Perea Rojas Marcos, F.; Menezes, MBC.; Mesa, JA.; Rubio-Del-Rey, F. (2020). Transportation infrastructure network design in the presence of modal competition: computational complexity classification and a genetic algorithm. Top. 28(2):442-474. https://doi.org/10.1007/s11750-019-00537-x

Noname manuscript No.
(will be inserted by the editor)

Transportation infrastructure network design in presence of modal competition: computational complexity classification and a genetic algorithm

Federico Perea* · **Mozart B.C. Menezes** · **Juan A. Mesa** · **Fernando Rubio-del-Rey**

Received: date / Accepted: date

Abstract In this paper we analyze the computational complexity of transportation infrastructure network design problems, in the presence of a competing transportation mode. Some of these problems have previously been introduced in the literature. All problems studied have a common objective: the maximization of the number of travelers using the new network to be built. The differences between them are due to two factors. The first one is the constraints that the new network should satisfy: 1) budget constraint, 2) no-cycle constraint, 3) both constraints. The second factor is the topology of the network formed by the feasible links and stations: 1) a general network, 2) a forest. By combining these two factors, in total we analyze six problems, five of them are shown to be NP-hard, the sixth being trivial. **Due to the NP-hardness of these problems, a genetic algorithm is proposed.** Computational experiments show the applicability of this algorithm.

Keywords Networks/graphs · Transportation · Computational complexity · Genetic Algorithms.

Federico Perea (corresponding author)
Departamento de Estadística e Investigación Operativa Aplicadas y Calidad, Universitat Politècnica de València (Spain)
Tel.: +34 963877000
Fax: +34 963877000
E-mail: perea@eio.upv.es

Mozart Menezes,
NEOMA Business School Department of Information Systems, Supply Chain Management and Decision Sciences 1 Rue du Maréchal Juin, 76130 Mont-Saint-Aignan, (France), E-mail: mozart.menezes@neoma-bs.fr

Juan A. Mesa
Departamento de Matemática Aplicada II, Universidad de Sevilla (Spain), E-mail: jmesa@us.es

Fernando Rubio-del-Rey
E-mail: fernando.rubio.rey@outlook.es

1 Introduction

This paper presents strategic level optimization models for transportation infrastructure design in competition with alternative transportation modes, and discusses on their computational complexity. After a discussion of some properties of the problems introduced, we present a metaheuristic **approach, which** is assessed based on running time and quality of the solutions returned.

The transportation systems are a crucial issue for a well organized society. The sequential transportation planning process starts with infrastructure network design. Thus, the efficiency and other properties of a transportation system strongly depend on the infrastructure network design. The following steps are: line planning, timetabling, and rolling stock and crew scheduling ([2], [5], [23]). Often, the infrastructure network construction requires a large investment, and the decisions involved have a strategic character, deserving to be tackled by analytical models, which can be done with the help of operations research, see [15].

Generally, there are several agents involved in the transportation system design: regulation and transportation agencies, users, network constructors and operational companies. Although these agents' objectives are sometimes conflicting, a future high utilization of the infrastructure is a common goal for all of them.

Usually, when a transportation network is going to be introduced, other transportation systems are already functioning. This fact makes the problem even more challenging, as the utilization of the new infrastructure heavily depends on the benefits that such new infrastructure provides to the users, in comparison with those offered by competing transportation networks.

Network design problems deal with the selection of (undirected) edges or (directed) arcs, in accordance with an objective function, regarding some origin-destination flows requirements. The academic literature related with network design is spread among several areas. For instance, the location of subnetworks on networks is a well studied field. Two reviews of the research conducted in the 1990s and before are [17] and [1]. More recent references on the location of subtrees on trees are [25] and specially the review [21]. During the last years, the research on network design expanded according to the area of application. Regarding transportation, the interested reader is referred to the survey [11].

Often, users choose transportation systems based on travel times. This induces decision makers to select a subnetwork, where the total travel time is minimal. In other words, given a set of feasible links and stations that can be built, the problem consists of selecting subset of them, so that some objective function, which takes into account time to complete origin/destination pair paths, is optimized. This is the topic dealt with in [22]. Mathematical programming models and heuristic algorithms for solving the infrastructure transit network design problem, with mode choice, can be found in [16] and [6].

In this paper, the objective function considered is the number of potential travelers that will use the network to be built. In other words, the goal is to find a set of feasible links and stations such that the resulting network maximizes the future utilization of traffic, subject to some very general constraints. In some formulations we require the solution to be a set of tree-networks (i.e., a forest network) and/or to satisfy a budget constraint. Potential travelers will use the new transportation network if this network offers them a higher utility than other transportation alternatives.

We analyze problems that take into account time-aspects for consumer-choice of transportation systems, and we discuss on their complexity. These problems are proven to be NP-hard. We also show that relaxing some of their constraints does not necessarily make them less complex.

The complexity of these problems, both in computation and structure, suggests the application of heuristics and metaheuristics. In this paper we propose a genetic algorithm. Genetic algorithms (GA) are among the most commonly used metaheuristics for solving combinatorial optimization problems. As examples of applications of GA within the specific area of transportation network design, we can mention the works [4] and [3]. These two papers underline the effectiveness of the genetic algorithm methodologies for the Transit Routing Problem and the Transit Scheduling Problem, with satisfactory results. [18] also address a transit network design problem by means of genetic algorithms. Two different algorithms are proposed to improve the initial solution obtained, both based on a greedy procedure. [24] proposed a GA for a distribution network problem. [26] formulated another genetic algorithm for a robust network design problem. The reader may note that the literature on transportation optimization and genetic algorithms is very extensive, and a complete review of it is out of the scope of this paper.

In short, the contribution of this paper is threefold:

1. **We prove the supermodularity of the estimated ridership objective function which will be useful for characterizing the computational complexity of the non-cyclic constrained problem.**
2. **We formally classify the complexity status of several problems as NP-hard. These problems have a common aim: finding a network that maximizes the trip coverage in competition with alternative transportation modes**.
3. We propose a Genetic Algorithm that improves upon the other metaheuristic introduced in the literature **(a GRASP algorithm in [6])** and shows satisfactory results in terms of speed and quality.

The rest of the paper is organized as follows. In Section 2, we define the problems to be studied. The objective function of these problems is further analyzed in Section 3. Section 4 shows the main theoretical results of this paper, regarding the computational complexity of the problems introduced. In Section 5 we describe the genetic algorithm proposed. In Section 6, the performance of the genetic algorithm is tested in a computational experience, and some of

its parameters are calibrated. The paper finishes with some conclusions and suggests directions for future research.

2 The railway network design problem

In this section, we formally describe the transportation network design problems under consideration. We assume that there is a set of feasible links (edges) and stations (nodes) that can be built. The objective is to select a subset of these links and stations, which will constitute the *new* network, contained in the underlying network. The new network will compete with existing transportation systems, in the sense that travelers will choose the new network if and only if such network offers the highest utility. Therefore, the objective is for the new network to attract as many travelers as possible.

We will define different problems, depending on the structure of the new network (forest or general graph), and on the presence or not of budget constraints. To illustrate, the new network could be a railway network. We want to remark that other transportation modes can fit within the problems proposed, which can be formally described from the following input data:

1. A set $N = \{v_1, v_2, \ldots, v_n\} \subset \mathbb{R}^2$ of potential sites for railway stations is given (cities or junctions). We will use the indexes i, j, k, p, q to denote nodes. Node v_i is located in $(v_i^1, v_i^2) \in \mathbb{R}^2$, and is denoted by means of its subindex i, whenever this creates no confusion.
2. A set $E \subset N \times N$ of (undirected) edges linking the elements in N is known, yielding the undirected connected graph $G = (N, E)$, of which links (edges) and stations (nodes) are to be selected. The set of (directed) arcs associated to E is denoted as $A(E)$, or just A when no confusion may arise. Let ℓ_e denote the length of edge $e \in E$. We say that S is a subgraph of G if both the node set and the link set of S are subsets of those of G, **and S itself is a graph**. For the sake of clarity, and abusing notation, in this paper $S \subset G$ denotes that S is a subgraph of G. Let $N(S)$ and $E(S)$ denote the node set and the edge set of S for any $S \subset G$, respectively. Let $d^S(p, q)$ be the shortest-path distance between v_p and v_q on the network S. Let $\mathcal{E}$ be the set of all sub-graphs of G that are forests, i.e. contain no cycles.
3. Let $c_i \in \mathbb{R}^+$ be the cost of building a station at node $v_i \in N$, and let $c_{ij} \in \mathbb{R}^+$ be the cost of building the link $\{v_i, v_j\} \in E$ for the railway network (we will assume links can be used in both directions). A bound $C_{\max} \in \mathbb{R}^+$ on the available budget is also given. In our problems, if $\{v_i, v_j\} \in E$ is built, so should stations $v_i, v_j \in N$.
4. Let $W \subseteq N \times N$ be the set of origin/destination (O/D) pairs. Parameter g_{pq} is the number of travelers of O/D pair (v_p, v_q), and u_{pq}^{ALT} is a parameter of the problem that defines the utility for this O/D pair using the alternative transportation system. Such **utility takes into account** several characteristics such as travel time, price of trip, comfort, etc. We assume that the demand is concentrated at nodes, and that a demand pair cannot

use the railway network $S \subset G$ if such two points are not connected in it (that is, we do not allow mode change during a trip).

