A Biased random key genetic algorithm for the field technician scheduling problem

Ricardo B. Damm, Mauricio G.C. Resende, Débora P. Ronconi

www.elsevier.com/locate/caor

PII: S0305-0548(16)30110-1
DOI: http://dx.doi.org/10.1016/j.cor.2016.05.003
Reference: CAOR3997

To appear in: *Computers and Operation Research*

Received date: 22 December 2014
Revised date: 27 April 2016
Accepted date: 3 May 2016

Cite this article as: Ricardo B. Damm, Mauricio G.C. Resende and Débora P Ronconi, A Biased random key genetic algorithm for the field technician scheduling problem, *Computers and Operation Research* http://dx.doi.org/10.1016/j.cor.2016.05.003

A Biased Random Key Genetic Algorithm for the Field Technician Scheduling Problem

Ricardo B. Damm[a], Mauricio G.C. Resende[b,*], Débora P. Ronconi[a,**]

[a] *University of São Paulo, Polytechnic School, Department of Production Engineering, São Paulo, Brazil, Av. Prof. Almeida Prado, 128, Cidade Universitária, 05508-070, São Paulo SP, Brazil*
[b] *Mathematical Optimization and Planning, Amazon.com, 333 Boren Avenue North, Seattle, WA 98109, USA*

Abstract

This paper addresses a problem that service companies often face: the field technician scheduling problem. The problem considers the assignment of a set of jobs or service tasks to a group of technicians. The tasks are in different locations within a city, with different time windows, priorities, and processing times. Technicians have different skills and working hours. The main objective is to maximize the sum of priority values associated with the tasks performed each day. Due to the complexity of this problem, constructive heuristics that explore specific characteristics of the problem are developed. A customized Biased Random Key Genetic Algorithm (BRKGA) is also proposed. Computational tests with 1040 instances are presented. The constructive heuristics outperformed a heuristic of the literature in 90% of the instances. In a comparative study with optimal solutions obtained for small-sized problems, the BRKGA reached 99% of the optimal values; for medium- and large-sized problems, the BRKGA provided solutions that are on average 3.6% below the upper bounds.

Keywords: Routing and scheduling technicians, time windows, heuristic, Biased Random Key Genetic Algorithm

1. Introduction

This paper analyses the field technician scheduling problem (FTSP), which service companies often face [1, 2, 3], especially in telecommunications [4, 5, 6, 7]. These services are generally

*Work of this author was done when he was employed by AT&T Labs Research.
**Corresponding author: tel.: +55 11 30915363; fax: +55 11 30915399.
Email address: dronconi@usp.br (Débora P. Ronconi)

maintenance or installation services that need to take into account several parameters: technicians with different skills and working hours, time windows of tasks, cost of displacement and travel time, priority or urgency of service, due dates of the orders, etc. There are several possible objectives for the FTSP, such as maximizing the number of performed tasks, minimizing the completion time of all tasks, minimizing costs or total displacement, minimizing the number of technicians, etc. In this paper we will primarily address the maximization of the total priority of the performed tasks.

According to Kovacs et al. [4] and Pillac et al. [8], the FTSP is an extension of the vehicle routing problem with time windows (VRPTW), which is NP-hard. As far as we know, the research on FTSP is not extensive and most papers focus on heuristic methods. Tsang and Voudouris [7] and Xu and Chiu [9] were among the first authors to study the problem. After 2005, the problem received increased attention in the literature, particularly in 2007, when the French Operational Research Society proposed the technician and task scheduling problem as the subject of the 2007 challenge in collaboration with France Telecom. This company needed to deal with a large increase in the number of services while having a limited number of technicians. Services were divided into three groups of priorities and the objective function minimized a weighted sum of the completion time of the last task of each group. Technicians had different levels of skills and teams could be organized to perform the services. Furthermore, some tasks could be outsourced and some tasks had precedence constraints; on the other hand, the time windows to perform the tasks were not imposed. Hashimoto et al. [5] applied the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic and Cordeau et al. [6] developed constructive heuristics and customized the Adaptive Large Neighborhood Search (ALNS) heuristic to solve this problem. Kovacs et al. [4] studied a similar problem considering tasks with the same priority but with different time windows. The objective function intended to minimize the costs of displacement and outsourcing. The authors also applied the ALNS heuristic.

