consists of nodes that run from computer labs.

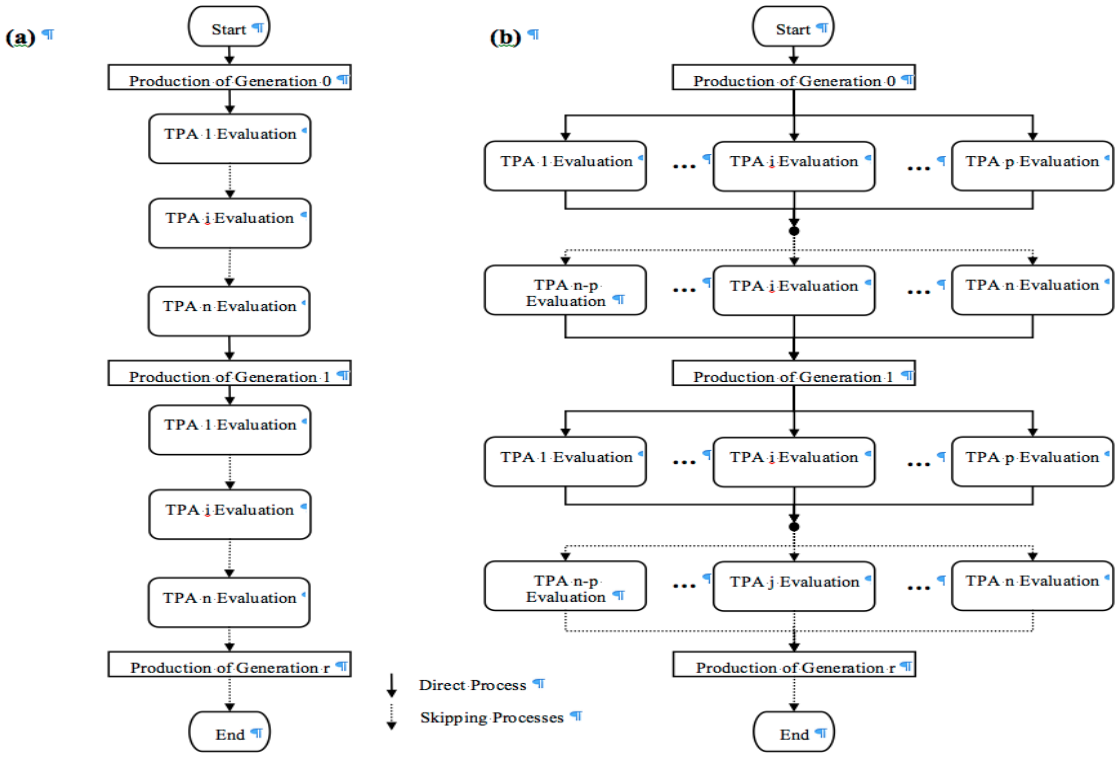

Figure 3. Sequence of components in serial genetic algorithm and parallel genetic algorithm.

They are only used when no one is currently using the desktop. These lab nodes are all running Windows XP or Windows 7, which works for us since our TPA evaluations are performed by Windows-based software. While issues emerging in adopting a general tool (HPC/HTC) tool to the RSA problem are tool-specific, important lessons can be learned. A license server restricts simultaneous runs of Visum with a hardware. The license server can run anywhere on the network, and need not be in the Condor pool. If *x* computers are in the network and *y* licenses are available, the maximum number of parallel TPA evaluations is *p=min(x, y)*.

The parallel scheduling used in this HTC approach is to queue *n* TPA evaluations (the jobs) when a generation is produced (see Figure 3(b)). The jobs are assigned to the first set of available nodes. For instance, if $p<n$ nodes are available, *p* jobs are assigned and the remaining *n-p* will wait in the queue. As soon as a job finishes on one node, the next queued job is assigned to that node. The next generation is produced when all TPAs are evaluated.

To evaluate the TPAs, a user submits jobs from the submission machine. For each job, Condor will copy input files and the objective evaluation program to the worker node and execute the program. Once completed, output data are copied back will be downloaded back to the submission machine. Some applications can be launched as a self-contained package, but Visum is not in that category. It requires interactive installation, with a 700-MB installer, which would require a considerable amount of time to copy to an execution node, even on a fast network. The solution was pre-installation of Visum on a subset of Sponge, where the owners were willing. Condor's scheduler must be told, upon submission, to send jobs only to nodes with Visum installed. This can be effected with Condor's *ClassAd* mechanism using custom ClassAd attributes, but in our implementation, we instead identified specific Visum-installed machines by name.