5. For every $S \subset G$, u_{pq}^S defines the utility of the railway network S for O/D pair (v_p, v_q). This utility depends on the shortest path between v_p and v_q on S. We will assume that the utility is monotone, in the sense that $u_{pq}^S \leq u_{pq}^T$ for all $S \subset T$. If there is no path between v_p and v_q in $S \subset G$, we set $u_{pq}^S = -\infty$.

Remark 1 Because the utility of an O/D pair is a function of the paths between the two points, if $S \subseteq T$ are forests, and both $v_p, v_q \in N(S)$, then $u_{pq}^S = u_{pq}^T$. This is due to the fact that, the only path between these two nodes in S and T is the same.

Given $S \subset G$, the proportion of travelers of the O/D pair (v_p, v_q) that will use the railway network S, depends on the utility of the railway network defined by S, and that of the alternative mode for this O/D pair, and is estimated by means of $\varphi(u_{pq}^S - u_{pq}^{ALT})$, where $\varphi : \mathbb{R} \to [0, 1]$ is a function that satisfies:

1. $\lim_{x \to -\infty} \varphi(x) = 0$ (no traveler of (v_p, v_q) will use the railway network, as u_{pq}^S is much less than u_{pq}^{ALT}). This implies that, if $S \subset G$ has no path between v_p and v_q, and therefore $u_{pq}^S = -\infty$, then none of this O/D pair travelers will use the railway network.
2. $\lim_{x \to +\infty} \varphi(x) = 1$ (all travelers of (v_p, v_q) will use the railway network, as u_{pq}^S is much higher than u_{pq}^{ALT}).
3. φ is non-decreasing so, assuming the utility of the alternative mode is fixed, the higher the utility of the railway network, the higher the proportion of users that will take this mode.

Because u_{pq}^{ALT} is fixed in our problems, and for the sake of readability, from now on we will write $\varphi(u_{pq}^S)$ instead of $\varphi(u_{pq}^S - u_{pq}^{ALT})$, whenever this creates no confusion. To illustrate, two examples of such φ functions are:

1. A logit function defined as

$$\varphi(u_{pq}^S) = \frac{1}{1 + \gamma_1 e^{-\gamma_2 (u_{pq}^S - u_{pq}^{ALT})}},$$

   where $\gamma_1, \gamma_2 > 0$ are parameters that need to be calibrated and depend on the instance.
2. A binary function defined as

$$\varphi(u_{pq}^S) = 1 \text{ if } u_{pq}^S \geq u_{pq}^{ALT},\ \varphi(u_{pq}^S) = 0 \text{ if } u_{pq}^S < u_{pq}^{ALT}.$$

   Using this function, $\varphi(u_{pq}^S) = 1$ means that the O/D pair is *covered*.

Whereas the logit function gives a smooth estimate of the proportion of travelers that will use the railway network according to utilities, and can take any real number in the interval (0,1), the binary function assumes that either all

travelers of a given O/D pair will use railway network ($\varphi(u_{pq}^S) = 1$) or none ($\varphi(u_{pq}^S) = 0$). The logit function was used by [16] and [20], whereas [14] and [6] make use of the binary function to estimate ridership.

From this function φ, the estimated ridership of the railway network can be defined, that is, an estimation of the number of travelers who will use it.

Definition 1 $Z(S) := \sum_{(v_p,v_q)\in W} g_{pq}\varphi(u_{pq}^S)$ is the estimated ridership of railway network S.

The cost of the railway network will also be used in the rest of the paper, and is defined as:

Definition 2 $C(S) := \sum_{v_i\in N(S)} c_i + \sum_{\{v_i,v_j\}\in E(S)} c_{ij}$ is the cost of railway network S.

We now define the three railway network design (RND) problems that are the **topic** of our paper. The first one (RND1) was studied in [6], with binary ridership function, and consists of finding a subnetwork of G that maximizes $Z(\cdot)$, subject to a budget constraint. The other two problems (RND2, RND3) impose for the railway network to have no cycles. The difference between the two latter problems is the presence or not of budget constraints. Although the no-cycle constraint could be considered artificial, we were inspired by the fact that new railway networks usually are a tree. To illustrate, consider the Spanish high-speed railway network, which started functioning in 1992. After more than 25 years functioning, by the time this paper was written, this network had 30 stations and some 2400 km, but no cycles at all. Other examples are the French or the Italian high-speed railway networks. One could say that, for a network to have cycles is a sign of *maturity*. We now formally describe the three problems.

1. RND1:
$$\begin{aligned} &\max Z(S) \\ &\text{s.t.: } S \subseteq G, \\ &\quad C(S) \leq C_{\max}. \end{aligned} \tag{1}$$
This problem consists of finding the subnetwork of G, satisfying the budget constraint, that maximizes the estimated ridership.
2. RND2:
$$\begin{aligned} &\max Z(S) \\ &\text{s.t.: } S \subseteq G, \\ &\quad C(S) \leq C_{\max}, \\ &\quad S \text{ has no cycles.} \end{aligned} \tag{2}$$
This problem consists of finding the forest contained in G, satisfying the budget constraint, that maximizes the estimated ridership.
3. RND3:
$$\begin{aligned} &\max Z(S) \\ &\text{s.t.: } S \subseteq G, \\ &\quad S \text{ has no cycles.} \end{aligned} \tag{3}$$
This problem consists of finding the forest contained in G that maximizes the estimated ridership (without any budget constraint).

3 Properties of the objective function

In this section, we analyze some properties of the ridership function, which will help us later in our computational complexity results. The next lemma states three easy-to-prove properties: the ridership function $Z(S)$ is positive, monotone non-decreasing, and separable by connected components.

Lemma 1 *1.* $Z(S) \geq 0, \ \forall \ S \subset G$.
2. $Z(S), \ S \in \mathcal{E}$, *is non-decreasing in* S.
3. If S *consists of* m *connected components* $\{S_1, ..., S_m\}$, *then* $Z(S) = \sum_{k=1}^{m} Z(S_k)$.

Proof It is trivial that the ridership function $Z(\cdot)$ is non-negative. The monotonicity property is true because, when adding an edge to a network, the utility using the railway does not decrease for any O/D pair (as the lengths of the shortest paths do not increase when a new edge is added), and therefore the number of travelers using the railway network does not decrease either. The separability property follows from the fact that $\varphi(u_{pq}^{S}) = 0$ if there is no path in S between v_p and v_q.

We now prove that our objective function satisfies the supermodularity property in problems RND2 and RND3.

Theorem 1 Z *is supermodular on the set of forests* $\mathcal{S}$.

Proof The function $Z(S)$ is supermodular in S if and only if (see [19])

$$Z(S \cup \{e\}) - Z(S) \leq Z(T \cup \{e\}) - Z(T), \ \ \forall \ S \subseteq T \text{ and } e = \{e_a, e_b\} \notin E(T), \tag{4}$$

where $S \cup \{e\}$ is the graph resulting when adding edge e (and its endnodes, if necessary) to graph S, for all $S \subset T$ and $e \notin E(T)$. Similarly, we denote $S \setminus \{e\}$ the network resulting when removing edge e from network S, and possibly one or both of its endnodes. In our case, it is imperative that $S, T, S \cup \{e\}$ and $T \cup \{e\}$ are elements of $\mathcal{E}$ (forests).

Without loss of generality, assume that T is a forest with m trees. Let us divide the proof into three cases.

• Case *i*: $T \cup \{e\}$ is a forest with $m + 1$ trees.

This is a very simple case. It implies that,

$$Z(S \cup \{e\}) - Z(S) = Z(T \cup \{e\}) - Z(T), \tag{5}$$

because if edge e induces a new subtree in T then it will induce a subtree in $S \subseteq T$. Thus, both sides of the equality reduce to $Z(\{e\})$, see Lemma 1.

• Case *ii*: $T \cup \{e\}$ is a forest with m trees.

Let T_k be the tree of T that edge e connects to. Note that $e \notin E(T_k)$. Without loss of generality, let node e_b be a leaf in $T_k \cup \{e\}$. Let S_e be the tree

of $S \cup \{e\}$ that edge e belongs to. That is, $e \in E(S_e)$. Note that $E(S_e)$ may be equal to $\{e\}$ if edge e is a tree by itself in $S \cup \{e\}$.

$$\begin{aligned}
Z(T \cup \{e\}) - Z(T) &= Z(T_k \cup \{e\}) - Z(T_k) \\
&= \sum_{p \in N(T_k \cup \{e\})} (g_{p,e_b} \varphi(u_{p,e_b}^{T_k \cup \{e\}}) + g_{e_b,p} \varphi(u_{e_b,p}^{T_k \cup \{e\}})) \quad (6) \\
&\geq \sum_{p \in N((S \cap T_k) \cup \{e\})} (g_{p,e_b} \varphi(u_{p,e_b}^{T_k \cup \{e\}}) + g_{e_b,p} \varphi(u_{e_b,p}^{T_k \cup \{e\}})) \quad (7) \\
&\geq \sum_{p \in N(S_e)} (g_{p,e_b} \varphi(u_{p,e_b}^{T_k \cup \{e\}}) + g_{e_b,p} \varphi(u_{e_b,p}^{T_k \cup \{e\}})) \quad (8) \\
&= \sum_{p \in N(S_e)} (g_{p,e_b} \varphi(u_{p,e_b}^{S_e}) + g_{e_b,p} \varphi(u_{e_b,p}^{S_e})) \quad (9) \\
&= Z(S_e) - Z(S_e - \{e\}) = Z(S \cup \{e\}) - Z(S). \quad (10)
\end{aligned}$$

The equality in (6) follows from the fact that the difference in ridership between $T_k \cup \{e\}$ and T_k is given by the traffic linked by e; (7) follows from $(S \cap T_k) \cup \{e\} \subseteq T_k \cup \{e\}$ and the monotonicity of $Z(S)$ (Lemma 1); and (8) follows from the monotonicity property and the fact that $S_e \subseteq (S \cap T_k) \cup \{e\}$. We define the intersection of two graphs S and T as the graph in which all nodes and all edges belong to both S and T. (9) is true because the path from $v_p \in N(S_e)$ to e_b is the same in $T \cup \{e\}$ and in S_e. The equalities in (10) are a direct consequence of the definition of Z.