Other research studies also considered the formation of teams of technicians. In Dohn et al. [10] the number of assigned tasks in a day is maximized subject to restrictions related to the teams and tasks time windows and with a limited number of teams. The authors introduced a branch-and-price approach to address the problem and found 11 optimal solutions of the 12

realistic instances. Li et al. [3] intended to minimize the total number of workers and total displacement in the port of Singapore by allocating different types of workers in teams. In this problem each job must be carried out in a preestablished time window. The authors developed two constructive heuristics associated with simulated annealing to solve the problem. In Overholts II et al. [11] the daily maintenance of military equipment in USA is scheduled to maximize the weighted sum of the maintenance tasks performed. A mixed integer linear programming model was developed to solve the problem. The authors presented a detailed sensitivity analysis on security criteria and quality of daily maintenance services.

The FTSP was also addressed considering real problems with the assumption that each task can be performed by a single technician. In Tsang and Voudouris [7], the objective was the minimization of the total cost: the cost of displacement of engineers, the cost of overtime, and the cost (or penalty) of tasks not executed. This problem was a real problem faced by British Telecom. Instead of specifying time windows in hours, three types of time windows were adopted for tasks: morning, afternoon, and indifferent. The authors developed two heuristics for the problem: Fast Local Search and Guided Local Search. Tang et al. [2] developed a tabu search metaheuristic for a real problem of United Technologies Corporation. It was a problem of periodic maintenance of pieces of equipment located in buildings that were geographically dispersed. The daily work of each technician has to be scheduled for a period of one or two weeks, aiming to maximize the total reward received from servicing the selected tasks over the scheduling period (in this context, rewards are a value assigned to each task to represent its urgency and not the monetary profit). Pillac et al. [8] analyzed the similarity between the technician routing and scheduling problem and the vehicle routing problem with time windows. The objective of both problems was to minimize the total travel time. The authors developed an ALNS parallel algorithm and validated the algorithm with instances of Solomon (1987) for the VRPTW. Recently, Cortés et al. [1] presented a real problem of maintenance of printing machines and digital copiers of a large company in Santiago, Chile. Twenty technicians usually visit seventy customers every day, and according to customer relevance, the company establishes a soft time window for each service, so that the most important customers are served first. The objective function considers the number of times that the time windows are violated, the number

of customers served, and the total travel time. A branch-and-price approach associated with constraint programming was developed.

Observe that while Tang et al. [2], Overholts II et al. [11], Cordeau et al. [6], and Hashimoto et al. [5] directly consider the priorities of the tasks, Tsang and Voudouris [7], Li et al. [3], Dohn et al. [10], Kovacs et al. [4], Pillac et al. [8], and Cortés et al. [1] tackle the allowed time windows to perform the tasks. As far as we know, the only research that simultaneously approaches these relevant characteristics of the maintenance services, as well as working hours of technicians, is the one presented by Xu and Chiu [9]. These authors addressed the FTSP aiming to assign a set of jobs, at different locations with time windows, to a group of field technicians with different job skills. The objective was to maximize the sum of priority values associated with the tasks performed in a day and, secondly, the idle time of the employees after returning to the base. A constructive heuristic, a local search algorithm, and a GRASP metaheuristic were proposed to solve this problem.

Motivated by the practical relevance of the FTSP, e.g. [1, 2, 3, 5, 6, 7, 11], and the reduced number of studies that consider time windows and priorities of the tasks concurrently, in the present paper we consider the same problem addressed by Xu and Chiu [9] through the development of methods that explore these specific characteristics. Note that in the problem addressed here, differently from the problem addressed in Cortés et al. [1], these tasks parameters are independent (in fact we consider that the time window of the task is defined by the client) and both are considered in our resolution methods.

Initially, constructive heuristics are proposed; the small computational effort of such strategy is one of the reasons that motivated this study. Then we present a customized Biased Random Key Genetic Algorithm (BRKGA) metaheuristic, a Genetic Algorithm (GA) that uses random keys and does not generate unfeasible solutions. The choice of BRKGA is based on its success in several combinatorial optimization problems: single machine [12], covering problem [13], divisible load scheduling [14], lot sizing [15], telecommunications [16], winner determination in auctions [17], bin packing problems [18], berth allocation problem [19], layout [20], and transportation planning [21]. In special, BRKGA was successfully applied to related routing problems such as the problem of routing and wavelength assignment in optical networks [22], the family travelling

salesman problem [23], and a problem of collection of blood samples at clinical laboratories [24], among others. On the other hand, as far as we know, the application of a population-based metaheuristic such as BRKGA to the field technician scheduling problem (FTSP) has never been reported in the literature. This scenario motivates the application of BRKGA to the FTSP. A computational study with 1040 instances was carried out to analyze the performance of the suggested algorithms in comparison with methods of the literature and proposed upper bounds; a comparison with optimal solutions values was also conducted for small problems.