Windows differentiates between the local or remote launch of an application. Windows also consults the user permissions to run an application either locally or remotely. A *COM server* was configured to grant suitable permissions to launch Condor jobs from a remote user.

6 Numerical Example

Three GA implementations (SGA, PGA-MT, and PGA-HTC) are applied to an example transit network, the layout of which is in Figure 4. This grid network consists of 86 nodes and 306 links. All circumferential nodes together with Centroid 22, 26, 43, 45, 62, and 66 are origin and destination nodes. A 'flat' demand matrix of 30 persons/hr is traveling from all origins to all destinations. The total demand for all the 36 origins and destinations is 37,800 persons/hr. There are 10 bus lines covering transit demand in the network (see Figure 4). The frequency of service for the bus lines is 10 minutes. Parameters used are extracted from those calibrated for the Melbourne Integrated Transport Model (MITM), a four-step model used by the Victorian State Government for planning in Melbourne (Department of Infrastructure, 2004). Vertical and horizontal links are 400m long with two lanes in each direction and a speed limit of 36 km/hr. It is assumed that if an exclusive lane is introduced on a link on one direction, it may not necessarily be introduced in the opposite direction. There are 120 links (uni-directional) in the network on which an exclusive lane can be introduced. These links are highlighted in black solid line. The following Akcelik cost functions (Ortúzar and Willumsen, 2001) are assumed for links with an exclusive lane (Equation (4)) and without (Equation (5)).

$$t^c_{1,a} = t_{0,a} + \frac{3600a}{4}\left[\left(\frac{x^c_a}{Cap^c_{1,a}} - 1\right) + \sqrt{\left(\frac{x^c_a}{Cap^c_{1,a}} - 1\right)^2 + \frac{8b}{ad}\left(\frac{x^c_a}{Cap^c_{1,a}}\right)}\right], t^t_{1,b} = t_{0,a} \quad (4)$$

$$t^c_{0,a} = t^b_{0,a} = t_{0,a} + \frac{3600a}{4}\left[\left(\frac{x^c_a + x^t_a}{Cap^c_{0,a}} - 1\right) + \sqrt{\left(\frac{x^c_a + x^t_a}{Cap^c_{0,a}} - 1\right)^2 + \frac{8b}{ad}\left(\frac{x^c_a + x^t_a}{Cap^c_{0,a}}\right)}\right] \quad (5)$$

where t_0 determines travel time with free flow speed, *a* is length of observation period, *b* is a constant, *d* is lane capacity, and other terms are as in the Section 8. Each link has 2 lanes, and:

$$a = 1hr, b = 1.4, d = 800veh/hr$$

$$Cap^c_{0,a} = 1800veh/hr$$

$$Cap^c_{1,a} = 900veh/hr$$

Mode share is determined using a Logit model. Traffic User Equilibrium (UE) and a frequency-based assignment is employed to model traffic and transit assignments, respectively. All these lower level transport models are implemented using Visum (PTV AG, 2009). The upper level objective function includes total travel time and total vehicle distance. The absolute value of the objective function can therefore be very large. A constant value is subtracted from the objective function value for all evaluations. Hence, the objective function value is relative. The weighting factors of the objective function are assumed to be 0.01. Regarding constraints the budget is assumed to allow for all candidate links for the provision of bus priority. The GA includes many parameters to tune. We suggest a particular set of values as a guideline in this example. It was assumed that population size, crossover probability (*cp*), and mutation probability (*mp*) are 40, 0.98, 0.01, respectively. The example demonstrates the HTC speedup compared to the serial approach. Although selection of the GA parameters may vary the absolute value of the execution time, the time differences on a relative basis are useful indicators to highlight the efficiency of the HTC approach. Table 1 describes seven computers we used in terms of the number of CPUs, versions of Windows, and of Visum. It demonstrates HTC incorporating diverse types of computers and software. Note that some processors can support two simultaneous threads per core. The first machine listed has four cores but can support eight threads, and perform up to eight TPA evaluations at a time. If all computers were allocated, 32 evaluations can be carried out simultaneously, requiring 32 licenses. The last column in Table 1 is the time spent evaluating one TPA on each machine. Machine 1 took the least time at 65 seconds, and Machine 7 was the slowest at 226 seconds. SGA, PGA by multithreading (MT), and PGA by HTC are explored.

