A hybrid genetic algorithm for road congestion minimization

Luciana S. Buriol[1] Michael J. Hirsch[2] Panos M. Pardalos [3] Tania Querido[3]
Mauricio G. C. Resende[4] Marcus Ritt[1]

[1]*Instituto de Informática, Universidade Federal do Rio Grande do Sul*
Av. Bento Gonçalves, 9500, Porto Alegre, Brazil.
{buriol,marcus.ritt}@inf.ufrgs.br

[2]*Raytheon, Inc., Network Centric Systems*
P.O. Box 12248, St. Petersburg, FL, USA.
mjh8787@ufl.edu

[3]*Dept. of Industrial and Systems Engineering, University of Florida*
303 Weil Hall, Gainesville, FL, 32611, USA.
{pardalos,queridо}@ufl.edu

[4]*Internet and Network Systems Research Center, AT&T Labs Research*
180 Park Avenue, Room C241, Florham Park, NJ 07932 USA.
mgcr@research.att.com

ABSTRACT

One of the main goals in a transportation planning process is to achieve solutions for two classical problems: the traffic assignment problem, which minimizes the total travel delay among all travelers, and the toll pricing problem which settles, based on data derived from the first problem, the tolls that would collectively benefit all travelers and would lead to a user equilibrium solution. Acquiring precision for this framework is a challenge for large networks. In this article, we propose an approach to solve the two problems jointly, making use of a Hybrid Genetic Algorithm for the optimization of transportation network performance by strategically allocating tolls on some of the links. Since a regular transportation network may have thousands of intersections and hundreds of roads, our algorithm takes advantage of mechanisms for speeding up shortest path algorithms.

Keywords: Transportation networks. Genetic algorithm. Shortest paths. Applications to Logistics and Transportation.

RESUMO

Um dos principais objetivos no planejamento de redes de transporte é obter soluções para dois problemas clássicos: o problema de designação de tráfego, o qual minimiza o tempo total de atraso considerando todos os viajantes, e o problema de designar valores aos pedágios o qual, considerando os dados do primeiro problema, atribui valores de pedágio que beneficiam todos os viajantes, conduzindo a uma solução de equilíbrio. Resolver tais problemas é um desafio para redes de grande dimensão. Neste artigo, propõe-se a resolução desses dois problemas em conjunto, fazendo uso de um Algoritmo Genético Híbrido para otimização de redes de transporte, alocando pedágios estrategicamente em alguns links. Visto que redes de transporte, em geral, são redes de grande dimensão, o algoritmo tem seu desempenho melhorado ao fazer uso de atualização dinâmica de caminhos mínimos.

Palavras-chave: Redes de transporte. Algoritmos genéticos. Caminhos mínimos. Aplicações a Logística e Transportes.

1 Introduction

Stable transportation systems are one of the main factors contributing to a high quality of life. Moreover, as reported by Arnott and Small (1994), millions of dollars are spent every day on traffic issues. Thus, traffic planning is a crucial component of any planning process for investment and operating policies. Traffic Assignment models have been used to provide the necessary description of real-world traffic flows with accuracy. These problems are mathematically modeled on a graph structure, with nodes representing locations of interest and arcs representing valid roads on which traffic can flow. Some pairs of nodes are called commodities, or origin-destination (*OD*) pairs, representing traffic flow start and end points. In most instances, each arc of the network has an associated capacity and cost of use, as a function of the amount of traffic using the arc. In addition, some arcs might have tolls levied on them, adding to the arc cost. The main goal in the traffic planning model is to levy tolls on some arcs of the network such that the overall cost of the network (the sum of the cost of each arc) is minimized.

As an example, one can look at New York City. Each day, many people living in New Jersey commute into New York City to work. Suppose we label the city of Newark (in New Jersey) as one origin node of our traffic network, and the borough of Queens (in NYC) as a destination node. It is easy to see that there are many possible traffic paths to go from the origin node to the destination node. Some of the arcs in these paths have tolls levied on them (Holland and Lincoln tunnels, for example), while others do not. In addition, each arc has an associated cost as a function of the number of commuters using that arc. Each commuter ideally would want to minimize his/her cost of getting from their respective origins to their respective destinations.

Optimizing transportation network performance has been widely discussed in the literature (Bai, 2004; Bai et al., 2006; Dial, 1999a,b; Florian and Hearn, 1995; Lawphongpanich and Hearn, 2004) and two fundamental traffic assignment models have been developed: *User Equilibrium* (*UE*) and *System Optimal* (*SO*) models. *UE* is used to describe the behavior of users on a given traffic network. In a *UE* solution, each driver will follow his/her shortest path (least cost path) in traveling from their origin to their destination. In contrast, *SO* describes a traffic network in its best operation. This means that a *SO* solution seeks to spread the flow over all the arcs of the network so that the overall network cost is minimized. Hence, a *UE* solution attempts to do what is best for each individual driver, without consideration of other users on the network, while a *SO* solution considers the overall performance of the network, without consideration of any one individual user. These two concepts seem contradictory, and in a way they are. The overall traffic assignment problem can therefore be viewed as simultaneously solving the *UE* and *SO* problems, i.e. to find a traffic flow that is both *UE* and *SO*. In most instances, tolls are introduced on some of the arcs in the network so that the resulting *SO* and *UE* solutions coincide.

