Optim Lett (2010) 4:619–633
DOI 10.1007/s11590-010-0226-6

ORIGINAL PAPER

A biased random-key genetic algorithm for road congestion minimization

Luciana S. Buriol · Michael J. Hirsch · Panos M. Pardalos · Tania Querido · Mauricio G. C. Resende · Marcus Ritt

Received: 11 July 2010 / Accepted: 27 July 2010 / Published online: 10 August 2010

Abstract One of the main goals in transportation planning is to achieve solutions for two classical problems, the traffic assignment and toll pricing problems. The traffic assignment problem aims to minimize total travel delay among all travelers. Based on data derived from the first problem, the toll pricing problem determines the set of tolls and corresponding tariffs that would collectively benefit all travelers and would lead to a user equilibrium solution. Obtaining high-quality solutions for this framework is a challenge for large networks. In this paper, we propose an approach to solve

L. S. Buriol · M. Ritt
Instituto de Informática, Universidade Federal do Rio Grande do Sul, Av. Bento Gonçalves, Porto Alegre 9500, Brazil
e-mail: buriol@inf.ufrgs.br

M. Ritt
e-mail: marcus.ritt@inf.ufrgs.br

M. J. Hirsch
Raytheon Company, 300 Sentinel Drive, Annapolis Junction, MD 20701, USA
e-mail: mjh8787@ufl.edu

P. M. Pardalos
Department of Industrial and Systems Engineering, University of Florida, 303 Weil Hall, Gainesville, FL 32611, USA
e-mail: pardalos@ufl.edu

T. Querido
Linear Options Consulting, 7450 SW 86th Way, Gainesville, FL 32608, USA
e-mail: tania@linearoptions.com

M. G. C. Resende (✉)
Algorithms and Optimization Research Department, AT&T Labs Research, 180 Park Avenue, Room C241, Florham Park, NJ 07932, USA
e-mail: mgcr@research.att.com

the two problems jointly, making use of a biased random-key genetic algorithm for the optimization of transportation network performance by strategically allocating tolls on some of the links of the road network. Since a transportation network may have thousands of intersections and hundreds of road segments, our algorithm takes advantage of mechanisms for speeding up shortest-path algorithms.

Keywords Transportation networks · System optimal · User equilibrium · Genetic algorithms · Dynamic shortest paths

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 [2], 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, with nodes representing locations of interest and arcs representing valid roads on which traffic can flow. Some pairs of nodes define commodities, or origination–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 origination 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 origination 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 the cost of their trip, i.e. of getting from their starting point to their destination.

Optimizing transportation network performance has been widely discussed in the literature [3,4,12,13,15,20] 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 their shortest path (least cost path) in traveling from their origination to their destination. In contrast, SO describes a traffic network at its best operation. This means that a SO solution seeks to spread the traffic 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 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 subproblems. In the literature, the SO problem is first considered (see, for instance [18]). 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 [23].

The minimum toll booth problem (MINTB) was introduced in [4] as an approach to minimize the number of toll locations for which a UE solution is achieved, maintaining the SO solution. MINTB was formulated as a mixed-integer program [4], and is NP-hard [3]. Various heuristics have been designed in an effort to solve the MINTB. The reader is referred to [3] for a survey of traffic assignment problems as well as solution techniques.

One problem with the above two-phase approach is that the SO solution may result in an infeasible UE solution. Hearn and Ramana [17] report infeasibility with a toll pricing problem for a network with 416 links, 962 nodes, and 1,623 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 [18] and relaxation of constraints [19] are employed. However, obtaining high quality solutions for this framework remains a challenge for large 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 [3,19] a binary vector $\{y_a\}$ is used to indicate whether arc a has a toll levied on it and the neighborhood is limited to adjacent vertices 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 neighborhood, even for small problem instances, the computation time reported was large, and the quality of the solution was poor. For a complete review of the design and evaluation of road network pricing schemes we refer the reader to the survey by Tsekeris and Voß [25].

In this paper we propose a biased random-key genetic algorithm (BRKGA), similar to the one presented in [14] for the optimization of Internet traffic flow, that seeks a system efficient pattern and user optimal solution for a fixed number of toll booths. The results presented here improve upon the ones in [9].