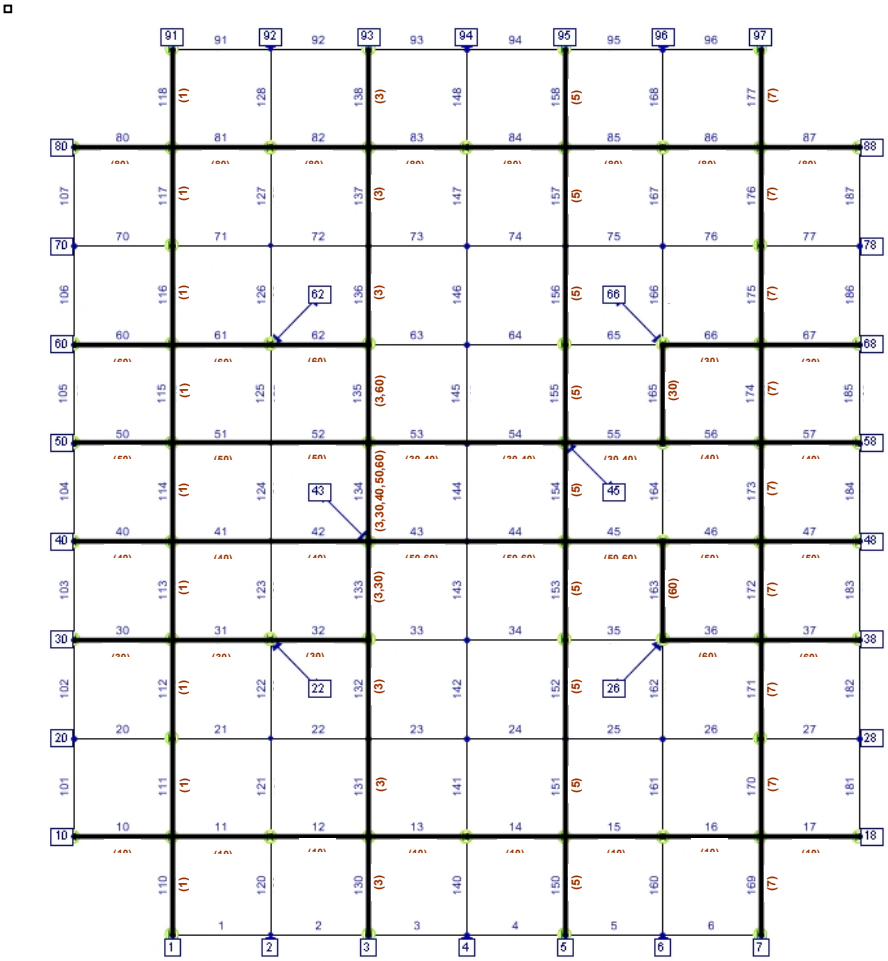

Figure 4. Example network with link numbers, origin destination nodes in boxes, and bus lines in parenthesis

Machine	CPU	Cores	Threads	Windows	Visum	Evaluation Time (s)
1	Intel Core i7 CPU 860 @ 2.8 GHz	4	8	7 64-bit	11.03 64-bit	65
2	Intel Core i7 CPU Q820 @ 1.73 GHz	4	8	7 64-bit	11.03 64-bit	147
3	Intel Core 2 Quad CPU Q6600 @ 2.4 GHz	4	4	7 64-bit	11.03 64-bit	101
4	Intel Core 2 Quad CPU Q6600 @ 2.4 GHz	4	4	XP 64-bit	11.01 32-bit	122
5	Intel Core 2 Quad CPU Q6600 @ 2.4 GHz	4	4	XP 64-bit	11.01 32-bit	121
6	Intel Core 2 Duo CPU E8500 @ 3.16 GHz	2	2	XP 64-bit	11.01 32-bit	88
7	Intel Pentium 4 CPU @ 3.2 GHz	2	2	XP 32-bit	11.01 32-bit	226

Table 1. Computers used in the experiments.

The base experiment (datum) for the MT approach is performed on Machine 4 with four threads, and for the HTC approach on Machines 1, 2, 3, and 6, with a total of 22 threads. The approach taken does not affect either the number of evaluations or the rate of improvement in the objective function. It does, however, affect the evaluation time. The minimum objective function value found in a run with 400 generations was *-4.757*.