It is important to note that while the transportation problem can be stated in terms of both system optimality and user equilibrium, to the best of our knowledge, there has been no effort to solve these problems jointly. In effect, the problem has always been split into two problems. In the literature, first the *SO* problem is considered (see, for instance Hearn and Ribera (1980)). Convex functions are used to represent the cost of traveling along each arc, as a function of the flow on the arc. This problem is solved to optimality, and the *SO* solution is then used as input into the *UE* problem. In order to induce users to choose the *SO* path solution, tolls are levied on certain arcs within the traffic network. A genetic algorithm which solves the toll location and level problem separately has been proposed by Shepherd and Sumalee (2004).

The Minimum Toll Booth Problem (*MINTB*) (Bai et al., 2006) describes an approach that minimizes the number of toll locations for which a *UE* solution is achieved, maintaining the *SO* solution. *MINTB* was formulated as a mixed-integer program (Bai et al., 2006), and is easily shown to be in the class of *NP-hard* problems (Bai, 2004). Various heuristics have been designed in an effort to solve the *MINTB*. The reader is referred to Bai (2004) for a complete description of this methodology, as well as background on traffic assignment problems.

One problem with the above two-phase approach is that the *SO* solution may result in an infeasible *UE* program. Hearn and Ramana (1988) report infeasibility with a toll pricing problem for a network of 416 links, 962 nodes and 1623 *OD* pairs, when an approximate solution to the *SO* program, with a relative optimality gap of 10^{-3}, is used to construct the constraints defining the *MINTB* program. To overcome infeasibility, methods based on penalty terms (Hearn and Ribera, 1980) and relaxation of constraints (Kim and Pardalos, 1999) are employed. However, acquiring precision for this framework remains a challenge for larger networks. Another issue, related to the heuristics defined for the *MINTB* problem, is to select an appropriate neighborhood structure, that is a set of solutions near a given solution. In Bai (2004); Kim and Pardalos (1999) a binary vector $\{y_a\}$ is used to indicate whether arc a has a toll levied on it. They limit the concept of neighborhood to adjacent vertexes in the unit hyper-cube (N.B.: each binary vector $\{y_a\}$ can be seen as one vertex of the unit hyper-cube). Due to this definition of neighborhood, even for small problem instances, the computation time was reported as large, and/or the quality of the solution was poor.

In this article we propose to use a Hybrid Genetic Algorithm with local improvement, first presented in Ericsson et al. (2002), for the optimization of traffic flow, leading to a system efficient pattern and user optimal solution on the network. We compare our approach with the two-phase approaches in the literature.

This paper is organized as follows. In Section 2 we present the mathematical framework for the traffic assignment problem. Section 3 describes our Hybrid Genetic Algorithm used to determine the optimal traffic pattern and tolling scheme. Computational results are reported in Section 4. Finally, conclusions are presented in Section 5.

2 Problem Formulation

Given a network topology and certain traffic flow demands, we levy tolls on arcs, seeking an efficient system such that the resulting commodity least-cost paths (*UE* solution) is optimal for the overall system. In a mathematical framework, consider a directed graph $G = (N, A)$, with N representing the set of nodes and A the set of arcs. Each arc $a \in A$ has an associated capacity c_a and cost Φ_a, which is a function of the load ℓ_a (or flow) on the arc, the time t_a to transverse the arc, power n_a, and cost Γ_a. In real-world traffic networks, arc (road) delay are generally described by nonlinear functions associated with these network congestion parameters. We assume that Φ_a is a strictly increasing, convex function. In addition, define $K \subseteq N \times N$ to be the set of commodities, or origin-destination (*OD*) pairs, having $o(k)$ and $d(k)$ as origin and destination nodes, respectively, $\forall\ k \in \hat{K} = \{1, \ldots, |K|\}$. Each commodity $k \in \hat{K}$ has an associated demand of traffic flow d_k defined, i.e. for each *OD* pair $\{o(k), d(k)\}$, there is an associated amount of flow d_k that emanates from node $o(k)$ and terminates at node $d(k)$. Furthermore, define x_a^k to be the the contribution of commodity k to the flow on arc a.