This paper is organized as follows. In Sect. 2 we present the mathematical framework for the traffic assignment problem. In Sect. 3 we review the basic concepts of BRKGA. Section 4 describes the BRKGA used to determine the optimal or near-optimal traffic pattern and tolling scheme. Computational results are reported in Sect. 5. Finally, conclusions are presented in Sect. 6.

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 multi-commodity least-cost (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 unloaded arc, a power parameter p_a, and a cost B_a. In real-world traffic networks, arc (road) delays 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 OD pairs, having $o(k)$ and $d(k)$ as origination 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 contribution of commodity k to the flow on arc a. We can write the traffic optimization problem as

$$\text{minimize} \quad \Phi = \sum_{a \in A} \ell_a t_a \left[1 + B_a (\ell_a / c_a)^{p_a}\right] / \sum_{k \in \hat{K}} d_k \tag{1}$$

subject to

$$\ell_a = \sum_{k \in \hat{K}} x_a^k, \quad \forall\, a \in A \tag{2}$$

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

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

Objective function (1) represents the average trip duration for the system, based on the travel cost function of the Bureau of Public Roads (BPR) [7]. 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 a fixed number $\mathcal{K}$ of tolls on arcs such that the delay value Φ is minimized, thus corresponding to a system efficient solution. In this function, ℓ_a / c_a is the utilization of arc a. In Sect. 5 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 $Mx^k = d_k,\ \forall\, k \in \hat{K}$, where M 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.

Our approach to solving (1–4) is indirect. We seek to levy tolls on $\mathcal{K}$ arcs of the transportation network such that if traffic is routed on least cost paths, then the

objective function $\Phi = \sum_{a \in A} \ell_a t_a [1 + B_a (\ell_a / c_a)^{p_a}] / \sum_{k \in \hat{K}} d_k$ is minimized. We call this problem the *toll booth problem*. We accomplish this minimization with a BRKGA.

3 Biased random-key genetic algorithms

Genetic algorithms with random keys, or *random-key genetic algorithms* (RKGA), were first introduced by Bean [6] for solving combinatorial optimization problems involving sequencing. In a RKGA, chromosomes are represented as vectors of randomly generated real numbers in the interval [0, 1]. A deterministic algorithm, called a *decoder*, takes as input a solution vector and associates with it a solution of the combinatorial optimization problem for which an objective value or fitness can be computed.

A RKGA evolves a population of random-key vectors over a number of iterations, called *generations*. The initial population is made up of p vectors of n random-keys. Each component of the solution vector is generated independently at random in the real interval [0, 1]. After the fitness of each individual is computed by the decoder in generation k, the population is partitioned into two groups of individuals: a small group of p_e *elite* individuals, i.e. those with the best fitness values, and the remaining set of $p - p_e$ *non-elite* individuals. In the experiments described in Sect. 5, we set $p_e = 0.25p$. To evolve the population, a new generation of individuals must be produced. All elite individual of the population of generation k are copied without modification to the population of generation $k + 1$. RKGAs implement mutation by introducing *mutants* into the population. A mutant is simply a vector of random keys generated in the same way that an element of the initial population is generated. At each generation, a small number p_m of mutants is introduced into the population. In the experiments described in Sect. 5, we set $p_m = 0.05p$. With the p_e elite individuals and the p_m mutants accounted for in population $k + 1$, $p - p_e - p_m$ additional individuals need to be produced to complete the p individuals that make up the new population. This is done by producing $p - p_e - p_m$ offspring through the process of mating or crossover.

Bean [6] selects two parents at random from the entire population to implement mating in a RKGA. A *biased random-key genetic algorithm,* or BRKGA [16], differs from a RKGA in the way parents are selected for mating. In a BRKGA, each element is generated combining one element selected at random from the elite partition in the current population and one from the non-elite partition. Repetition in the selection of a mate is allowed and therefore an individual can produce more than one offspring in the same generation. *Parameterized uniform crossover* [24] is used to implement mating in BRKGAs. Let ρ_e be the probability that an offspring inherits the vector component of its elite parent. Let n denote the number of components in the solution vector of an individual. For $i = 1, \ldots, n$, the ith component $c(i)$ of the offspring vector c takes on the value of the ith component $e(i)$ of the elite parent e with probability ρ_e and the value of the ith component $\bar{e}(i)$ of the non-elite parent $\bar{e}$ with probability $1 - \rho_e$. In the experiments described in Sect. 5, we set $\rho_e = 0.7$.