• Case *iii*: $T \cup \{e\}$ is a forest with $m - 1$ trees.

In this case, edge e is joining two trees in T. Let us call them T^+ and T^-. Assume $e_a \in T^+$ and $e_b \in T^-$. Let $S^+ = S \cap T^+$ and $S^- = S \cap T^-$. Then, from the separability of Z (see Lemma 1),

$$\begin{aligned}
Z(T \cup \{e\}) - Z(T) &= Z(T^- \cup T^+ \cup \{e\}) - (Z(T^-) + Z(T^+)) \\
&= \sum_{v_p \in N(T^-),\, v_q \in N(T^+)} g_{pq} \varphi(u_{pq}^{T^- \cup T^+ \cup \{e\}}) \\
&\quad + \sum_{v_p \in N(T^+),\, v_q \in N(T^-)} g_{pq} \varphi(u_{pq}^{T^- \cup T^+ \cup \{e\}}). \quad (11)
\end{aligned}$$

Let us now calculate $Z(S \cup \{e\}) - Z(S)$. Due to Lemma 1, this difference is equal to

$$Z(S^+ \cup S^- \cup \{e\}) - (Z(S^+) + Z(S^-)). \quad (12)$$

This difference is calculated according to three possible cases:

1. $S^+ \cup S^- \cup \{e\}$ consists of three trees. In such case, (12) is

$$Z(\{e\}) = g_{e_a e_b} \varphi(u_{e_a e_b}^{\{e\}}) + g_{e_b e_a} \varphi(u_{e_b e_a}^{\{e\}})$$

2. $S^+ \cup S^- \cup \{e\}$ consists of two trees. Without loss of generality assume such two trees are $S^+ \cup \{e\}$ and S^-. In this case, (12) is

$$Z(S^+ \cup \{e\}) - Z(S^+) = \sum_{v_p \in N(S^+)} g_{pe_b} \varphi(u_{pe_b}^{S^+ \cup \{e\}}) + g_{e_b p} \varphi(u_{e_b p}^{S^+ \cup \{e\}})$$

3. $S^+ \cup S^- \cup \{e\}$ is one tree. In this case (12) is

$$\sum_{v_p \in N(S^-),\, v_q \in N(S^+)} g_{pq} \varphi(u_{pq}^{(S^-) \cup (S^+) \cup \{e\}}) + \sum_{v_p \in N(S^+),\, v_q \in N(S^-)} g_{pq} \varphi(u_{pq}^{(S-) \cup (S^+) \cup \{e\}})$$

In any of the three cases, and due to the monotonicity of Z, (11) $\geq$ (12). Therefore,

$$Z(T \cup \{e\}) - Z(T) \geq Z(S \cup \{e\}) - Z(S).$$

Remark 2 The supermodularity property does not hold for problem RND1, which is shown in Example 1.

Example 1 Let G be the complete 3-node graph, in which all edges have cost zero and length 1. Let $g_{1,2} = 1$, and let $g_{pq} = 0$ for any other O/D pair. Define the utility of O/D pair (v_p, v_q) using the road network as $u_{pq}^{ALT} = -d^G(p, q)$, where $d^S(p, q)$ is the distance between v_p and v_q on the network S, for all $S \subset G$. It is easy to see that $u_{1,2}^{ALT} = -1$. For any $S \subset G$, define the utility of O/D pair (v_p, v_q) using the railway network S as $u_{pq}^S = -0.1 d^S(p, q)$.

Consider φ as the binary function, that is, $\varphi_{pq}(u_{pq}^S) = 1$ if $u_{pq}^S > u_{pq}^{ALT}$, and zero otherwise.

Let $S = \emptyset, T = (\{v_1, v_2, v_3\}, \{\{v_1, v_3\}, \{v_2, v_3\}\}), e = \{v_1, v_2\}$. It follows that $u_{1,2}^{S \cup \{e\}} = u_{1,2}^{T \cup \{e\}} = -0.1,\ u_{1,2}^S = -\infty,\ u_{1,2}^T = -0.2$.

Then, $Z(S \cup \{e\}) = g_{1,2} \varphi(u_{1,2}^{S \cup \{e\}}) = 1$. Analogously, $Z(S) = 0,\ Z(T) = Z(T \cup \{e\}) = 1$. This proves that, for some $S \subset T$,

$$Z(S \cup \{e\}) - Z(S) > Z(T \cup \{e\}) - Z(T),$$

and therefore $Z(\cdot)$ is not supermodular for RND1.

4 Computational complexity

In this section we discuss on the theoretical complexity of the three problems proposed.

4.1 Problems RND1 and RND2 complexity

We now prove that both RND1 and RND2 are NP-hard.

Proposition 1 *RND1 is NP-hard.*

The proof is similar to the one found in [14], **only changing the way in which the input data are organized,** and is included in the Appendix for the sake of completeness. **The problem treated in this paper is slightly different from the problem treated in [14]. The main difference is that we now consider that the proportion of passengers attracted by the railway network could be any function taking its values in the interval $[0, 1]$, whereas in [14] this function was binary (for each OD pair, either all passengers were attracted or none).** The same proof could be used to show that RND2 is NP-hard, as the solution network has no cycles.

Corollary 1 *Problem RND2 is NP-hard.*

The proof to this result is in the Appendix.

4.2 Complexity of RND3

Problem RND3 consists of maximizing the estimated ridership so that all nodes have a station and are part of the system, and there are no cycles. That is, we want to find $E_t \subset E$ such that $(N, E_t) \in \mathcal{E}$, (N, E_t) maximizes the ridership. Therefore, RND3 is:

$$\underset{E_t \subset E:(N,E_t)\in\mathcal{E}}{\text{Maximize}} \; Z(E_t) = \sum_{(v_p,v_q)\in W} g_{pq} \;\; \varphi(u_{pq}^{E_t}). \tag{13}$$

Note that, for the sake of simplicity, we refer to a graph (N, E_t) only by its set of edges (E_t), whenever this creates no confusion.

The following lemma proves that, a forest that maximizes ridership (solution to problem RND3) is a spanning tree.

Lemma 2 *For any disconnected $E_f : (N, E_f) \in \mathcal{E}$, there exists $(N, E_t) \in \mathcal{E}$ such that (N, E_t) is a tree and $Z(E_f) \leq Z(E_t)$.*

Proof Assume that E_f consists of m trees, $E_f^1, ..., E_f^m$. Without loss of generality, assume that E_f^1 and E_f^2 can be connected without creating a cycle, and let $\bar{E}$ be the tree we obtain by connecting E_f^1 and E_f^2. It is obvious that such two subtrees exist since G is connected. Connect the two subtrees, and call the new forest E_f^+, which consists of $m-1$ trees, $\bar{E}, E_f^3, ..., E_f^m$. Because $u_{pq}^{E_f^1} = u_{pq}^{E_f^+} \;\forall\; v_p, v_q \in N(E_f^1)$, and $u_{pq}^{E_f^2} = u_{pq}^{E_f^+} \;\forall\; v_p, v_q \in N(E_f^2)$ (see Remark 1) we have,

$$\begin{aligned} Z(E_f^+) - Z(E_f) &= Z(\bar{E}) + \sum_{k=3}^{m} Z(E_f^k) - \sum_{k=1}^{m} Z(E_f^k) && (14)\\ &= Z(\bar{E}) - (Z(E_f^1) + Z(E_f^2)) && (15)\\ &= (Z(E_f^1) + Z(E_f^2) + \sum_{(v_p,v_q)\in E^*} g_{pq}\varphi(u_{pq}^{E^*})) && (16)\\ &- (Z(E_f^1) + Z(E_f^2)) \geq 0, && (17) \end{aligned}$$

where $E^* = \{\{v_p, v_q\} \subset N(\bar{E}) : (v_p \in N(E_f^1), v_q \notin N(E_f^1))$ or $(v_p \notin N(E_f^1), v_q \in N(E_f^1))$ or $(v_p \in N(E_f^2), v_q \notin N(E_f^2))$ or $(v_p \notin N(E_f^2), v_q \in N(E_f^2))$ or $(v_p, v_q \notin (N(E_f^1) \cup N(E_f^2)))\}$. The process above can be repeated, until a spanning tree is formed.

Recall that minimizing a submodular function is equivalent to maximizing a supermodular function. Problem RND3 seems to have nice properties that suggest that it is easy to solve: after all, minimizing a submodular function is not NP-hard, and the minimum (modular) spanning tree is an easy problem that is solved with a greedy-type algorithm (Kruskal's Algorithm). In fact, minimizing a submodular function without constraints is strongly polynomially solvable (see [9] and [10]). Matroidal constraints can be added without increasing the problem's complexity, as long as further properties of the set function are present, such as symmetry (see [9] and [10]). Unfortunately, our objective function does not have those properties. On the contrary, $RND3$ is an NP-hard problem. We state this with the following proposition.