The execution time of SGA is prohibitively long, being sequential. The number of generations was not carried past 300. All our four runs evaluated about 1700 TPAs each by the 50th generation. Although these runs do not follow exactly the same path in finding minimum, the trend shows that the value improves gradually at each successive evaluation. Figure 5 demonstrates the descent towards the minimum of the objective function value for two MT and two HTC runs. For comparison purposes, the SGA runs are also graphed. All approaches take the same downward trend to the minimum, but the implementation of the evaluation step results in different execution times. Three sets of experiments were organized with a population size of 40, crossover probability (*cp*) of 0.98, and mutation probabilities (*mp*) of 0.005, 0.01, and 0.02. The change in *mp* can change the number of evaluated TPAs. Figure 5 shows the quickest descent to the minimum of about 7.0 for HTC-1 and HTC-2 at about 100,000 seconds, with up to 32 simultaneous threads possible. MT-1 and MT-2 are not far behind at about 135,000 and 150,000 seconds, respectively, also to descend to a minimum of about 7.0. SGA runs went for much longer. For example, to reach a value of 30, SGA-3 takes about 170,000 seconds (two days) while HTC needs only 2,000 seconds. SGA-4 with 300 generations exceeded 5 days!

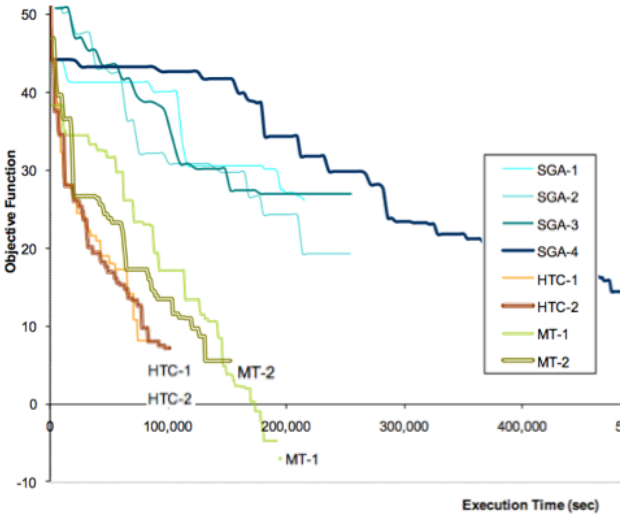

Figure 5. TPA evaluations in different modes.

Three measures were used in this study: (1) average time per evaluation (ATE), (2) speedup, which is the ratio of ATE in one run to the ATE of one SGA run, and (3) efficiency, which is the ratio of the speedup to the number of available threads. The speedup and efficiency of SGA runs are 1. Table 2 shows that ATE does not change significantly with *mp*. The number of cores are more significant, so the ATE for SGA, MT, and HTC runs are approximately 140, 40, and 14 seconds, respectively, where the number of cores are 1, 4 and 10, respectively. The efficiency measure demonstrates that in return for adding each thread in the MT approach, the execution time has improved by 80-90%. However, the efficiency in the HTC approach was just above 50% for the addition of each thread. There is considerable overhead incurred with distribution and queuing in HTC.

Table 3 presents the effects of the number of available threads. Experiment E228 has the lowest ATE at 11.7 seconds. There are some important results in Table 3. The ATE did not improve when the number of threads went from 22 to 26. Experiment logs reveal that about 1650 TPA evaluations are performed in each run, to an average of 33 evaluations per generation. Nevertheless, this is not uniformly distributed. The TPA evaluations are recorded to prevent evaluating a TPA twice. Therefore, while the average number of evaluations per generation is 33, the first generations evaluate close to 40 (which is the population size) evaluations, while the last generations evaluate just over 20 TPAs. When close to 40 TPAs are being evaluated, both experiments E228 and E230 may allocate two or less evaluations to a thread. This means about two evaluations run in sequence. Similarly, when just above 20 TPAs are being evaluated, both E228 and E230 have enough threads to run all evaluations simultaneously. Therefore, an increase of four threads does not improve execution time. Accordingly, the ATE in experiment E231 should be similar to E228 and E230, but it increases instead. We added a very slow Machine 7 to the pool. In the time it takes for it to evaluate one TPA, other machines can evaluate between two to four. Machine 7 holds up the other available threads, extending the evaluation time of each generation.