When the next population is complete, i.e. when it has p individuals, fitness values are computed for all of the newly created random-key vectors and the population is partitioned into elite and non-elite individuals to start a new generation.

A BRKGA searches the solution space of the combinatorial optimization problem indirectly by searching the continuous n-dimensional hypercube, using the decoder to map solutions in the hypercube to solutions in the solution space of the combinatorial optimization problem where the fitness is evaluated.

To specify a BRKGA, we simply need to specify how solutions are encoded and decoded. We specify our algorithm next.

4 A BRKGA for the toll booth problem

In this section we describe a BRKGA for the toll booth problem by first showing how solutions are encoded and then how they are decoded.

A BRKGA was applied to open shortest path first (OSPF) [10,14] and distributed exponentially weighted flow splitting (DEFT) [22] intra-domain Internet routing problems. In this paper, we take advantage of some similarities between the OSPF routing problem and the traffic assignment problem, and apply a BRKGA, similar to the one proposed in [10], to optimize traffic networks.

4.1 Solution encoding

Solutions in the toll booth problem are specified by the location of the toll booths and the corresponding toll tariff for each booth. Solutions are encoded with the $2m$-dimensional array $\mathcal{X}$, where $m = |A|$. Components $\mathcal{X}_{m+1}, \ldots, \mathcal{X}_{2m}$ determine the location of the toll booths and together with components $\mathcal{X}_1, \ldots, \mathcal{X}_m$, they determine the toll tariffs. Tariffs are integer-valued in the interval $[1, w_{\max}]$.

4.2 Solution decoding

The decoder takes as input the encoded solution vector $(\mathcal{X}_1, \ldots, \mathcal{X}_m, \mathcal{X}_{m+1}, \ldots, \mathcal{X}_{2m})$, and outputs a solution of the toll booth problem, i.e. the location and tariff of each of the $\mathcal{K}$ toll booths as well as the average trip duration Φ (which is the objective function we seek to minimize). The random keys $\mathcal{X}_{m+1}, \ldots, \mathcal{X}_{2m}$ are used to determine the location of the $\mathcal{K}$ toll booths. The keys are sorted (ties are broken by key index) and the indices of $\mathcal{K}$ largest keys correspond to the locations where tolls are to be levied, i.e. if key $\mathcal{X}_{m+a}$ is one of the $\mathcal{K}$ largest among keys $\mathcal{X}_{m+1}, \ldots, \mathcal{X}_{2m}$, then a toll booth is placed on arc a. For all $a \in A$, let the zero-one variable $b_a = 1$ if and only if a toll booth is placed on arc a. Then arc $a \in A$ has toll tariff

$$w_a = \lceil \mathcal{X}_a \cdot w_{\max} \rceil \cdot b_a, \tag{5}$$

i.e. if arc a has a toll booth, the value of its toll tariff is determined by the random key $\mathcal{X}_a$.

Fig. 1 Pseudo-code for the solution evaluation procedure

```
procedure EvaluateSolution(w)
1    forall t ∈ T do
2        π^t ← ReverseDijkstra(w);
3        g^t ← ComputeSPG(w, π^t);
4        δ^t ← ComputeDelta(g^t);
5        L^t ← ComputePartialLoads(μ, δ, π, g^t);
6    end forall
7    l ← ComputeTotalLoads(L);
8    Φ ← Σ_{a∈A} φ_a;
9    return(Φ);
end procedure
```

Once toll booth locations with their corresponding tariffs are determined, the decoder needs to determine the cost of the solution w. To do this, the total flow ℓ_a is computed for all edges $a \in A$ by routing the traffic on least cost routes. This is accomplished by routing forward each demand from its starting node to its destination. Traffic at intermediate nodes is split equally among all outgoing links on least cost paths to the destination. The corresponding cost (mean delay time) is computed as

$$\Phi = \sum_{a \in A} \ell_a t_a [1 + B_a (\ell_a / c_a)^{p_a}] / \sum_{k \in \hat{K}} d_k. \tag{6}$$

A fast local search attempts to modify the tariffs to further reduce the mean delay.