Then, we can write the traffic optimization problem as (1) - (4).

$$\text{minimize} \quad \Phi = \sum_{a \in A} \ell_a t_a \big[1 + \Gamma_a (\ell_a / c_a)^{n_a}\big] \Big/ \sum_{k \in \hat{K}} d_k \tag{1}$$

$$\text{subject to} \quad \ell_a = \sum_{k \in \hat{K}} x_a^k \qquad \forall\ a \in A \tag{2}$$

$$\sum_{i:(j,i) \in A} x_{(j,i)}^k - \sum_{i:(i,j) \in A} x_{(i,j)}^k = \begin{cases} -d_k & \text{if } j = d(k) \\ d_k & \text{if } j = o(k) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$x_a^k \geq 0,\ \forall\ a \in A,\ \forall\ k \in \hat{K}. \tag{4}$$

The objective function (1) represents the mean delay time for the system which is based on the Bureau of Public Roads (BPR) function for travel costs. This function may vary according to a specific network. Φ uses the volume delay (time) on arc a as a function of total flow. Our goal is to allocate tolls on arcs such that the delay value Φ is minimized and we have a system efficient solution. In this function, ℓ_a/c_a describes the utilization of arc a. In Section 4 we describe in more detail the delay function for some real-world problems. Constraint (2) defines the load on each arc a as the sum of flow on arc a arising from each commodity. Constraint (3) defines flow conservation on the network, which is equivalent to the system of equations $Bx^k = d_k,\ \forall\ k \in \hat{K}$, where B is the arc-node incidence matrix for the network and $x^k = \{x_a^k\}_{a \in A}$ is the flow vector corresponding to commodity $k \in \hat{K}$. Constraint (4) specifies that the flow on each arc must be non-negative.

As seen in the next section, we distribute tolls on τ of the arcs of the network, leading to a traffic balance and congestion minimization.

3 A Hybrid Genetic Algorithm for the Toll Booth Problem

In this section we summarize the description of the hybrid genetic algorithm used for solving the toll booth problem.

A Genetic Algorithm was successfully applied to Open Shortest Path First (*OSPF*) intra-domain Internet routing problems (Ericsson et al., 2002), and in Buriol et al. (2005) a Hybrid Genetic Algorithm (HGA) was proposed to solve the same problem with additional local improvements. In the present work, we take advantage of some similarities between the *OSPF* routing problem and the traffic assignment problem, and apply the HGA proposed in Buriol et al. (2005) adjusted to optimize the traffic network.

A genetic algorithm is a population-based metaheuristic used to obtain high quality solutions for combinatorial optimization problems. In this context, a population is a set of feasible solutions. Solutions in a population are combined (through crossover) and perturbed (by mutation) to produce a new generation of solutions. When solutions are combined, attributes of higher-quality solutions have a greater probability to be passed down to the next generation. This process is repeated over many generations as long as the quality of the solutions in the new population improves over time. We next show how this idea can be explored for the toll booth problem.

Each solution is represented by two arrays w and b. Array w stores the integer arc weights, while b is a binary array indicating the set of tolls. An arc a of the network has weight equal to w_a in case $b_a = true$ and zero in case $b_a = false$. Each individual weight belongs to the interval $[1, w_{\max}]$. A solution w defines a total flow $\ell_a, a \in A$ by means of an equal-cost multipath routing. In OSPF routing, there is no link weight equal to zero, and the shortest path is the one with the shortest distance. In our implementation, non tolled links are considered to have weight zero. Moreover, two paths are considered of equal cost if they have the same total distance and the same number of hops. In case they have the same total distance, but different hop counts, the shortest path is considered the one with less hops. Each demand is routed forward to its destination. Traffic at intermediate nodes is split equally among all outgoing links on shortest paths to the destination. After the flow is defined, the solution is associated with a fitness value defined by the objective function Φ.

The initial population is randomly generated, with arc weights selected uniformly in the interval $[1, w_{\max}/3]$. A number of K links, chosen at random, are set as having tolls, e.g., b_i is set to *t*rue for K links. The population is partitioned into three sets $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$. The best solutions are kept in $\mathcal{A}$, while the worst ones are in $\mathcal{C}$. All solutions in $\mathcal{A}$ are promoted to the next generation. Solutions in $\mathcal{B}$ are replaced by crossover of one parent from $\mathcal{A}$ with another from $\mathcal{B} \cup \mathcal{C}$ using the *random keys* crossover scheme of Bean (1994). All solutions in $\mathcal{C}$ are replaced by new randomly generated solutions with arc weights selected in the interval $[1, w_{\max}]$.

In the random-keys scheme, crossover is carried out on a selected pair of parent solutions to produce an offspring solution. Unlike Bean (1994), we use a biased random-keys scheme, where

each selected pair consists of an elite parent and a non-elite parent. The elite parent is selected, uniformly at random, from solutions in set $\mathcal{A}$, while the non-elite parent is selected, at random, uniformly from solutions in set $\mathcal{B} \cup \mathcal{C}$. Each weight of the w array in the offspring solution is either inherited from one of its parents or is reset by mutation. With mutation probability p_m, the weight from w is reset to a value selected at random in the interval $[1, w_{\max}]$. If mutation does not occur, then the child inherits the weight from its elite parent with a given probability $p_{\mathcal{A}} > 1/2$. After the crossover, array b is adjusted:

- b_i is *true* in case the correspondent values in both parents are *true*.
- 50% of the positions b_i, chosen at random, which only one of the parents has the corresponding position equal to true, are set to true in the child solution;
- all other positions of b_i are set to *false*.