This paper is organized as follows. Section 2 presents the mathematical model of the problem. Section 3 describes the proposed constructive heuristics, while Section 4 explains how the Biased Random Key Genetic Algorithm was customized for the focused problem. Section 5 presents two different schemes for the generation of upper bounds for the FSTP. Section 6 describes the computational experiments and the last section summarizes the main results.

2. Mixed Integer Linear Programming Model

Based on the model proposed by Xu and Chiu [9], a Mixed Integer Linear Programming (MILP) model for the problem is presented below. Let $J = \{1, ..., n\}$ be a set of independent tasks or services and $K = \{1, ..., m\}$ a set of technicians available to execute them. Let p_i be the processing time of task i that must be performed within a specific time window $[e_i, l_i]$. Technicians should have their daily work schedule between $[a_k, b_k]$. The travel time from location i to j is c_{ij}, with $i, j \in J \cup \{0\}$, where 0 is a dummy task which represents the origin. Each task i has a priority w_i. The skill of a technician k to perform a task i is given by s_{ik}, a binary parameter, where 1 indicates "being capable" and 0, "unable to execute it". Next, the variables of the MILP model are described: y_{ik} is a binary variable that equals 1 if task i is assigned to technician k and 0, otherwise; x_{ijk} is also a binary variable which assumes value 1 when task i precedes task j in the route of technician k and 0, otherwise; t_i is the start time of task i execution, and z_k is the idle time after technician k returns to the origin. The objective is to maximize the sum of priority values associated with the tasks performed each day and, secondly, the idle time of the employees after returning to the origin (at the end of the day).

MILP model.

$$\max \sum_{i \in J} \sum_{\substack{k \in K \\ |s_{ik}=1}} \frac{w_i y_{ik}}{MW} + \sum_{k \in K} \frac{z_k}{MZ} \tag{1}$$

Subject to:

$$\sum_{\substack{k \in K \\ |s_{ik}=1}} y_{ik} \leq 1 \qquad i \in J \tag{2}$$

$$\sum_{\substack{j \in J \cup \{0\} \setminus i \\ |s_{jk}=1}} x_{ijk} = \sum_{\substack{j \in J \cup \{0\} \setminus i \\ |s_{jk}=1}} x_{jik} = y_{ik} \qquad k \in K, i \in J, s_{ik} = 1 \tag{3}$$

$$\sum_{\substack{i \in J \\ |s_{ik}=1}} x_{0ik} \leq 1 \qquad k \in K \tag{4}$$

$$e_i \leq t_i \leq l_i - p_i \sum_{\substack{k \in K \\ |s_{ik}=1}} y_{ik} \qquad i \in J \tag{5}$$

$$t_i + p_i + c_{ij} \leq t_j + M(1 - x_{ijk}) \qquad k \in K, i \neq j \in J, s_{ik} = s_{jk} = 1 \tag{6}$$

$$a_k + c_{0j} \leq t_j + M(1 - x_{0jk}) \qquad k \in K, j \in J, s_{jk} = 1 \tag{7}$$

$$t_i + p_i + c_{i0} \leq b_k - z_k + M(1 - x_{i0k}) \qquad k \in K, i \in J, s_{ik} = 1 \tag{8}$$

$$z_k \leq b_k - a_k \qquad k \in K \tag{9}$$

$$y_{ik} \in \{0, 1\} \qquad i \in J, k \in K, s_{ik} = 1 \tag{10}$$

$$x_{ijk} \in \{0, 1\} \qquad i, j \in J \cup \{0\}, k \in K, s_{ik} = s_{jk} = 1 \tag{11}$$

$$t_i, z_k \in R_+ \qquad i \in J, k \in K \tag{12}$$

Where:

$$MW = \min_{i \in J} w_i$$

$$MZ = \sum_{k \in K} (b_k - a_k)$$

$$M = \max \left(\max_{i \in J} l_i, \max_{k \in K} a_k \right) + \max_{i,j \in J \cup \{0\}} c_{ij}$$

In the objective function (1), the first term maximizes the sum of priority values associated with the tasks performed each day, while the second term maximizes the sum of the idle time after technicians return to the base. In other words, the first term aims to select technicians for the priority tasks and the secondary objective minimizes their total travel time and waiting

time. The inclusion of this secondary objective is a strategy that allows performing new external tasks at the end of the day or makes it possible for technicians to perform administrative tasks (such as reporting, performing equipment maintenance, etc.). The purpose of the denominator is two-fold: obtain dimensionless values of the sum and ensure that it is always better to perform a task and reduce technicians' idle time, but never vice versa. Note that the term associated with the execution of a task always has a value greater than one, while the term associated with the idleness of a technician always has a value smaller than one.