In OSPF routing, there is no link weight equal to zero, so the least weight path is the one with the shortest distance. For the toll booth problem, untolled links are considered to have zero weight (no tariff). The least cost paths are calculated based on the tariffs of the tolled arcs. Depending on the number of tolls, there can be several paths with cost zero. Thus, we consider a slightly different notion of least cost path. Two paths are considered of equal cost if they have the same total cost and the same number of hops. In case they have the same total cost, but different hop counts, the least cost path is considered to be the one with the fewest hops.

In the remainder of this section, we show how the total flow on each edge is computed and how local search is carried out.

4.3 Solution evaluation

The main computational bottleneck of this BRKGA is in the solution evaluation. In this section, we describe the procedure used for evaluating a solution (see pseudo-code in Fig. 1).

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

In order to update 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 [1] with a simple change. Two paths are considered of equal cost if they have the same total distance and the same hop counts. Since we compute shortest paths to all destination nodes, we reverse the direction of all arcs in G and compute the distances π_u^t, $\forall\, u \in N$, to destination in T, similar to what is done in [10]. 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 function (6).

4.4 Local improvement procedure

We next describe an adaptation for the toll booth problem of the local improvement procedure proposed in [10]. Starting from a given solution w produced by the decoder, 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 the decoded solution.

Besides being computationally demanding, the use of large local search neighborhoods in a BRKGA decoder can lead to loss of population diversity, and consequently premature convergence to low-quality local minima. Below, we describe the local improvement procedure using a small 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 small subset of the arcs. These candidate arcs are selected among those with the highest routing costs and whose tariff is less than $w_{\max}$. To reduce the routing cost of a candidate arc, the procedure attempts to increase its tariff, in case there is a toll installed on the arc, in order to induce a reduction of its load with the objective of reducing not only Φ_a but also the total cost Φ. If the selected arc has no toll booth present, a toll booth is created on it with tariff of value one. After the procedure attempts to increase its tariff, a toll booth is removed from some other link to maintain the constant number of toll booths.

To select the link to have its toll removed, a subset of r arcs of R, the set of tolled arcs, are tested in circular order to avoid testing an arc twice without having tested all tolled arcs. In case the solution does not improve, the search 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 change in the toll tariffs can improve the total routing cost Φ. The pseudo-code in Fig. 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 in each local improvement iteration. In the experiments described in Sect. 5, we set

```
procedure LocalImprovement(q, w, b)
1     Sort the arcs such that Φ_a ≥ Φ_{a+1}, ∀a = 1,...,|A| − 1;
2     for 1 ≤ i ≤ q do
3           w̃ ← w; b̃ ← b; Φ' ← Φ_{w,b};
4           a' ← a_i; flag ← false;
5           if b_{a'} = 0, i.e. a' has no toll booth then
6                 b_{a'} ← 1; w_{a'} ← 1;
7                 flag ← true;
8           end if
9           for ŵ = w_{a'} + 1,...,w_{a'} + ⌈(w_max − w_{a'})/4⌉ do
10                w'_a ← w_a, ∀a ∈ A, a ≠ a';
11                w'_{a'} ← ŵ;
12                if Φ_{w',b} < Φ_{w,b} then
13                      w ← w';
14                end if
15          end for
16          if flag then
17                for r arcs ā ∈ R do
18                      b'_a ← b_a, ∀a ∈ A, a ≠ ā; b'_ā ← 0;
19                      if Φ_{w,b'} < Φ' then break
20                end for
21          end if
22          if solution improved then
23                b ← b';
24          else
25                w ← w̃; b ← b̃;
26          end if
27    end for
end LocalImprovement.
```

Fig. 2 Pseudo-code of procedure `LocalImprovement`

$q = 5$. In the pseudo-code, solution values for the solution represented by vector pairs $\{w, b\}$ are denoted by $\Phi_{w,b}$.