In this fashion, we generate a child with exactly the same number of tolls than as the parents. Next we describe the solution evaluation.

3.1 Solution Evaluation

Depending on the problem, the main effort of the algorithm can be in the crossover operator or in the solution evaluation. For the case of the weight setting problem the solution evaluation takes longer than the crossover operator. In this section, we describe the procedure used for evaluating a solution. This procedure is presented in Figure 1.

Let T be the set of destination nodes. We compute $|T|$ single-destination shortest path graphs g^t. Each g^t, with destination $t \in T$, has an $|A|$-vector, L^t, associated with the arcs, that stores the partial loads flowing to t traversing each arc $a \in A$. The total load on each arc is represented in the $|A|$-vector l, which stores the total load traversing each arc $a \in A$. For each destination t, the $|N|$-vectors π^t and δ^t are associated with its nodes. The distance from each node to t is stored in π^t, while δ^t keeps the number of arc multiplicities (links) outgoing from each node in g^t.

```
procedure EvaluateSolution(w, lf, rf)
1   forall a ∈ A do μ_a = 1;
2   forall t ∈ T do
3       π^t ← ReverseDijkstra(w);
4       g^t ← ComputeSPG(w, π^t);
5       δ^t ← ComputeDelta(g^t);
6       L^t ← ComputePartialLoads(μ, δ, π, g^t);
7   end forall
8   l ← ComputeTotalLoads(L);
9   S ← UpdateMultiandDelta();
10  if |S| > 0 UpdateSolution();
11  forall a ∈ A if l_a = 0 then μ_a = 0;
12  f ← Σ_{a∈A} μ_a;
13  return(f, μ);
end procedure
```

Figure 1: Pseudo-code for the solution evaluation procedure.

In order to update the system by means of the new arc loads, we compute the shortest paths to all destination nodes $t \in T$ and arrive at a graph $G^t = (N, A^t)$, $\forall\, t \in T$. This is achieved using Dijkstra's well-known shortest path algorithm (Ahuja et al., 1993) with a simple change. A small cost is added to the node distances for each traversed link. With this modification, two paths are considered of equal cost if they have the same total distance and the same hop counts. Since in our network we are computing shortest paths to all destination nodes (i.e. sink nodes), we reverse the direction of all arcs in G and compute the distances π^t_u, $\forall\, u \in N$ to destination in T (Buriol et al.,

2005). Given the shortest paths to each destination, we can calculate the flows L^t for all OD demand pairs with destination t and finally the total flows l. The cost of a solution is computed according to (1). Next the local search procedure is presented.

3.2 Local Improvement Procedure

In this section, we describe the local improvement procedure proposed in Buriol et al. (2005) and adapted for this problem. Starting from a given solution, the local improvement procedure analyzes solutions in the neighborhood of a current solution w searching for a solution having a smaller cost. If such a solution exists, then it replaces the current solution. Otherwise, the current solution is returned as a local minimum.

The local improvement procedure is incorporated in the genetic algorithm to enhance its ability to find better-quality solutions with less computational effort. Local improvement is applied to each solution generated by the crossover operator. Besides being computationally demanding, the use of large neighborhoods in a hybrid genetic algorithm can lead to loss of population diversity, and consequently premature convergence to low-quality local minima. We next describe the local improvement procedure using a reduced neighborhood.

As before, let l_a denote the total load on arc $a \in A$ in the solution defined by the current weight settings w. We recall that $\Phi_a(l_a)$ denotes the routing cost on this arc. The local improvement procedure examines the effect of increasing the weights of a subset of the arcs. These candidate arcs are selected among those with the highest routing costs and whose weight is smaller than $w_{\max}$. To reduce the routing cost of a candidate arc, the procedure attempts to increase its weight, in case there is a toll installed on the arc, in order to induce a reduction of its load. If the selected arc has no toll installed, a toll is installed on it with initial weight one, and the procedure attemps to increase its weight, and a toll is removed from some other link. To select the link to have its toll removed, a subset of ten tolled arcs are tested in circular order to avoid testing an arc twice without having tested all tolled arcs. Initially, tolled arcs are tested in order of increasing routing cost, but once a change is performed, the new tolled arc is placed in the position occupied by the previous tolled arc, and the the order can be not respected anymore, since the vector is not resorted. In case the solution did not improve, the solution returns to the previous state. If this leads to a reduction in the overall routing cost, the change is accepted and the procedure is restarted. The procedure stops at a local minimum when no improvement results from changing the weights of the candidate arcs. The pseudo-code in Figure 2 describes the local improvement procedure in detail.