Proposition 2 *Problem $RND3$ belongs to the class of NP-hard problems.*

Proof The proof follows from [8]. In that paper, the authors show that minimizing a submodular set function subject to the solution being a spanning tree is NP-hard.

Further discussion on the hardness of some well known submodular minimization problems with extra constraints can be found in [12] and [19].

4.3 When graph G is a forest

After a careful look at the proof of Proposition 1, one can derive that both RND1 and RND2 are still NP-hard, even when the underlying graph G is a forest. This comes from the fact that the graph used in this proof is a tree. On the other hand, assuming this topology for the underlying graph, problem RND3 is trivial, because the complete graph G is a solution to it.

Corollary 2 *When the graph of potential links and stations is a forest, both RND1 and RND2 are NP-hard, and RND3 is trivial.*

5 A Genetic Algorithm

Due to the complexity of these problems, in this section we give an efficient procedure to solve them. More specifically, we will propose an efficient procedure to find feasible solutions to RND1, since this is the problem most commonly found in practice, out of the problems presented in this paper. Adaptations of this algorithm to RND2 and RND3 are straightforward, **and will be explained at the end of this section**.

Genetic Algorithms (GA) are among the most generally used metaheuristics for solving combinatorial optimization problems. Their functioning is based on genomics and evolutionary processes in nature. The main idea of a genetic algorithm is to simulate the evolution of a population of individuals (in our case, individuals are feasible solutions to the problem). The aim is to obtain better and better populations, which implies better and better solutions.

The genetic algorithm first builds an initial population. Afterwards, crossover and mutation operations will be applied in order to improve this initial population. After each of these two operations, the population is updated in the following way: if one of the new solutions found is better than the worst element of the population I_w, then the new solution enters the population and I_w is removed. Algorithm 1 summarizes the GA proposed. The functions involved in it will be detailed in the rest of this section.

Data: An RND1 instance
Generate the initial population with $i_{\max}$ solutions;
while *There is time available* **do**
  Randomly select two elements of the population;
  Do the crossover operation;
  Update the population;
  Select an element of the population;
  Do the mutation operation with probability p_m;
  Update the population;
end
Result: The best element of the population.

Algorithm 1: Scheme of the genetic algorithm.

5.1 Initial population

The initial population is built by executing the constructive phase of the GRASP algorithm designed in [6] a number of times. A high-level description of how this algorithm works is explained here for the sake of completeness.

This GRASP algorithm is divided into two phases, which are repeated a number of times. At each iteration, we first construct a feasible solution in the construction phase. Such a phase begins with a randomly chosen edge. In the next step, a new edge is randomly selected out of a restricted candidate list, consisting of the edges that maximize the gain in the objective function when they are individually added to the one chosen before. Next we add one more edge to the previously chosen two edges in the same greedy way. This process is repeated until no more edges can be added without violating the budget constraint. All elements in the initial population are created this way.

In order to have a population consisting of different solutions, if the GRASP algorithm finds a solution that is already in the population, then this solution

is discarded and a new execution of the GRASP is run in order to find a new solution. The size of the initial population is a parameter of the algorithm, denoted by $i_{\max}$.

Two versions of the GRASP algorithm will be executed:

- One in which only connected networks are allowed (GRASP1).
- Another in which non-connected networks are allowed (GRASP2).

The difference between these two versions is that in the first one, the restricted candidate list consists only on the edges incident to the network built so far in the process that do not violate the budget constraint. The second version only imposes for an edge to be in the restricted candidate list that the inclusion of such edge does not violate the budget constraint.

The crossover and mutation operations of the rest of the algorithm will be the same, independently of the choice of GRASP algorithm for building the initial population.

5.2 Codification

The solutions found in the initial population have to be coded in such a way that the genetic algorithm operations can be applied. If $G = (N, E)$ is the graph of feasible links and stations and $|E| = m$, then a solution to RND1 will be uniquely characterized by a list of m zeroes and ones. If position k^{th} of this list is one, then the k^{th} edge of the underlying graph G is part of the solution. Otherwise, if the k^{th} entry of this list is zero, then the corresponding edge is not part of the solution.

To illustrate, consider $G = (N, E)$ with $E = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}\}$. The solution consisting of edges $\{1, 2\}$ and $\{2, 3\}$ is coded as $(1, 0, 1, 0)$. Each element of this vector is called a *bit*.

5.3 Crossover

The crossover operation is designed in the following **way**. Two solutions are randomly chosen in the current population, which will be the parents. Let us denote these parents as I_1 and I_2, and let us denote as I_1' and I_2' the offsprings resulting after crossing I_1 and I_2. We will consider in our experiments the single-point crossover and the two-point crossover.

- Single-point: an integer $p_c \in \{1, ..., m-1\}$ is randomly generated. Then, I_1' will take the first p_c bits of I_1 and the last $m - p_c$ bits of I_2. Conversely, I_2' will take the first p_c bits of I_2 and the last $m - p_c$ bits of I_1.
  Example: assume $I_1 = (1, 0, 1, 0)$ and $I_2 = (0, 1, 1, 0)$. Assume further that $p_c = 1$. Then, $I_1' = (1, 1, 1, 0)$ and $I_2' = (0, 0, 1, 0)$.
- Two-point: two integers $p_c \in \{1, ..., m-1\}$ and $p_c' \in \{p_c + 1, ..., m\}$ are randomly generated. Then, I_1' will take the first p_c bits of I_1, bits $p_c + 1$ to p_c' of I_2, and the last $m - p_c'$ bits of I_1. Conversely, I_2' will take the first p_c

bits of I_2, bits $p_c + 1$ to p'_c of I_1, and the last $m - p'_c$ bits of I_2. If $p'_c = m$, then we have the single-point crossover.
Example: assume the same I_1 and I_2 as before, and $p_c = 2, p'_c = 3$. Then, $I'_1 = (1, 0, 1, 0)$ and $I'_2 = (0, 1, 1, 0)$. Note that, in this case, the offsprings are the same solutions as the parents.

In case the resulting offspring is "too expensive", i.e., it is not a feasible solution because it exceeds the budget, then its edge (and the corresponding stations) that most reduces the total cost is removed. This procedure is repeated until a solution that does not violate the budget constraint is obtained.

In case the resulting offspring is "too cheap", i.e., there is enough remaining budget to include more edges, then the solution is augmented following the same GRASP methodology to build the initial population.

After the crossover operation, if I'_1 improves the worst element of the population I_w, then the population is updated by including I'_1 and removing I_w. The same is repeated with I'_2.

5.4 Mutation

First, one solution I of the current population is randomly selected. The mutation operation consists of randomly selecting a bit of I that has value zero (in position b_0), and randomly selecting a bit of I that has value one (in position b_1). Then, both bits are exchanged with a certain probability and the resulting solution is denoted as I'.

If the resulting solution is "too expensive" or "too cheap", the solution is corrected following the same procedures defined before for the crossover operation.

Example: $I = (1, 0, 1, 0)$. b_0 is randomly selected, either 2 or 4 (the bits that are equal to zero). Assume $b_0 = 2$. Then, b_1 is randomly selected, either 1 or 3 (the bits that are equal to one). Assume $b_1 = 3$. Then, we exchange the positions $b_0 = 2$ and $b_1 = 3$, and the new element is $I' = (1, 1, 0, 0)$.

After the mutation operation, the population is updated by including I' and removing I. This update is only different in case I is the best element of the population, in which case I' substitutes I in the population only if I' improves I (that is to say, the best element found so far remains in the population).

5.5 Extensions to RND2 and RND3

The GA presented before for RND1 can easily be extended to RND2 and RND3, as follows:

RND2 **In the constructive phase, a new arc is only added if it does not produce any cycle. The crossover and mutation operations are only performed if the resulting networks have no cycles.**

RND3 **The same as for RND2 but, in this case, the budget is not a limitation. So the constructive phase will continue until no more arcs can be added without provoking a cycle.**

6 Computational experience

This section summarizes a computational experience over a set of randomly generated instances. We solve each instance for the three problems introduced in this paper. RND1 is solved by means of the MILP in the appendix and the genetic algorithm presented in Section 5, for different values of its parameters. RND2 and RND3 are only solved by means of the genetic algorithms, as the no-cycle constraints would make the model intractable even for small instances.

6.1 Instance generation

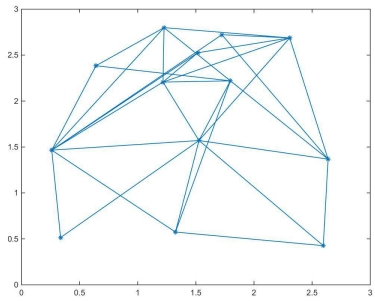
In these experiments, we consider four basic configurations for the underlying graph G: 3x3, 5x2, 4x3, 5x3, where $n \times m$ means that a rectangular grid with n horizontal cells, and m vertical cells, is created, the sides of each cell being of unitary length. For each grid, there are c cells that generate a higher number of trips. We tested three values for this parameter, $c \in \{0, 1, 2\}$, and we call these cells *centers*. Such attraction centers are randomly chosen in the grid.