The loop in lines 2–27 analyzes at most q selected candidate arcs for weight increase in the current solution. The arc indexes are renumbered in line 1 such that the arcs are considered in non-increasing order of routing cost. The current solution $\{w, b\}$ and its cost $\Phi_{w,b}$ is saved in line 3. Arc a' is selected in line 4. If arc a' has no toll booth, we install a toll of tariff one (line 6), and set a *flag* in line 7 to indicate that this operation was performed. The loop in lines 9–15 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 10 and 11. If the new solution w' has a smaller routing cost than the current solution (test in line 12), then the current solution is updated in line 13. The loop in lines 16–21 is executed only if the current arc being analyzed was previously not tolled. In line 17, r arcs belonging to the set R of tolled arcs, are tested one at a time, always considering arcs with smallest costs first. In our experiments, we set $r = 10$. 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 order may be not respected anymore, since the vector is not resorted. We observed that resorting the vector did not show improvement in the final results. In line 19 we test if the solution is better than the current solution in the beginning of the loop in line 2. In case the new solution is better, it is taken as the current solution in line 23, and the for loop stops. If there is

no better solution, then the current solution is reset to the solution at the start of the current iteration in line 25.

The routing cost $\Phi(w')$ associated with the neighbor solution w' must be evaluated in lines 12 and 19. Instead of computing it from scratch, we use fast update procedures for recomputing the shortest path graphs as well as the arc loads. When a toll booth is installed in an arc, or a toll booth is removed from an arc, or the weight of a tolled arc changes, we used the dynamic shortest paths described in [8] to update the shortest path graph, instead of recomputing it from scratch. Once the new arc loads are known, the total routing cost is computed as the sum of the individual arc routing costs. In [10] the use of shortest paths reduced the total runtime of the algorithm by a factor of approximately 15. We observe similar speedups here.

5 Computational results

We performed a set of experiments with the aim of exploring the quality of the solutions found by our heuristic methods. We compare two variants of the BRKGA: one with the fast local search in the decoder and another without. We refer to these heuristics as BRKGA-LS and BRKGA, respectively. For the experiments, we used a computer with an Intel Pentium Core 2 Duo processor, running at 3 GHz, with 2 Gb of main memory. The algorithms were coded in C and compiled with the `gcc` compiler, version 4.2.4 with optimization flag `-O3`.

Both variants of the genetic algorithm were run with identical parameters. The size of the population was set at 50, the elite set had 13 solutions, while the mutant set had three. In the crossover, the probability of the offspring inheriting the random key of the elite parent was set at 70%.

5.1 The nine-node example

To provide an example of how our heuristics work in comparison with the MINTB approach, we consider the nine-node problem proposed in [17]. The objective function used for this problem is based on BPR data and is identical to the one used to describe cost delay for larger instances. The `nine-node` network has 18 links, and four OD pairs. Figure 3 displays the optimality gap obtained for this example when running BRKGA-LS for different numbers of tolls.

The objective function value of the optimal solution of model (1–4) for this instance is 22.59314 [17]. It is important to note that BRKGA-LS does not produce the optimal configuration for this instance. The solution found by BRKGA-LS was about the same as the one found by BRKGA. This is due to the fact that, in small networks, a system optimal solution can deviate significantly from an equal-cost multi-path routing solution.

For this instance, MINTB generated a solution with zero-cost paths for all commodities, while our approach allows non-zero flows on all links. The fact that we split flows evenly on all outgoing arcs while minimizing the cost of the user's paths allows for us to find solutions displaying system efficiency and user equilibrium.

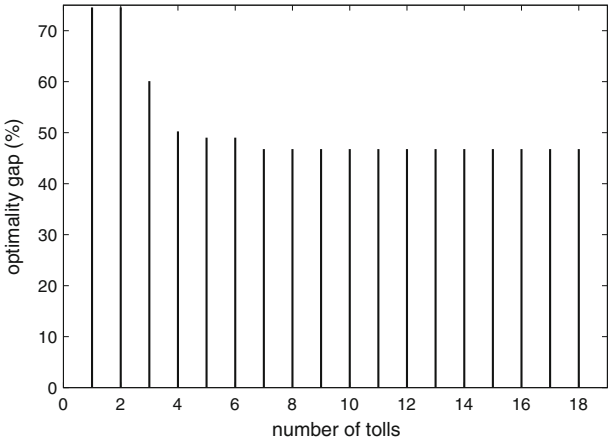

Fig. 3 Number of tolls installed versus optimality gap for the `nine-node` example