The procedure `LocalImprovement` takes as input parameters the current solution defined by the weights w, the vector b that indicates which are the tolled arcs, and a parameter q which specifies the maximum number of candidate arcs to be examined at each local improvement iteration.

The counter of candidate arcs is initialized in line 2. The loop in lines 2 to 25 investigates at most q selected candidate arcs for weight increase in the current solution. The arc indexes are renumbered in line 3 such that the arcs are considered in non-increasing order of routing cost.

Arc a' is selected in line 8. If arc a' has no toll installed on it, we install a toll of weight one on it (lines 6 and 7), and mark a $flag$ that this operation was performed.

The loop in lines 10 to 16 examines all possible weight changes for arc a' in the range $[w_{a'} + 1, w_{a'} + \lceil (w_{\max} - w_{a'})/4 \rceil]$. A neighbor solution w', keeping all arc weights unchanged except for arc a', is built in lines 11 and 12. If the new solution w' has a smaller routing cost than the current solution (test in line 13), then the current solution is updated in line 14, arc a' is unmarked in line 15, and the arc counter i is reset in line 16.R

The loop in lines 17 to 23 are executed only if the current arc being analised was previously not tolled. In line 18, each arc belonging to the set R of tolled arcs, are tested one by one, always testing arcs with lower testing costs first. In line 20 we test if the solution is better than the current solution in the beginning of loop in line 2. In case the new solution is better, it is taken as the current solution, and the for loop stops. If there is no better solution, then the current solution is reset to the solution

```
procedure LocalImprovement(q, w, b)
1     i ← 1;
2     while i ≤ q do
3           Renumber the arc indexes such that
                  Ψ_a(l_a) ≥ Ψ_{a+1}(l_{a+1}), ∀a = 1, ..., |A| − 1;
4           a′ ← 1; flag ← F;
5           if b_{a′} = F
6                 b_{a′} ← T
7                 w_{a′} ← 1
8                 flag ← V
9           end if
10          for ŵ = w_{a′} + 1, ..., w_{a′} + ⌈(w_max − w_{a′})/4⌉ do
11                w′_a ← w_a, ∀a ∈ A, a ≠ a′;
12                w′_{a′} ← ŵ;
13                if Ψ_{w′,b′} < Ψ_{w,b} then
14                      w ← w′;
15                end if
16          end for
17          if flag then
18                for ten a′′ ∈ R do
19                      b_{a′′} ← F;
20                      if Ψ_{w′,b′′} < Ψ_{w,b} then break
21                end for
22          end if
23          if notImproved(w, w'') then restoreOriginalSol(a', a'', w'')
24          i ← i + 1;
25    end while
end LocalImprovement.
```

Figure 2: Pseudo-code of procedure `LocalImprovement`.

considered in the beginning of loop of line 2 to 25.

The routing cost $\Phi(w')$ associated with the neighbor solution w' must be evaluated in lines 13, 20, and 23. Instead of computing it from scratch, we use fast update procedures for recomputing the shortest path graphs as well as the arc loads. These procedures are considered in the next section of the paper. Once the new arc loads are known, the total routing cost is computed as the sum of the individual arc routing costs.

3.3 Dynamic Updates

We denote by $G^t = (N, A^t)$ the shortest paths graph associated with each destination node $t \in T$. When the weight of a single arc a' is changed, the graph G^t does not have to be recomputed from scratch. Instead, we update the part of it which is affected by the weight change.

In Buriol et al. (2005), dynamic shortest path algorithms were presented for the case of positive arc weights. In this paper we deal with non-negative weights, i.e. arcs with zero or positive weights.

When a toll is installed in an arc, or a toll is removed from an arc, or the weight of a tolled arc changes, we used the dynamic shortest paths described in Buriol et al. (2005) to update the shortest path graph, instead of recomputing it from scratch.

The possibility of having weights with cost zero cost allows for cycles of cost zero. To avoid that, we add the value $1/|E|$ to the distance for each arc traversed. So, for alternative shortest paths with cost zero, it is possible to know which has fewer hops comparing the real values of their distances. Since a direct path is always shorter in number of hops than a path with cycles, the cycles are eliminated. Using this rule, all alternative shortest paths of cost zero, but longer in number of hops, are also eliminated. Thus, if a node has multiple shortest path of cost zero, just one with the fewest hops will remain.

The loads are also updated, instead of being calculated from scratch. The approach used for updating the affect part of the graph is presented in Buriol et al. (2005), and we use the same

algorithm. Only the part of the graph whose loads were affected by the arc weight increase is explored.