One node is randomly located inside each cell in the grid, imposing that such node is not too close to its cell sides. To illustrate, the node located in the cell delimited by coordinates (0,0), (0,1), (1,0), (1,1) has coordinates $(0.5 + \mathcal{U}(-0.25, 0.25), 0.5 + \mathcal{U}(-0.25, 0.25))$, where $\mathcal{U}(a, b)$ denotes a uniform random distribution between a and b. Besides, inside the center cells four more nodes are located. Therefore, the $n \times m$ configuration with c centers consists of $n \cdot m + 4c$ nodes. The edge set of each configuration is built by joining the nodes of neighboring cells with probability $1/2$. Afterwards, the neighbors of neighbors are directly linked with probability $2/7$, the resulting graph being G. These probabilities were chosen so that the resulting underlying networks were not too dense nor too sparse, mimicking the network density usually found in practice. Although most of the railway networks can be represented by planar graphs we have considered a more general setting allowing edge crossings. Each combination of grid and number of centers was randomly replicated 5 times, therefore having in total $4 \times 3 \times 5 = 60$ different instances. An example of an underlying network generated for these experiments is shown in Figure 1.

We now show two indicators that asses the connectivity and the density of the underlying networks. The connectivity of the arising networks can be measured by means of the connectivity indicator defined as the number of edges divided by the number of nodes:

$$\beta = \frac{|E|}{|N|} \tag{18}$$

Fig. 1 Example of an underlying graph: grid 3x3, one center.



The average value of such β parameter equals 2.29. If we separate by configuration, the average connectivity indicator results as expressed in Table 1.

The density of the networks is assessed by means of the number of edges divided by the number of edges of the complete graph, following this formula:

$$\gamma = \frac{|E|}{n(n-1)/2} \tag{19}$$

The average value of such γ parameter equals 0.32. The average results per configuration are shown in Table 1. Since the range of the connectivity and density indicators for non-planar networks are $[(n-1)/n, (n-1)/2]$ and $[2/n, 1]$, respectively, we may state that the underlying networks proposed are not too connected nor too dense, as usually happens in practice.

Table 1 Connectivity and density parameters of the underlying networks randomly generated for the experiments.

Configuration	Connectivity	Density
3x3	2,36	0,39
4x3	2,33	0,31
5x2	2,28	0,34
5x3	2,18	0,24
Overall	2,29	0,32

The cost of building a station in node v_i, denoted as c_i, is randomly generated as a $\mathcal{U}(7 \cdot 10^5, 13 \cdot 10^5)$, meaning that on average, each station costs one million monetary units.

The utility of the road network for each OD-pair (p, q) is assumed to be 2.21 times the Euclidean distance between v_p and v_q, that is $u^{ALT} = 2.21 d_{pq}$, as suggested in [13]. This utility includes both the traveling time (estimated as twice the Euclidean distance) and the price of the journey (0.21 monetary units per length unit), equally weighted.

The cost of building each link, c_{ij}, is fixed to its Euclidean length times 10^7, meaning that building the tracks costs ten million monetary units per length unit.

Then, a weight is assigned to each node, which is inversely proportional to the sum of the distances from this node to the centers: $w_i = 1/(1 + d_i)$, where d_i is the sum of the distances from node i to the centers of the corresponding instance (if $c = 0$ we set $d_i = 0$). Afterwards, the OD matrix is computed as follows. The flow between nodes v_p and v_q is set to $g_{pq} = 100 w_p w_q \frac{d_{pq}^{2.9}}{e^{1.1 d_{pq}}}$, where d_{pq} is the Euclidean distance from v_p to v_q. The calibration of the values of these parameters can be found in [13].

The maximum budget allowed is set to be 25% of the cost of building the whole network G, that is to say, $C_{\max} = 0.25(\sum_{i \in N} c_i + \sum_{\{i,j\} \in E} c_{ij})$.

For each OD pair (p, q), in these experiments we consider that the utility of the railway network depends on the riding time, the number of stops, and the ticket price, as detailed in the Appendix, Equation (38). In order to mimic the behavior of many railway networks (specially metro and commuter networks), the ticket price depends on the zones in which the OD nodes are located. In these experiments, the nodes are divided into three different zones, depending on their distances to the centers. The ticket price depends on how many zone changes you have to do in your journey on the network. Full data about the instances are available from the authors upon request. Note that the instances of this paper extend those used in [6], by including attraction centers, different zones, the possibility of directly linking neighbors of neighbors, a more realistic model for the passenger flows, among other changes. We believe that the new set of instances better reflects realistic situations.

6.2 Algorithms tested

We solved RND1 for each of the above described 60 instances, by means of the mixed integer linear program (MILP) shown in the Appendix. The solver used was CPLEX 12.6, on Virtual Windows 7 machines with 2 virtual processors and 8 GBytes of RAM memory. A virtualization server composed of 30 blades is employed. Each blade runs two Intel XEON E5420 processors running at 2.5 GHz. and 16 GBytes of RAM memory. For all instances, the maximum computational time was set to one hour. The genetic algorithms were coded and executed in for Microsoft VBA.

In order to test the validity of the GA presented in Section 5 over RND1 and RND2, we also solved the 60 instances by means of this algorithm combining different parameter values, as explained below. The size of the initial population was set to $i_{\max} = 10$. In order to calibrate the other parameters, a full factorial design of experiments was performed, modifying the following factors:

- Factor "Constructive algorithm", taking three levels: GRASP1 (allowing only connected networks), GRASP2 (allowing non-connected networks as well), random generation.

- Factor "Number of crossover points", taking three levels: $\{0, 1, 2\}$.
- Factor "Probability of mutation", taking three levels: $\{0, 0.25, 0.50\}$.

Therefore, each of the 60 instances was solved by the GA with each of the possible $3^3 = 27$ combinations of factors, having in total $27 \times 60 = 1620$ executions of the genetic algorithms, for each RND problem ($2 \times 1620 = 3240$ executions in total). Each execution run until no improvement was found in 100 consecutive iterations, or a maximum CPU time of 600 seconds was reached.

The reader may note that if the number of crossover points is zero, then the crossover operation is not performed. Similarly, the mutation operation is not performed if the probability of mutation is zero. In case both are zero, only the constructive phase is performed. In such case, after the generation of the initial population, the constructive algorithm runs until 100 solutions are generated without no improvement in the best solution, or the 600-second time limit is reached.

6.3 Results for RND1

We now summarize the results obtained in the 60 RND1 instances.

6.3.1 Effect of GA operators

The first output we want to analyze is the impact that the crossover and mutation operations have on the initial population. That is, do these two operators improve the initial population? For this aim, we have computed the percentage increase in coverage (PIC), between the value given by the initial solution (Z_i) and the value given by the best solution found at the end of the genetic algorithm (Z_g), computed as follows:

$$PIC = 100\frac{Z_g - Z_i}{Z_i}. \tag{20}$$

In the following tables, column "Crossover" denotes the number of crossover points used, and column "Mutation" denotes the probability of mutation. The next three columns give the average values of the characteristic tested, for each of the three constructive algorithms proposed for the generation of the initial population. This way, column "GRASP1" gives, for the combination of number of crossover points and probability of mutation detailed in the corresponding row, the average value over all instances (CPU time, objective function, etc). Similarly, columns GRASP2 and RANDOM give the average values obtained for these constructive algorithms.

In Table 2 we observe that, regardless of the number of crossover points and probability of mutation, the genetic algorithm operators always improve the solution obtained in the initial population, no matter the constructive algorithm used. We observe a larger improvement if the initial population is built randomly, because such initial population has networks with lower coverage. An ANOVA analysis shows that the three factors considered (number of

Table 2 Overall percentage increase in coverage (%) of the crossover and mutation operations over the initial population for the different GA tested, for RND1.

		Constructive algorithm		
Crossover	Mutation	GRASP1	GRASP2	RANDOM
0	0	10,52	10,64	43,63
1	0	9,37	10,56	102,00
2	0	11,03	7,75	81,25
0	0.25	1,61	5,16	33,29
1	0.25	11,17	10,45	101,76
2	0.25	11,57	10,06	94,93
0	0.5	5,45	7,51	72,64
1	0.5	11,48	9,71	100,22
2	0.5	11,39	8,58	**102,31**

crossover points, probability of mutation, constructive algorithm) significantly affect the improvement caused by the genetic operations, with individual p-values equal to 0, 0.03, and 0, respectively.

We also want to check how much time is needed to perform the genetic operations. Therefore, we have computed the percentage increase in time (PIT), between the time needed to compute the initial population (t_i) and the time that the crossover and mutation operations run until finding the best solution (t_g), computed as follows:

$$PIT = 100\frac{t_g}{t_i}. \tag{21}$$

In Table 3 we observe that the time that the crossover and mutation oper-

Table 3 Overall percentage increase in time (%) between the time to find the initial population and the time needed by the genetic, for the different GA tested, for RND1.

		Constructive algorithm		
Crossover	Mutation	GRASP1	GRASP2	RANDOM
0	0	279,18	452,69	754,88
1	0	438,96	384,53	1795,64
2	0	418,50	289,00	1960,71
0	0.25	89,69	101,86	280,54
1	0.25	492,51	404,19	2048,29
2	0.25	527,18	346,48	2430,32
0	0.5	235,74	179,11	764,24
1	0.5	693,00	392,70	**2624,96**
2	0.5	531,81	487,55	2437,55

ations run heavily depend on the constructive operator. For example, for 2 crossover points and probability of mutation 0.5, the genetic operations run for 5.31 times more than the GRASP1 to generate the initial population. If the GRASP2 algorithm was used for the generation of the initial population, this factor is 4.87. If the initial population is randomly generated, the GA operators run for 24.37 times more than the generation of the initial population. This factor is much larger when the initial population is built randomly, because

such generation takes less time than using one of the GRASP algorithms. An ANOVA analysis shows that the three factors considered (number of crossover points, probability of mutation, constructive algorithm) significantly affect the extra time needed by the genetic operations, with individual p-values equal to 0, 0.0003, and 0, respectively.