Table 1 Attributes of real-world problem instances

Instance	Vertices	Arcs	OD pairs	Destinations
`Sioux Falls`	24	76	528	24
`Stockholm`	416	962	1,623	45
`Barcelona`	1,020	2,522	7,922	108
`Winnipeg`	1,052	2,836	4,345	138

5.2 Real-world problems

Some real-world problems with known attributes from the transportation science literature have a particular objective function. We consider for example, the `Sioux Falls`, North Dakota instance [21]. In this instance, the delay function on each arc is $\Phi_a = \ell_a t_a[1 + \beta_a(\ell_a/c_a)^4]$. Other instances, such as `Stockholm`, `Winnipeg`, and `Barcelona` are also studied in this paper, and have the same delay function. Their attributes (number of nodes, number of links, number of OD pairs, and number of destinations) are displayed in Table 1. These instances are available online [5].

5.3 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` [11], a freely available solver for convex programs.

Our implementation uses a more compact, but equivalent, formulation of (1–4), which represents the flows of all OD 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 and 73 smaller than the number of OD pairs.

Table 2 Optimal solutions

Instance	Optimal value	Solution time (s)
`Nine-node`	22.539181	<1
`Sioux Falls`	19.950794	22
`Stockholm`	Did not finish	>259,200

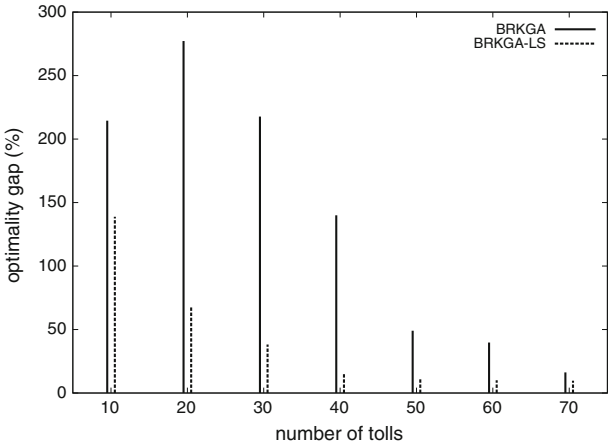

Fig. 4 Number of tolls installed versus optimality gap of results of the both variants of BRKGA (with and without local search) for the `Sioux Falls` instance. We tested installing 10, 20, 30, 40, 50, 60, and 70 tolls

The `cvxopt` solver was able to produce optimal values for only the two smallest instances shown in Table 2. On `Stockholm`, an instance with 416 nodes and 962 arcs, the solver did not terminate within three days of CPU time. Therefore, we did not attempt to optimally solve the two other larger instances with `cvxopt`.

5.4 Quality of the BRKGA solutions

We compare the best solution values and the optimality gap (when possible) obtained by two variants of our heuristic, BRKGA and BRKGA-LS. For each instance, we used different numbers of tolled arcs, varying from a few and up to a toll booth on every arc. For each number of tolled arcs, we ran both variants of the heuristic three times with different random seeds for 5,000 generations, but at most up to a time limit of one hour. The results represent the average of these runs. Each run of instance `Sioux Falls` took on average about 2 min of CPU time, while the runs on instance `Stockholm` took about 18 min. Runs for instances `Winnipeg` and `Barcelona` always terminated with the 60-min time limit. The number of generations for instance `Winnipeg` varied from 169 to 912, while for instance `Barcelona` this number varied from 240 to 1296. The BRKGA-LS variant spent between 50 and 70% of its time in the local search.

Figures 3 and 4 show computational results for instances `nine-node` and `Sioux Falls`, respectively. Since it was possible to optimally solve these instances, the plots