7 Conclusions and Future Work

We presented a solution to Road Space Allocation using serial GA, parallel GA with multithreading, and parallel GA with HTC. The optimum was found regardless of the GA variant, but performance varied. PGA-MT with four threads reduced execution time by 3.2 to 3.7 times compared to SGA, and PGA-HTC with 18 threads by 9.3 to 9.8 times. MT is more efficient, but challenging to use for large-scale, realistic networks since the number of threads on a computer is generally constrained. In contrast, there is practically no limit in the HTC approach via incremental expansion.

A novel outcome is the successful implementation of HTC with commercial software on Windows. However, the overhead of pre-installed commercial software like Visum cannot be taken for granted. There is considerable benefit in grid computing, but it is not so accommodating to commercial packages. A logical follow-up is to explore *cloud computing* (Foster et al., 2008) with standard or custom settings and applications on the cloud resources. The framework is generic enough to apply to the entire family of Network Design Problems (NDPs). Applying the framework to NDP problems in large-scale networks can be a challenge. Moreover, substitution and comparison of other heuristic methods with the GA could be another area of future studies.

8 Notations

A : Set of all links in the network, $A = A_1 \cup A_2$

A_1 : Set of links in the network where provision of priority is impossible,

A_2 : Set of links where the provision of priority (introducing exclusive lane) is possible,

B : Set of links with a bus line on them, walking links, and transfer links,

L : Set of bus lines,

f_a : Sum of frequency of service for bus lines on link a,

f_p : Frequency of service for bus line p,

l_a : Length of link a,

n : GA Population size

s_a : Bus service time on link a which is equal to running time plus duel time at stops,

$t_{0-1,a}^{c,b}(x)$: Travel time on link a by mode car (c) or bus (b), which is a function of flow, with no exclusive lane (0), with exclusive lane (1)

$x_a^{c,b}$: Passenger flow on link a by car (c) or bus (b),

w_i^b : Waiting time and transfer time at stops.

Bdg : Available budget,

mp	Experiment Code	Approach	Number of Evaluations on Generation 50	Execution Time (sec)	Average time per evaluation	Number of Cores	Number of Threads	Speed up	Efficiency
0.005	E218	SGA	1649	240618	145.9	1	1	1	1
0.005	E220	MT	1513	59865	39.6	4	4	3.687	0.922
0.005	E219	HTC	1454	21607	14.9	10	18	9.821	0.546
0.01	E210	SGA	1680	241475	143.7	1	1	1	1
0.01	E223	MT	1543	66480	43.1	4	4	3.335	0.834
0.01	E227	HTC	1626	24918	15.3	10	18	9.378	0.521
0.02	E215	SGA	1721	231197	134.3	1	1	1	1
0.02	E224	MT	1714	72237	42.1	4	4	3.187	0.797
0.02	E214	HTC	1683	24204	14.4	10	18	9.343	0.519

Table 2. Comparison of the speedup using MT and HTC approaches.

Experiment Code	No. of Cores	No. of Threads	Machine ID	Evaluations on Gen. 50	Exec. Time (sec)	ATE (sec)	Speedup	Efficiency
E210	1	1	4	1680	241475	143.7	1	1
E225	6	10	2, 6	1724	37296	21.6	6.643	0.664
E226	10	14	2, 4, 6	1481	28013	18.9	7.597	0.543
E227	10	18	1, 2, 6	1626	24918	15.3	9.378	0.521
E228	16	22	1, 2, 4, 6	1659	19335	11.7	12.331	0.560
E230	20	26	1, 2, 4, 5, 6	1614	19053	11.8	12.173	0.468
E231	22	28	1, 2, 4, 5, 6, 7	1645	26235	15.9	9.011	0.322

Table 3. Comparison of HTC speedup, varying cores.

$Cpc_{0-1,a}^{c,b}$: Capacity of link a for mode car (c) or bus (b) with no exclusive lane (0), with exclusive lane (1)

Exc_a: Cost of implementing an exclusive lane on link a,

$Imp^{c,b}$: Aggregate weight of operation costs of a car (c) or bus (b) to the community including: emissions, noise, accident, and reliability impacts.

Occ^c: Average occupancy rate for the car mode,

$\alpha, \beta, \gamma, \eta$: Weighting factors to convert the units and adjust the relative importance of each impact in the objective function, $\alpha, \beta, \gamma, \eta \geq 0$,

ϕ_a : Equals to 1 if there is an exclusive lane on link a, 0 otherwise

9 Acknowledgment

We received generous support from PTV AG, the Monash e-Research Centre (MeRC), and the Australian Research Council (ARC) for partial support.