6.3.2 Quality of solutions

The second output we want to analyze is the quality of the solution returned, both by the MILP and by the GA. The quality of the MILP can be summarized in terms of number of optimal solutions guaranteed, and average MILP gap. These measures are summarized below.

- 47 out of the 60 instances were solved to optimality, which makes 78.3% of solutions with gap = 0.
- The average gap over all instances is 5.9%. If we restrict this analysis to the 13 instances in which the gap is strictly positive, this average gap increases to 27%.

In the rest of the section, we will divide our analysis in these two sets of instances: those which were solved to optimality in less than one hour by the MILP, and those which were not.

In order to asses the quality of the GA, we measure the *relative percent deviation* (RPD) with respect to the solution found by the MILP. For each instance and each algorithm, the RPD was measured as follows:

$$RPD = 100\frac{Z_{MILP} - Z_g}{Z_{MILP}}, \tag{22}$$

where Z_g is the coverage of the solution yielded by the corresponding genetic algorithm, and Z_{MILP} is the coverage of the solution found by the MILP. We have, for the two sets of instances, the following results:

- All instances.
  Table 4 shows the average RPD of the genetic algorithms tested over all 60 RND1 instances.
  In terms of deviation with respect to the MILP solution, the genetic algorithm with 2 crossover points, probability of mutation 0.5, and GRASP2 for generating the initial population yields the best results (average RPD equal to 2.19%). An ANOVA analysis shows that the three factors considered (number of crossover points, probability of mutation, constructive algorithm) significantly affect the RPD, with individual p-values equal to 0, 0.01, and 0, respectively.
- Instances with gap = 0.
  Table 5 shows the average RPD of the genetic algorithms tested over the 47 instances solved to optimality by the MILP. Thus, for instances in which the MILP found the optimal solution, there are some combinations of the genetic algorithm which find solutions only 1.32% far from these optimal

Table 4 Average RPD of genetic algorithms with respect to MILP solution, over all RND1 instances, in %.

		Constructive algorithm		
Crossover	Mutation	GRASP1	GRASP2	RANDOM
0	0	3,31	3,04	31,46
1	0	3,53	2,38	12,85
2	0	2,52	2,31	11,83
0	0.25	7,84	7,93	37,69
1	0.25	2,26	2,71	10,46
2	0.25	2,71	2,45	9,32
0	0.5	5,54	6,07	23,38
1	0.5	2,45	3,01	9,59
2	0.5	2,63	**2,19**	10,72

Table 5 Averge RPD of genetic algorithms with respect to MILP solution, over RND1 instances solved to optimality, in %.

		Constructive algorithm		
Crossover	Mutation	GRASP1	GRASP2	RANDOM
0	0	2,53	2,00	26,54
1	0	2,76	1,71	10,55
2	0	1,88	1,50	9,26
0	0.25	8,13	8,02	34,76
1	0.25	1,45	2,19	7,85
2	0.25	1,67	1,91	7,72
0	0.5	5,09	5,10	21,07
1	0.5	1,47	2,21	7,55
2	0.5	**1,32**	1,85	8,92

solutions, on average (the genetic algorithm with 2 crossover points, probability of mutation 0.5, using GRASP1 as constructive). An ANOVA analysis over the instances solved to optimality only, shows that the three factors considered (number of crossover points, probability of mutation, constructive algorithm) significantly affect the RPD, with individual p-values equal to 0, 0.01, and 0, respectively.

6.3.3 CPU time

The third output we want to analyze is the computational effort of the algorithms presented. In the previous section we showed that the relative percent deviations of the solutions found by the genetic algorithms are quite controlled, around 2% on average for the best combinations of GA parameters. We will now analyze the CPU time of the GA algorithms.

– All instances.
The average CPU time of the MILP is 1072.3 seconds, whereas the average CPU time of the genetic algorithms is as given in Table 6. Roughly speaking, the genetic algorithm gets solutions which are on average 2% far from the MILP (Table 4), in one third of the time. An ANOVA analysis over the instances solved to optimality only, shows that the three factors

Table 6 Average CPU time of genetic algorithms, over all RND1 instances.

		Constructive algorithm		
Crossover	Mutation	GRASP1	GRASP2	RANDOM
0	0	357,39	389,25	166,17
1	0	327,51	357,28	234,88
2	0	329,45	334,33	262,48
0	0.25	164,43	170,56	**43,93**
1	0.25	361,79	343,95	293,93
2	0.25	364,61	353,87	319,71
0	0.5	216,58	215,37	115,89
1	0.5	390,79	367,73	329,45
2	0.5	369,77	374,09	324,28

considered (number of crossover points, probability of mutation, constructive algorithm) significantly affect the CPU time, with individual p-values equal to 0, 0.01, and 0, respectively.

– Instances with gap = 0.
The average CPU time of the MILP is 373.0 seconds, whereas the average CPU time of the genetic algorithms is as given in Table 7. We observe that,

Table 7 Average CPU time of GA algorithms for RND1 instances with GAP=0. %.

		Constructive algorithm		
Crossover	Mutation	GRASP1	GRASP2	RANDOM
0	0	270,22	312,62	107,74
1	0	241,40	265,90	175,70
2	0	236,33	240,08	195,30
0	0.25	84,54	78,75	**26,67**
1	0.25	278,07	267,91	216,44
2	0.25	283,97	262,79	247,23
0	0.5	130,23	121,36	77,33
1	0.5	315,47	283,68	265,00
2	0.5	292,19	301,58	248,78

again, the fastest combination of factors is given by randomly building the initial population, not doing the crossover operation, and applying probability of mutation 0.25. An ANOVA analysis over the instances solved to optimality only, shows that the three factors considered (number of crossover points, probability of mutation, constructive algorithm) significantly affect the CPU time over the instances solved to optimality, with individual p-values equal to 0, 0.01, and 0, respectively.

6.3.4 Summary of results for RND1

Table 8 shows, for any of the variables studied, the best combination of genetic algorithm factors in the RND1 instances. As we noted before, all factors are individually significant for every variable studied.

Although we observe that randomly built initial solutions are the ones inducing the most improvement when applying the GA operators, that is just

Table 8 Best combination of Genetic Algorithm factors (crossover points, probability of mutation, constructive algorithm).

Variable	Best combination of GA factors
Improvement of coverage over initial population	(2,0.5,RANDOM)
Time increase over initial population	(1,0.5,RANDOM)
Average RPD over all instances	(2,0.5,GRASP2)
Average RPD over instances solved to optimality	(2,0.5,GRASP1)
Average CPU time over all instances	(0,0.25,RANDOM)
Average CPU time over instances solved to optimality	(0,0.25,RANDOM)

a consequence from the fact that the initial population is worse than when applying the other two constructive approaches for the initial population. We also observe how the best quality solutions are obtained when using 2 crossover points and probability of mutation 0.5. Given the lack of dominance between GRASP1 and GRASP2 as seed for the best quality solution, the logical recommendation is to always try both.

6.4 Results for RND2

We now summarize the results obtained in the 60 RND2 instances.

6.4.1 Effect of GA operators

Like in the RND1 instances, the first output we want to analyze is the impact that the crossover and mutation operations have on the initial population. For this aim, in Table 9 we show the average PIC for all combinations of parameters tested. We observe that the genetic algorithm operators always

Table 9 Overall percentage improvement in coverage (%) of the crossover and mutation operations over the initial population for the different GA tested, for RND2.

		Constructive algorithm		
Crossover	Mutation	GRASP1	GRASP2	RANDOM
0	0	13,43	12,16	44,76
1	0	11,55	12,42	80,31
2	0	11,12	11,28	84,56
0	0.25	4,55	3,09	36,34
1	0.25	10,13	11,81	100,94
2	0.25	11,68	12,57	92,43
0	0.5	8,57	5,60	60,34
1	0.5	12,46	12,44	101,06
2	0.5	11,17	11,21	**106,78**

improve the solution obtained in the initial population. Again, we observe a larger improvement if the initial population is built randomly, because such initial population has networks with lower coverage. An ANOVA analysis shows that the three factors considered (number of crossover points, probability of

mutation, constructive algorithm) significantly affect the improvement caused by the genetic operations, with individual p-values equal to 0, 0.02, and 0, respectively.

With respect to the increase in CPU time, we now observe a larger increase than in the RND1 instances, in CPU time with respect to the time devoted to build the initial population, see Table 10. An ANOVA analysis shows that the

Table 10 Overall time increase % (PIT) between the time to find the initial population and the time needed by the genetic, for the different GA tested, in RND2.

		Constructive algorithm		
Crossover	Mutation	GRASP1	GRASP2	RANDOM
0	0	1235,19	1097,28	1393,18
1	0	1325,85	831,69	1578,55
2	0	1554,27	860,94	1516,50
0	0.25	294,97	164,64	280,85
1	0.25	1740,22	1034,19	2113,50
2	0.25	1870,47	1106,99	2167,68
0	0.5	772,11	364,73	575,79
1	0.5	2176,10	1204,70	**2439,73**
2	0.5	2076,93	1213,09	2311,92

three factors considered (number of crossover points, probability of mutation, constructive algorithm) significantly affect the extra time needed by the genetic operations, with all individual p-values equal to 0.