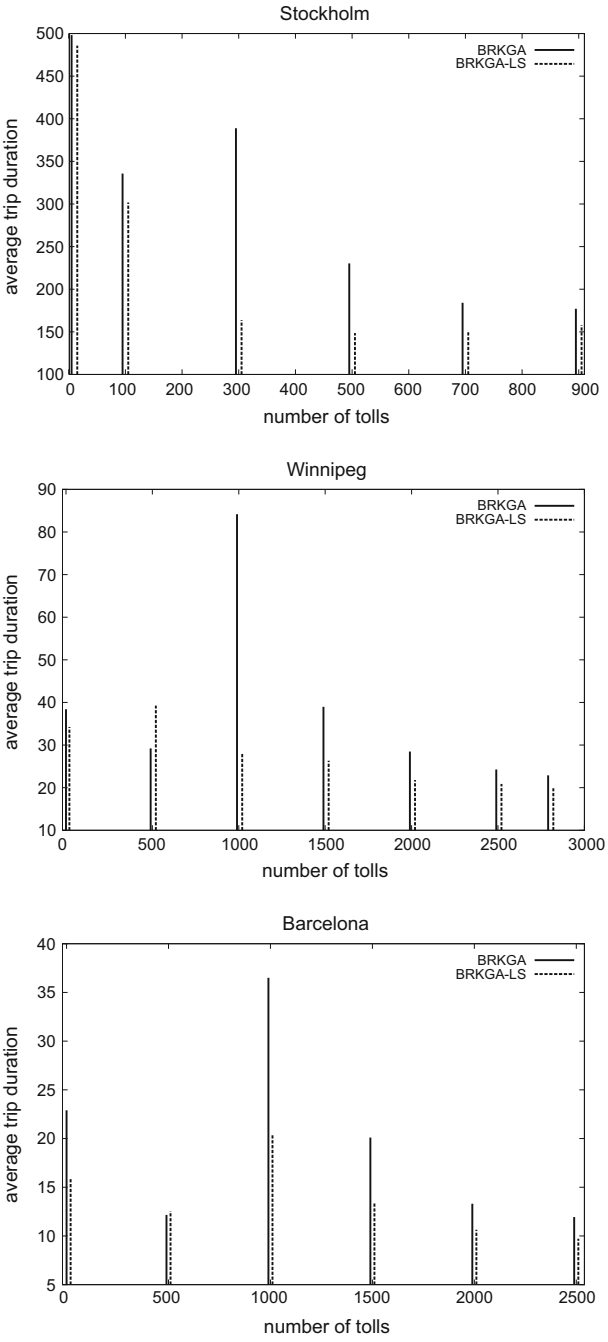

Fig. 5 Number of tolls installed versus quality of results of BRKGA and BRKGA-LS on instances `Stockholm` (*top*), `Winnipeg` (*middle*), and `Barcelona` (*bottom*)

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 (40 or more), the solution found by BRKGA-LS lies within 15% of the SO solution for the `Sioux Falls` instance.

Figure 5 shows computational results for instances `Stockholm`, `Winnipeg`, and `Barcelona`. For each instance, we present results found by BRKGA and BRKGA-LS.

The experiments show that BRKGA-LS finds better solutions than BRKGA with similar running times. For all tests performed, on only two cases was the average solution value found by BRKGA better than the corresponding value found by BRKGA-LS (`Winnipeg` with 500 tolls and `Barcelona` with 500 tolls).

For most of the instances, the quality of the results improves with larger toll sets. For BRKGA-LS, the solution value almost always decreases monotonically with an increasing number of tolls, while for BRKGA, even more variation is observed.

Given that in almost all cases BRKGA-LS finds better solutions as the number of installed tolls increases, one can choose a trade-off between the number of tolled links and the quality of the solution.

6 Conclusions

In this paper, we adapted the evolutionary algorithm introduced in [10] to a transportation problem. We were only able to solve small instances to optimality with `cvxopt`, a solver for convex optimization. We proposed and evaluated two variants of a BRKGA, one with and the other without local search in the decoder. By means of computing a solution that minimizes the average delay of the trips, 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 both genetic algorithms obtain solutions of good quality for small and large instances. In almost all cases, the quality of the solution improves with the increase in the number of tolls deployed. However, a large number of tolls may be not suitable to apply, at this time, in a real-world situation. Thus, one can choose the number of tolls to be installed by considering the quality of solution achieved.