4 Computational Results

4.1 The Nine Node Example

To provide an example on how our HGA works, in comparison to the MINTB approach, we discuss the nine node problem generated in Hearn and Ramana (1988). The objective function used for this problem is based on the BPR data and is the same used to describe cost delay for larger instances. The associated network has 18 links, and four O-D pairs, namely (1,3), (1,4), (2,3) and (2,4). Figure 3 displays the optimality gap obtained for this example when running HGA for different number of tolls.

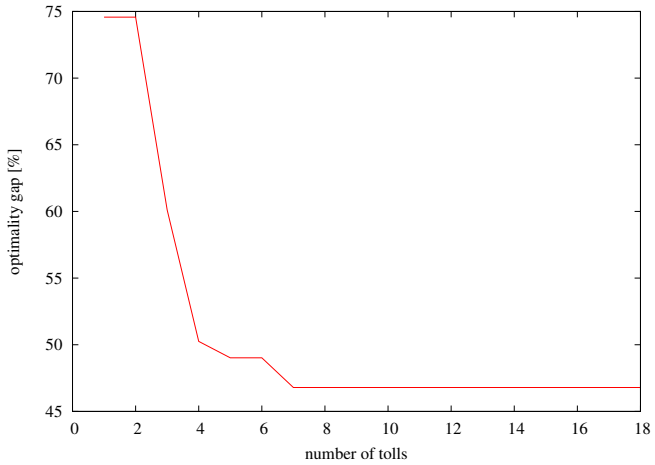

Figure 3: Number of tolls installed vs. optimality gap for the nine node example.

The objective function value of the optimal solution for this instance is 22.59314 (Hearn and Ramana, 1988). It is important to note that our HGA does not produce the optimal configuration. The solution found by the HGA was about the same found by the GA. This is due to the fact, that in small networks, a system optimal solution can significantly deviate from an equal-cost multi-path routing.

4.2 Realistic Problems

Some realistic problems where attributes are known in the transportation science literature have a particular objective function. We consider for example, `Sioux Falls`, North Dakota (LeBlanc et al., 1975). In this case, the delay function on each arc is known as $\Phi_a = \sum_{a \in A} \ell_a t_a [1 + \beta_a (\ell_a / c_a)^4]$. Other instances, such as `Stockholm`, `Winnipeg`, and `Barcelona` are also studied in this paper, and have a similar delay function for their links. Their attributes (number of nodes, number of links, number of O-D pairs) are displayed in Table. 1.

4.3 Optimal solutions

The traffic optimization problem (1)–(4) has a convex objective function and linear constraints. Therefore it can, in principle, be solved by standard methods of convex optimization. We implemented a solver for the traffic optimization problem based on `cvxopt` (Dahl and Vandenberghe, 2005), a freely available solver for convex programs.

Table 1: Attributes of realistic problem instances.

Instance	Vertices	Arcs	OD pairs	Destinations
Sioux Falls	24	76	528	24
Stockholm	416	962	1623	45
Barcelona	1020	2522	7922	108
Winnipeg	1052	2836	4345	138

Table 2: Optimal solutions.

Instance	Optimal value	Solution time [s]
Nine node problem	22.539181	< 1
Sioux Falls	19.950794	22
Stockholm	-	> 86400

Our implementation uses a more compact, but equivalent formulation of (1)–(4), which represents the flows of all O-D pairs with the same destination as a single commodity. This reduces the number of variables from $|A|\,|K|$ to $|A|\,D$, where $D = |\{d \mid (o,d) \in K\}|$ is the number of different destinations. Table 1 shows that this number is a factor between 22 to 73 lesser than the number of O-D pairs.

The solver has been able to produce optimal values only for the two smallest instances shown in Table 2. On the next larger instance, Stockholm, the solver did not terminate within three days of CPU time. Thus, the results of the GA and HGA for the nine node problem and the Sioux Falls instance in Fig. 3 and 4 show the optimality gap (in percent above the optimal solution), while the results for the remaining instances are absolute values. Figure 4 shows that for a sufficient number of installed tolls, the heuristic solution lies within 10% of the SO solution.

4.4 Quality of the HGA solutions

We compared the best solution values and the optimality gap (where possible) obtained by the HGA and by the GA (HGA without local search). For each instance, we used different numbers of tolled arcs, varying from a few tolled arcs up to tolls on all arcs. For each number of tolled arcs, we ran the GA and the HGA three times with different random seeds for 5000 generations, but at most up to a time limit of one hour. The results represent the average of these runs. For the experiments, we used a Intel Pentium Core2 Duo, running at 2.4 GHz, with 3 GB of RAM. Each run of instance `Sioux Falls` spent in average about 2 minutes of CPU time, while the runs of instance `Stockholm` spent about 18 minutes. Runs for instances `Winnipeg` and `Barcelona` stopped always by the time limit of 60 minutes. The HGA spent between 50 to 70 percent of its time in the local search.