6.4.2 Quality of solutions

The second output we want to analyze is the quality of the solution returned by the GA. Since we do not have solutions given by the MILP, we compare the results with respect to the best solution given by any of the genetics tested. Therefore, in order to asses the quality of the GA, we measure the *relative percent deviation* (RPD) with respect to the best solution found by any of the GA tested, as follows:

$$RPD = 100\frac{Z_{BEST} - Z_g}{Z_{BEST}}, \tag{23}$$

where Z_g is the coverage of the solution yielded by the corresponding genetic algorithm, and Z_{BEST} is the coverage of the best solution found by any of the 27 GA tested. Table 11 shows the average RPD of the genetic algorithms tested over all 60 RND2 instances. In terms of deviation with respect to the best solution found, the genetic algorithm with 2 crossover points, probability of mutation 0.25, and GRASP1 for generating the initial population yields the best results (average RPD equal to 1.44%). An ANOVA analysis shows that the three factors considered (number of crossover points, probability of mutation, constructive algorithm) significantly affect the RPD, with individual p-values equal to 0, 0.007, and 0, respectively.

Table 11 Average RPD of genetic algorithms with respect to the best solution found, over RND2 instances, in %.

		Constructive algorithm		
Crossover	Mutation	GRASP1	GRASP2	RANDOM
0	0	1,69	2,40	32,19
1	0	2,79	2,11	14,90
2	0	1,56	2,98	16,64
0	0.25	8,54	8,86	33,89
1	0.25	2,05	2,80	12,50
2	0.25	**1,44**	2,56	11,72
0	0.5	5,11	6,06	28,25
1	0.5	1,93	2,09	9,29
2	0.5	1,93	2,15	10,92

6.4.3 CPU time

The average CPU time of the genetic algorithms in the RND2 instances is as given in Table 12. It seems that the combination in which the GA converges

Table 12 Average CPU time of genetic algorithms, over RND2 instances.

		Constructive algorithm		
Crossover	Mutation	GRASP1	GRASP2	RANDOM
0	0	252,66	330,16	275,21
1	0	224,82	204,25	246,40
2	0	214,59	218,57	235,11
0	0.25	64,08	68,60	**56,59**
1	0.25	244,50	240,11	301,16
2	0.25	249,37	278,37	302,82
0	0.5	120,41	115,47	113,94
1	0.5	283,03	271,31	324,81
2	0.5	280,66	288,51	320,41

the fastest is that with no crossover operator, probability of mutation 0.25, and initial population built randomly. An ANOVA analysis shows that the number of crossover points is significant (p-value equal to 0), the probability of mutation is also significant (p-value equal to 0.0005), whereas the constructive algorithm is on the edge (p-value equal to 0.07), if we consider the standard significance level of 0.05. This means that the constructive algorithm chosen is not as relevant, when we speak about time until the algorithm stops, as the other two factors.

6.4.4 Summary of results for RND2

Table 13 shows, for any of the variables studied, the best combination of genetic algorithm factors in RND2 instances. As we noted before, all factors are individually significant for every variable studied.

Like in RND1, building the initial population randomly yields the best results in terms of CPU time and improvement with respect to that initial pop-

Table 13 Best combination of Genetic Algorithm factors (crossover points, probability of mutation, constructive algorithm).

Variable	Best combination of GA factors
Improvement of coverage over initial population	(2,0.5,RANDOM)
Time increase over initial population	(1,0.5,RANDOM)
Average RPD over all instances	(2,0.25,GRASP1)
Average CPU time over all instances	(0,0.25,RANDOM)

ulation. However, the best quality solutions are obtained when using GRASP1 as a constructive, 2 crossover points, and 0.25 as probability of mutation.

6.5 Comparison between RND1 and RND2

In this section we compare the coverage obtained by the genetic algorithms when the no-cycle constraint is imposed (RND2) and when the resulting network is allowed to have cycles (RND1). Note that in both problems the budget constraint is imposed. For this aim, we have compared the percentage difference between the coverage obtained for both problems as follows:

$$Z_d = 100\frac{Z_1 - Z_2}{Z_1}, \tag{24}$$

where Z_1 and Z_2 are the trip coverage obtained when solving RND1 and RND2, respectively. Table 14 summarizes these results. We observe how for

Table 14 Percentage difference between the trip coverage of RND1 and the trip coverage of RND2, for all genetic algorithms tested.

		Constructive algorithm		
Crossover	Mutation	GRASP1	GRASP2	RANDOM
0	0	-1,34	2,21	1,46
1	0	-0,17	1,66	2,14
2	0	0,90	2,78	3,85
0	0.25	-3,41	0,64	-15,88
1	0.25	0,07	2,53	2,72
2	0.25	-1,55	-0,18	-1,67
0	0.5	-2,48	1,49	-0,66
1	0.5	1,69	-1,36	-2,01
2	0.5	0,10	1,95	-3,78

15 combinations, the average coverage is better for RND1 (positive value in the table), whereas for the other 12 combinations RND2 yields larger coverage (negative value in the table). We also observe that the largest differences are observed when using RANDOM for building the initial population. The largest difference in favor or RND1 is obtained when using two crossover points and no mutation operator (3.85%). The largest difference in favor of RND2 is obtained when doing no crossover operation, and using 0.25 as probability of mutation (15.88%).

Regarding CPU times, we also compute the percentage difference between the time until the stop criteria was met by the genetic algorithms in both RND1 and RND2 (t_1 and t_2, respectively), for every instance, as follows:

$$t_d = 100\frac{t_1 - t_2}{t_1}. \tag{25}$$

Table 15 summarizes these results. We observe that for 23 of the 27 possible

Table 15 Percentage difference between the time until stop of RND1 and that of RND2, for all genetic algorithms tested.

		Constructive algorithm		
Crossover	Mutation	GRASP1	GRASP2	RANDOM
0	0	29,30	15,18	-65,62
1	0	31,35	42,83	-4,90
2	0	34,86	34,62	10,43
0	0.25	61,03	59,78	-28,82
1	0.25	32,42	30,19	-2,46
2	0.25	31,61	21,34	5,28
0	0.5	44,40	46,39	1,68
1	0.5	27,57	26,22	1,41
2	0.5	24,10	22,88	1,19

combinations of factors, RND1 is slower than RND2 (positive values), on average. We also observe that the only combinations in which the GA over RND1 stopped sooner than the GA over RND2 are always when building the initial population randomly.

Therefore, it seems that the GA obtains larger trip coverage when the no-cycle constraint is not imposed (RND1), but also in longer CPU times.

7 Conclusions

This paper presents strategic level optimization models for transportation infrastructure, and discusses the complexity of these models and several of its constraint-variations (six in total). Such variations appear when we combine budget restrictions and topological constraints. We show when the problem is trivial and when it is NP-hard. We provide examples of transformations to prove NP-hardness. In this process, we investigate and show the submodularity of the objective function of the problem.

Due to the proven complexity of these problems, we then turn our attention to metaheuristic-approaches. We test a Genetic Algorithm to address the problem that we consider the most realistic case: budget constraint and no restriction on the topology of the resulting network. Our tests suggest that Genetic Algorithms are quite efficient in computational time and also in terms of optimality gap.

A straightforward extension of both the MILP and the GA algorithm comes with the objective is not to build a network from

scratch, but rather to extend an existing network. In the corresponding MILP, the variables referring to the existing facilities (stations and links) should be fixed. Similarly, the GA algorithm would start from the existing network, and add arcs and nodes in the same way as described in this paper.

Acknowledgments

Mozart Menezes and Juan A. Mesa were partially supported by project MTM2015-67706-P (MINECO/FEDER,UE). Federico Perea was partially supported by Spanish Ministry of Science, Innovation, and Universities, under projects "OPTEP-Port Terminal Operations Optimization" (No. RTI2018-094940-B-I00) and MTM2016-74983, financed with FEDER funds, and by the *Universitat Politècnica de València* under grant SP20180164 of the program *Primeros Proyectos de Investigación (PAID-06-18), Vicerrectorado de Investigación, Innovación y Transferencia.* All this support is gratefully acknowledged.