10 References

Akhter, S. and Roberts, J. (2006): *Multi-core Programming: Increasing Performance through Software Multi-threading*, Intel Press.

Athanasaki, E., Anastopoulos, N., Kourtis, K. and Koziris, N. (2008): Exploring the performance limits of simultaneous multithreading for memory intensive applications. *Journal of Supercomputing*, **44**:64-97.

Bard, J. F. (1998): *Practical Bilevel Optimization : Algorithms and Applications*, Kluwer, Dordrecht, The Netherlands.

Bard, J. F. and Falks, J. E. (1982): Explicit solution to the multi-level programming problem. *Computers and Operations Research,* **9:**77-100.

Ben-Ayed, O. and Blair, C. E. (1990): Computational difficulties of bilevel linear programming. *Operations Research*, **38**(3):556-560.

Black, J. A. (1991): Urban arterial road demand management - environment and energy, with particular reference to public transport priority. *Road Demand Management Seminar 1991*, Melbourne, Australia. Haymarket, NSW, Australia, AUSTROADS.

Bly, P. H., Webster, F. V. and Oldfield, R. H. (1978): Justification for bus lanes in urban areas. *Traffic Engineering and Control*, Feb. 1978, **19**(2):56-59.

Cantú-Paz, E. (2000) *Efficient and Accurate Parallel Genetic Algorithms*, Boston, Mass., Kluwer.

Cevallos, F. and Zhao, F. (2006): Minimizing transfer times in public transit network with genetic algorithm. *Transportation Research Record*, **1971**:74-79.

Chen, Q., Shi, F., Yao, J.-L. and Long, K.-J. (2007): Bi-level programming model for urban bus lanes' layout. *Int. Conf. on Transportation Engineering, ICTE 2007,* Chengdu, China, 394-399.

Currie, G., Sarvi, M. and Young, B. (2007): A new approach to evaluating on-road public transport priority projects: Balancing the demand for limited road-space. *Transportation*, **34**:413-428.

Currie, G., Sarvi, M. and Young, W. (2004) A comprehensive approach to balanced road space allocation in relation to transit priority. *83rd TRB annual meeting*. Washington DC, Transportation Research Board.

Department of Infrastructure (2004): *Melbourne Multi-modal Integrated Transport Model (MITM), User Guide.*

Dongarra, J., Sterling, T., Simon, H. and Strohmaier, E. (2005): High-performance computing: clusters, constellations, MPPs and future directions. *Computing in Science and Engineering*, IEEE Computer Society, **7**:51-59.

Duff-Riddell, W. R. and Bester, C. J. (2005): Network modeling approach to transit network design. *Journal of Urban Planning and Development*, **131**:87-97.

Eichler, M. and Daganzo, C. F. (2006): Bus lanes with intermittent priority: Strategy formulae and an

evaluation. *Transportation Research Part B: Methodological*, **40**:31-744.

Evjen, B. (2004): *Professional VB.NET 2003*, Indianapolis, IN, J. Wiley.

Fan, W. and Machemehl, R. B. (2006): Optimal transit route network design problem with variable transit demand: genetic algorithm approach. *Journal of Transportation Engineering*, **132:**pp 40-51.

Foster, I. T., Kesselman, C. and Tuecke, S. (2001): The anatomy of the Grid: enabling scalable virtual organizations. *Int. Journal of Supercomputer Applications*, **15**(3):200-222.

Foster, I. T., Zhao, Y., Raicu, I. and Lu, S. (2008): Cloud computing and grid computing 360-degree compared. *Grid Computing Environments Workshop (GCE '08),* 1-10.

Gentzsch, W. (2001): Sun Grid Engine -- Towards creating a compute power grid. *First IEEE/ACM Int. Symp. on Cluster Computing and the Grid*, 35-36.

Goldberg, D. E. (2002): *The Design Of Innovation: Lessons From and For Competent Genetic Algorithms*, Boston, Kluwer Academic Publishers.

Haupt, R. L., Haupt, R. L. and Haupt, S. E. (2004): *Practical Genetic Algorithms*, NY, Wiley Interscience.

Monash University: SPONGE – *Harvesting Spare CPU cycles.* http://www.monash.edu.au/eresearch/activities/sponge.html (last visited 2011).