Figures 4 to 7 show computational results for instances `Sioux Falls`, `Stockholm`, and `Winnipeg`, respectively. On the x-axis are presented the number of tolls installed, while the y-axis presents the solution value. For each instance, we present results found by the GA and HGA algorithms.

By the experimental results we can observe that the solution obtained by the HGA and GA algorithms are competitive. For instances `Sioux Falls` and `Stockholm` the HGA presented better results, while for instances `Winnipeg` and `Barcelona` the GA presented better solutions.

For most of the instances, the quality of the results improves with larger toll sets. The solution value almost decreases monotonically with an increasing number of tolls, with exception of instance `Winnipeg` (instance `Barcelona` presented this behavior only for the HGA algorithm).

Given that in almost all cases the solution is better for a larger number of tolled links, one can choose the optimal trade-off between the number of tolled links and the quality of the solution.

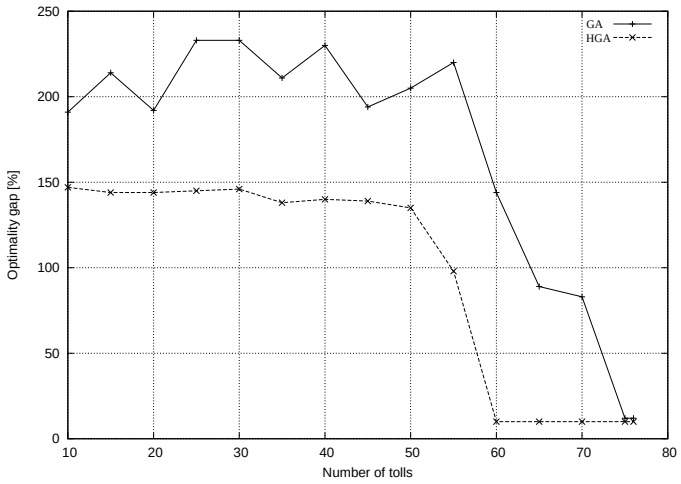

Figure 4: Number of tolls installed vs. quality of results, for the GA and HGA algorithms, for instance `Sioux Falls`. The number of installed tolls tested varies from 10 up to 75, increasing five by five.

5 Conclusions

In the present work, we have adapted the evolutionary algorithm from Buriol et al. (2005) to a transportation problem. We tested both the genetic algorithm and the hybrid genetic algorithm. By means of computing a solution that minimizes the mean delay of the system, we deal with both *SO* and *UE* problems simultaneously. As we have applied a heuristic to solve this problem, there is no guarantee that the system optimal solution is achieved. Instead, an efficient solution for the overall transportation system is obtained. We show the genetic algorithm as well as the hybrid genetic algorithm obtain solutions of good quality. For the `Sioux Falls` we were able to confirm an optimality gap of less than 10%. Solutions for other three large instances were presented, showing the ability of the GA and HGA algorithms to deal with large instances.

6 Acknowledgements

Luciana S. Buriol and Marcus Ritt have received support from the Brazilian government (CNPq) under project no. 481256/2008-3.

References

Ahuja, R. K., Magnanti, T. L. and Orlin., J. B. (1993). *Network Flows – theory, algorithms, and applications*, Prentice Hall.

Arnott, R. and Small, K. (1994). The economics of traffic congestion, *American Scientist* **82**: 446–455.

Bai, L. (2004). *Computational methods for toll pricing models*, PhD thesis, University of Florida, Gainesville, Florida.

Bai, L., Hearn, D. W. and Lawphongpanich, S. (2006). Relaxed toll sets for congestion pricing problems, *in* S. L. D.W. Hearn and M. Smith (eds), *Mathematical and Computational Models for Congestion Charging*, Springer.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization, *ORSA J. on Comp.* **6**: 154–160.

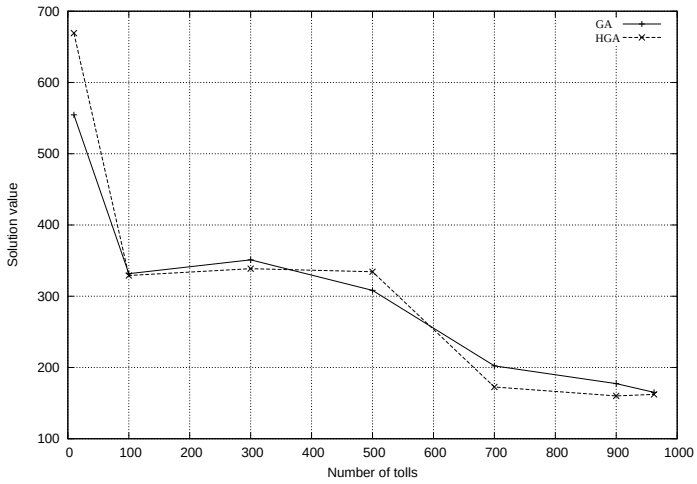

Figure 5: Number of tolls installed vs. quality of results, for the GA and HGA algorithms, for instance `Stockholm`. The number of installed tolls tested were 10, 100, 300, 500, 700, and 900.