References

1. Balakrishnan, A., Magnanti, T.L., Mirchandani, P.: Annotated Bibliography in Combinatorial Optimization, chap. Network design. John Wiley and Sons, New York (1997)
2. Bussieck, M., Winter, T., Zimmermann, U.: Discrete optimization in public rail transport. Mathematical Programming **79**(1–3), 415–444 (1997)
3. Chakroborty, P.: Genetic algorithms for optimal urban transit network design. Computer-Aided Civil and Infrastructure Engineering **18**, 184–200 (2003)
4. Chakroborty, P., Dwivedi, T.: Optimal route network design for transit systems using genetic algorithms. Engineering Optimization **34**(1), 83–100 (2002)
5. Desaulniers, G., Hickman, M.D.: Transportation Handbooks in operations research and management science, vol. 14, chap. Public Transit, pp. 69–127. Elsevier, Amsterdam (2007)
6. García-Archilla, B., Lozano, A.J., Mesa, J.A., Perea, F.: GRASP algorithms for the robust railway network design problem. Journal of Heuristics **19**(2), 399–422 (2013)
7. Garey, M., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company (1979)
8. Goel, G., Karande, C., Tripathi, P., Wanga, L.: Approximability of combinatorial problems with multi-agent submodular cost functions. ACM SIGecom Exchanges **9**(1), 1–4 (2010)
9. Grötschel, M., Lovász, L., Schrijver, A.: The ellipsoid method and its consequences in combinatorial optimization. Combinatorica **1**, 169 – 197 (1981)
10. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization, 2nd ed. Springer-Verlag, New York (1993)
11. Guihaire, V., Hao, J.K.: Transit network design and scheduling: a global review. Transportation Research Part A **42**, 1251–1273 (2008)
12. Jegelka, S.S.: Combinatorial problems with sub-modular coupling in machine learning and computer vision. Thesis ETH Zurich (2012)
13. Laporte, G., Mesa, J.A., Ortega, F.A.: Assessing the efficiency of rapid transit configurations. Top **5**, 95–104 (1995)
14. Laporte, G., Mesa, J.A., Perea, F.: A game theoretic framework for the robust railway transit network design problem. Transportation Research Part B **44**, 447–459 (2010)
15. Magnanti, T.L., Wong, R.T.: Network design and transportation planning: models and algorithms. Transportation Science **18**, 1–55 (1984)

16. Marín, A., García-Ródenas, R.: Location of infrastructure in urban railway networks. Computers and Operations Research **36**(5), 1461–1477 (2009)
17. Mesa, J.A., Boffey, B.T.: A review of extensive facility location in networks. European Journal of Operational Research **95**, 592–603 (1996)
18. Nayeem, M.A., Rahman, M.K., Rahman, M.S.: Transit network design by genetic algorithm with elitism. Transportation Research Part C: Emerging Technologies **46**, 30–45 (2014)
19. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. A Wiley-Interscience publication (1999)
20. Perea, F., Mesa, J.A., Laporte, G.: Adding a new station and a road link to a road-rail network in the presence of modal competition. Transportation Research Part B **68**, 1–16 (2014)
21. Puerto, J., Ricca, F., Scozzari, A.: Extensive facility location problems on networks: an updated review. TOP **26**(2), 187–226 (2018)
22. Schmidt, M., Schöbel, A.: Location of speed-up subnetworks. Annals of Operations Research **223**(1), 379–401 (2014)
23. Schöbel, A.: Line planning in public transportation: models and methods. OR Spectrum **34**(3), 491–510 (2012)
24. Sourirajan, K., Ozsen, L., Uzsoy, R.: A genetic algorithm for a single product network design model with lead time and safety stock considerations. European Journal of Operational Research **197**(2), 599–608 (2009)
25. Székely, L., Wang, H.: On subtrees of trees. Advances in Applied Mathematics **34**, 138–155 (2005)
26. Ukkusuri, S.V., Mathew, T.V., Waller, S.T.: Robust transportation network design under demand uncertainty. Computer-Aided Civil and Infrastructure Engineering **22**, 6–18 (2007)

Appendix

7.1 Proofs

We begin this appendix with the proof of Proposition 1.

Proof We prove this result by reduction from the knapsack problem. Consider the knapsack problem with item values equal to b_i, and item weights equal to w_i, $i = 1, ..., n$, and capacity $w_{\max}$, which is formulated as:

$$\begin{array}{l} \max \sum_{i=1}^{n} b_i x_i \\ \text{s.t.: } \sum_{i=1}^{n} w_i x_i \leq w_{\max} \\ \quad\quad x_i \in \{0,1\}, i = 1, ..., n. \end{array} \tag{26}$$

Consider the following input data: $N = \{v_0, v_1, ..., v_n\}$, $E = \{\{v_0, v_i\} : i = 1, ..., n\}$, $W = \{(v_0, v_i) : i = 1, ..., n\}$, $C_{\max} = w_{\max}$, $c_{0,i} = 0$, $c_i = w_i$, $i = 1, ..., n$, see Figure 2. Choose $g_{0,i}$, $\ell_{0,i}$, and $u_{0,i}^{ALT}$ so that $g_{0,i}\varphi(\ell_{0,i} - u_{0,i}^{ALT}) = b_i$ for all $i = 1, ..., n$. For this, choose $u_{0,i}^{ALT} = u$ and $\ell_{0,i} = \ell$ for all $i = 1, .., n$ so that $\varphi(\ell - u) > 0$. Note that these values exist because $\lim \varphi_{x \to +\infty}(x) = 1$. Let $\varphi^* = \varphi(\ell - u)$. Then define $g_{0,i} = b_i/\varphi^*$.

Solving this instance of RND1 solves the knapsack problem (26). So, if there was an algorithm that could polynomially solve this RND1 instance, you would be able to solve the knapsack problem in polynomial time, which is a contradiction because the knapsack problem is NP-hard (see [7]).

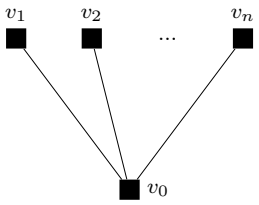


Fig. 2 A network to proof the NP-hardness of RND1.

The proof to Corollary 1 follows.

Proof The reader may note that, with the **input data** in the proof of Proposition 1, the solution to problem RND1 is the same as the solution to problem RND2, with the same data (because the underlying chosen network (N, E) is a star, and therefore any subnetwork of it is also a star).

7.2 MILP model

We now detail the MILP program we designed for solving the RND1 problem in the experiments, similar to the one introduced in [6]. The following variables are needed:

- y_i is a binary variable to decide whether or not node v_i is a station of the railway network.
- x_{ij} is a binary variable to decide whether or not edge (i, j) is a link of the railway network.
- r_{pq} is a binary variable to decide whether or not there is a path for OD-pair (p, q) in the railway network.
- w_{pq} is a binary variable to decide whether or not the railway network has a better utility than the road network for OD-pair (p, q).
- f_{ij}^{pq} is a binary variable to decide whether or not OD-pair (p, q) will use edge (i, j) from v_i to v_j in their route on the railway network.

The objective of our model is to maximize the railway trip coverage:

$$\max \sum_{(p,q)\in W} g_{pq} w_{pq}. \tag{27}$$

The constraints of our model have been grouped according to their aims

- Budget constraints,

$$\sum_{(i,j)\in E} c_{ij}\, x_{ij} + \sum_{n_i \in N} c_i y_i \leq C_{\max}. \tag{28}$$

– Edges can be used in both senses, and if a link is built, then its endnodes must be stations of the railway network,

$$x_{ij} = x_{ji},\ (i,j) \in E, \tag{29}$$
$$x_{ij} \leq y_i,\ (i,j) \in E, \tag{30}$$
$$x_{ij} \leq y_j,\ (i,j) \in E. \tag{31}$$

– Routing demand conservation constraints,

$$\sum_{i:(i,p)\in A} f_{ip}^{pq} = 0,\ (p,q) \in W, \tag{32}$$
$$\sum_{j:(p,j)\in A} f_{pj}^{pq} = r_{pq},\ (p,q) \in W, \tag{33}$$
$$\sum_{i:(i,q)\in A} f_{iq}^{pq} = r_{pq},\ (p,q) \in W, \tag{34}$$
$$\sum_{j:(q,j)\in A} f_{qj}^{pq} = 0,\ (p,q) \in W, \tag{35}$$
$$\sum_{i:(i,k)\in A} f_{ik}^{pq} - \sum_{j:(k,j)\in A} f_{kj}^{pq} = 0,\ \forall\ k \notin \{p,q\},\ (p,q) \in W. \tag{36}$$

– Location-Allocation constraints

$$f_{ij}^{pq} + r_{pq} - 1 \leq\ x_{ij},\ (i,j) \in A,\ (p,q) \in W. \tag{37}$$

– **Disutility of the railway network**

$$u_{pq} = \sum_{(i,j)\in A} d_{ij} f_{ij}^{pq} + M(1 - r_{pq}) + t_s((\sum_{(i,j)\in A} f_{ij}^{pq}) - 1) + \gamma(|zone_p - zone_q| + 1) \tag{38}$$

– Splitting demand constraints

$$(u_{pq} - u_{pq}^{ALT}) - M(1 - w_{pq}) \leq 0. \tag{39}$$

where M is a real number sufficiently large.

Constraint (28) states that construction costs cannot exceed the budget, $C_{\max}$. Constraints (29) allow the constructed links to be used in both directions. Constraint (30) and (31) impose that, if a link is built, then its corresponding endnodes should have a station. Constraints (32) to (36) are flow conservation constraints for variables f. Note that if $r_{pq} = 0$, there will be no flow from p to q via the railway network. Constraints (37) force that demands are only allocated through public arcs if the corresponding edges are built. Constraints (38) define the utility of each OD-pair in the railway network, which depends on the riding time, the number of stops, and the journey price. In these experiments, we took $t_s = 0.5$ and $\gamma = 1$. This implies that the stop time at stations is 0.5, and

that the ticket price is 1, 2, or 3, if the number of zone changes is 0, 1, or 2, respectively. Note that the utility of (p, q) is a large enough constant M if there is no path from p to q in the railway network. Constraints (39) impose for each OD-pair (p, q) that, if their utility using the road network is better than their utility using the railway network, then $w_{pq} = 0$, and therefore this OD-pair is not covered.

Besides these constraints, we also added to our model the following cuts:

$$f_{ij}^{pq} \leq x_{ij}, \forall\ (p,q) \in W,\ (i,j) \in A.$$
$$w_{pq} \leq r_{pq}.$$

The first one imposes that no edge can be used if its corresponding link is not built. The second one imposes that an OD-pair without a path in the railway network cannot be covered. Previous experience showed that these two cuts significantly reduced the computational times.