Jepson, D. and Ferreira, L. (2000): Assessing travel time impacts of measures to enhance bus operations. Part 2: Study methodology and main findings. *Road and Transport Research*, **9**:4-19.

Litzkow, M. J., Livny, M. and Mutka, M. W. (1988): Condor – a hunter for idle workstations. *8th Int. Conf. on Distributed Computing Systems*, 104-111.

Liu, R., Van Vliet, D. and Watling, D. (2006): Microsimulation models incorporating both demand and supply dynamics. *Transportation Research Part A: Policy and Practice*, **40**:125-150.

Liu, W.-M., Jiang, S. and Fu, L.-F. (2008): Bi-level program model for multi-type freeway discrete equilibrium network design. *Zhongguo Gonglu Xuebao/China Journal of Highway and Transport*, **21**:94-99.

Mesbah, M., Sarvi, M. and Currie, G. (2008): A new methodology for optimizing transit priority at the network level. *Transportation Research Record: Journal of the Transportation Research Board*, **2089**:93-100.

Mesbah, M., Sarvi, M. and Currie, G. (2011): Optimization of transit priority in the transportation network using a genetic algorithm. *IEEE Transactions on Intelligent Transportation Systems*, **12**:908-919.

Mesbah, M., Sarvi, M., Currie, G. and Saffarzadeh, M. (2010): A policy making tool for optimization of transit priority lanes in an urban network. *Transportation Research Record*, **2197**:54-62.

Mesbah, M., Sarvi, M., Ouveysi, I. and Currie, G. (2011): Optimization of transit priority in the transportation network using a decomposition methodology. *Transportation Research Part C: Emerging Technologies*, **19**: 363-373.

Ortúzar, J. D. D. and Willumsen, L. G. (2001): *Modelling Transport*, Chichester NY, J. Wiley.

Papacostas, C. S. and Prevedouros, P. D. (1993): *Transportation Engineering and Planning*, Englewood Cliffs, NJ, Prentice-Hall.

PTV AG (2009): *VISUM 11 User Manual, 11th ed*. Karlsruhe, Germany.

Raicu, I., Foster, I. T. and Zhao, Y. (2010) Many-task computing for grids and supercomputers. *IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS)*, 1-11.

Russell, S. J. and Norvig, P. (2003): *Artificial Intelligence: a modern approach*, Upper Saddle River, N.J., Prentice-Hall.

Sheffi, Y. (1984): *Urban Transportation Networks: Equilibrium Analysis With Mathematical Programming Methods*, Englewood Cliffs, N.J., Prentice-Hall.

Shimizu, K., Ishizuka, Y. and Bard, J. F. (1997): *Nondifferentiable And Two-Level Mathematical Programming*, Boston, Kluwer Academic Publishers.

Simaan, M. (1977): Stackelberg optimization of two-level systems. *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-7**:554-559.

Sterling, T., Becker, D., Warren, M., Cwik, T., Salmon, J. and Nitzberg, B. (1998): An assessment of Beowulf-class computing for NASA requirements: initial findings from the first NASA workshop on Beowulf-class clustered computing. *Proceedings of IEEE Aerospace Conference*, **4**:367-381.

Stirzaker, C. and Dia, H. (2007): Evaluation of transportation infrastructure management strategies using microscopic traffic simulation. *Journal of Infrastructure Systems*, **13**:168-174.

Strohmaier, E., Dongarra, J. J., Meuer, H. W. and Simon, H. D. (2005): Recent trends in the marketplace of high performance computing. *Parallel Computing*, **31**:261-273.

Thain, D., Tannenbaum, T. and Livny, M. (2005): Distributed computing in practice: the Condor experience: *Concurrency and Computation: Practice and Experience*, **17**:323-356.

Viegas, J. (1996): Turn of the century, survival of the compact city, revival of public transport. In *Meersman, H. and Van De Voorde, E. (Eds.) Transforming the Port and Transportation Business*, Antwerp, Belgium.

Viegas, J. and Lu, B. (2004): The intermittent bus lane signals setting within an area. *Transportation Research Part C: Emerging Technologies*, **12**:453-469.

Waterson, B. J., Rajbhandari, B. and Hounsell, N. B. (2003): Simulating the impacts of strong bus priority measures. *Journal of Transportation Engineering*, **129**:642-647.

Yang, H. and Bell, M. G. H. (1998) Models and algorithms for road network design: a review and some new developments. *Transport Reviews,* **18**:257-278.