Buriol, L. S., Resende, M. G. C., Ribiero, C. C. and Thorup, M. (2005). A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing, *Networks* **46**: 36–56.

Dahl and Vandenberghe (2005). CVXOPT.
URL: *http://abel.ee.ucla.edu/cvxopt*

Dial, R. B. (1999a). Minimal-revenue congestion pricing part I: A fast algorithm for the single origin case, *Transportation Research Part B* **33**: 189–202.

Dial, R. B. (1999b). Minimal-revenue congestion pricing part II: An efficient algorithm for the general case, *Transportation Research Part B* **34**: 645–665.

Ericsson, M., Resende, M. G. C. and Pardalos, P. M. (2002). A genetic algorithm for the weight setting problem in OSPF routing, *Journal of Combinatorial Optimization* **6**: 299–2002.

Florian, M. and Hearn, D. (1995). Network equilibrium models and algorithms, *in* M. O. Ball et al. (eds), *Network Routing*, Elsevier Science, pp. 485–550.

Hearn, D. W. and Ramana, M. (1988). *Solving congestion toll pricing models*, Equilibrium and Advances in Transportation Modeling, North-Holland, New York.

Hearn, D. W. and Ribera, J. (1980). Bounded flow equilibrium by penalty methods, *Proceedings of the IEEE International Conference on Circuits and Computers* **1**: 162–164.

Kim, D. and Pardalos, P. (1999). A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure, *Operations Research Letters* **24**: 195–203.

Lawphongpanich, S. and Hearn, D. W. (2004). An MPEC approach to second-best toll pricing, *Mathematical Programming, Series B* pp. 33–55.

LeBlanc, L. J., Morlok, E. K., and Pierskalla, W. P. (1975). An efficient approach to solving the road network equilibrium traffic assignment problem, *Transportation Research* **9**: 309–318.

Shepherd, S. and Sumalee, S. (2004). A genetic algorithm based approach to optimal toll level and location problems, *Networks and Spatial Economics* **4**(2): 161–179.

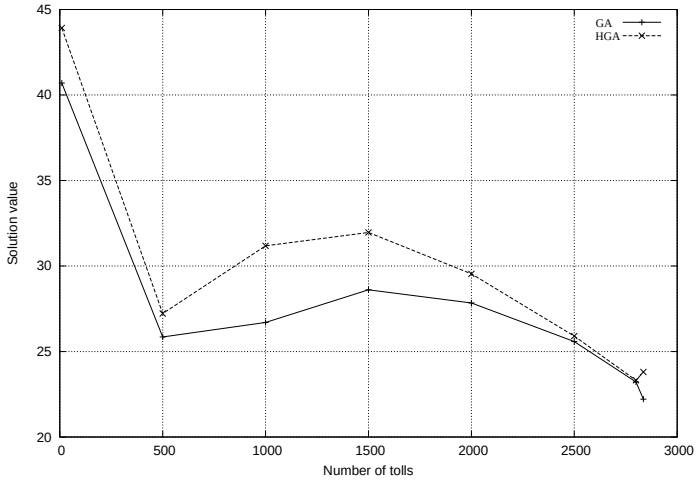

Figure 6: Number of tolls installed vs. quality of results, for the GA and HGA algorithms, for instance `Winnipeg`. The number of installed tolls tested were 10, 500, 1000, 1500, 2000, 2500, and 2800.

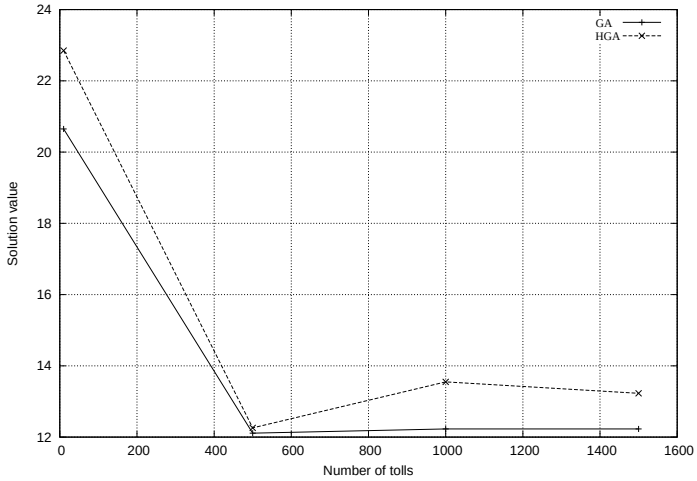

Figure 7: Number of tolls installed vs. quality of results, for the GA and HGA algorithms, for instance `Barcelona`. The number of installed tolls tested were 10, 500, 1000, 1500, 2000, and 2500.