Acknowledgments Luciana S. Buriol and Marcus Ritt received support from the Brazilian National Council of Technological and Scientific Development (CNPq) under project number 483413/2009-7. Research by P. M. Pardalos has been supported by a grant from the University of Florida Center for Multimodal Solutions for Congestion Mitigation, United States Department of Transportation/Federal Highway Administration.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows—Theory, Algorithms, and Applications. Prentice-Hall, Englewood Cliffs (1993)
2. Arnott, R., Small, K.: The economics of traffic congestion. Am. Sci. **82**, 446–455 (1994)
3. Bai, L.: Computational methods for toll pricing models. Ph.D. thesis, University of Florida, Gainesville, Florida (2004)

4. Bai, L., Hearn, D.W., Lawphongpanich, S. : Relaxed toll sets for congestion pricing problems. In: Hearn, D., Lawphongpanich, S., Smith, M. (eds.) Mathematical and Computational Models for Congestion Charging, Springer, Berlin (2006)
5. Bar-Gera, H.: Transportation networks test problems (2007). http://www.bgu.ac.il/~bargera/tntp
6. Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA J. Comput. **6**, 154–160 (1994)
7. Bureau of Public Roads: Traffic Assignment Manual. Tech. rep., US Dept. of Commerce, Urban Planning Division, Washington, DC (1964)
8. Buriol, L., Resende, M., Thorup, M.: Speeding up dynamic shortest-path algorithms. INFORMS J. Comput. **20**, 191–204 (2008). doi:10.1287/ijoc.1070.0231
9. Buriol, L.S., Hirsch, M.J., Pardalos, P., Querido, T., Resende, M.G., Ritt, M.: A hybrid genetic algorithm for road congestion minimization. In: Proceedings of the XLI Simpósio Brasileiro de Pesquisa Operacional, pp. 2515–2526 (2009)
10. Buriol, L.S., Resende, M.G.C., Ribiero, C.C., Thorup, M.: A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. Networks **46**, 36–56 (2005)
11. Dahl, J., Landenberghe, L.: CVXOPT (2005). http://abel.ee.ucla.edu/cvxopt
12. Dial, R.B.: Minimal-revenue congestion pricing part I: a fast algorithm for the single origin case. Transp. Res. B **33**, 189–202 (1999)
13. Dial, R.B.: Minimal-revenue congestion pricing part II: an efficient algorithm for the general case. Transp. Res. B **34**, 645–665 (1999)
14. Ericsson, M., Resende, M.G.C., Pardalos, P.M.: A genetic algorithm for the weight setting problem in OSPF routing. J. Combin. Optim. **6**, 299–333 (2002)
15. Florian, M., Hearn, D. et al.: Network equilibrium models and algorithms. In: Ball, M.O. (ed.) Network Routing, pp. 485–550. Elsevier Science, Amsterdam (1995)
16. Gonçalves, J., Resende, M.: Biased random-key genetic algorithms for combinatorial optimization. Tech. rep., AT&T Labs Research, Florham Park, NJ (2010). (http://www.research.att.com/~mgcr/doc/srkga.pdf). To appear in J. Heuristics
17. Hearn, D.W., Ramana, M.: Solving Congestion Toll Pricing Models. Equilibrium and Advances in Transportation Modeling. North-Holland, New York (1988)
18. Hearn, D.W., Ribera, J.: Bounded flow equilibrium by penalty methods. In: Proceedings of the IEEE International Conference on Circuits and Computers, pp. 162–164 (1980)
19. Kim, D., Pardalos, P.: A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure. Oper. Res. Lett. **24**, 195–203 (1999)
20. Lawphongpanich, S., Hearn, D.W.: An MPEC approach to second-best toll pricing. Math. Program. Ser. B **101**, 33–55 (2004)
21. LeBlanc, L.J., Morlok, E.K., Pierskalla, W.P.: An efficient approach to solving the road network equilibrium traffic assignment problem. Transp. Res. **9**, 309–318 (1975)
22. Reis, R., Ritt M., Buriol, L.S., Resende, M.G.C.: A biased random-key genetic algorithm for OSPF and DEFT routing to minimize network congestion. Int. Trans. Oper. Res. (2010, in press)
23. Shepherd, S., Sumalee, S.: A genetic algorithm based approach to optimal toll level and location problems. Netw. Spatial Econ. **4**(2), 161–179 (2004)
24. Spears, W., DeJong, K.: On the virtues of parameterized uniform crossover. In: Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 230–236 (1991)
25. Tsekeris, T., Voß, S.: Design and evaluation of road pricing: state-of-the-art and methodological advances. Netnomics **10**, 5–52 (2009)