Constraints (2) ensure that each task is assigned at most to one technician. In equations (3), when a task i is assigned to a technician k, there is only one predecessor and one successor of i. Each technician must leave the origin at most once, as indicated by (4); these constraints directly impact constraints (7) and (8), which ensure that solutions are within the time window of technicians. Each task must be performed within its time window, as shown by constraints (5). When two tasks are assigned to the same technician and are consecutively executed, constraints (6) guarantee that the beginning of service j occurs after the completion time of its predecessor i plus the travel time between their locations. Constraints (6) - together with (7) and (8) - prevent subtours. Constraints (9) ensure the unoccupied time of technicians after returning to the origin as less than or equal to the size of their time windows. Constraints (10) and (11) define variables x and y as binaries, while constraints (12) define t and z as non-negative real variables. Note that the variables y_{ik} are generated only when $s_{ik} = 1$ and variables x_{ijk} when $s_{ik} = s_{jk} = 1$, i.e. when the technicians can execute the tasks.

3. Constructive Heuristics

In this section, we propose three constructive heuristics for the FTSP, namely Shortest travel time, Nearest technician (that is a greedy insertion heuristic that is similar to the one of Xu and Chiu [9]), and Cluster. Roughly speaking, these constructive heuristics order the tasks according to a decision criterion and, after that, assign these tasks to technicians. The details of the heuristics are presented below.

3.1. Shortest travel time (STT)

This method consists of four steps:

1. *Task Ordering:* Sort the tasks in decreasing order of ρ, given by:

$$\rho_i = \begin{cases} w_i - \frac{NST_i}{m+1}, & \text{if } \frac{n}{m} < 10 \\ w_i + \frac{w_i}{p_i} + \frac{p_i}{l_i - e_i}, & \text{otherwise} \end{cases} \tag{13}$$

where NST_i is the number of skilled technicians for task i.

Note that if the instance has few tasks per technician (less than 10), the jobs will be sorted in decreasing order of priority (w_i) and the second part of the expression will only serves as a tiebreak when two or more tasks have the same priority. On the other hand, when tasks per technicians are greater than or equal to 10, it is more likely that some tasks will not be performed due to the limited working time of the technicians. Considering this characteristic of these instances, the calculation of the ρ also takes account the processing time of the tasks. The jobs will be sorted in decreasing order of priority plus priority per processing time ($w_i + \frac{w_i}{p_i}$); if two or more tasks have the same value for this expression, the value of $\frac{p_i}{l_i - e_i}$ is used as a tiebreak, i.e., tasks with the highest processing time and shortest time window are programmed before. If two or more tasks have the same ρ_i, sort the tasks in increasing order of the identification number of the task. Select the task i with the best rating.

2. *Evaluation of candidate technicians:* for each technician k able to execute task i (i.e. $s_{ik} = 1$), evaluate all the possible positions of this task in his/her route. Choose the position that provides a feasible solution with the greatest z_k.

3. *Selection of one technician:* if there is no feasible solution in Step 2, go to 4. Otherwise, select the technician who can serve task i with the lowest total travel time (from the beginning until return to the base). If more than one technician has the lowest total travel time, the second and the third decision criteria are the greatest value for z_k and the identification number of the technician, respectively.

4. *Stop condition:* if task i is the last task sorted in Step 1, stop. Otherwise, select the next task i classified in the first step and go back to Step 2.

3.2. Nearest technician (NT)

This method also consists of four steps:

1. *Task Ordering:* Sort the tasks in decreasing order of ρ, given by (13). Select the task i with the best rating.
2. *Selection of one technician:* select the skilled technician k (not tried yet) with the last scheduled task closest to the task i. When more than one skilled technician has the minimum distance to task i (which occurs, for example, when all technicians are at the base), select technician k who has the smallest sum of weighted skills ($\sum_{j \in D} w_j \, s_{jk}$, where D is the set of tasks that have not been tried to be inserted yet). The last decision criterion is the technician identification number. If no technician can perform this task, go to Step 4.
3. *Scheduling the task:* evaluate all possible positions of the task i in the route of technician k. If there are feasible solutions, choose the position that provides a solution with the greatest z_k and go to the next step. Otherwise, return to Step 2.
4. *Stop condition:* if task i is the last task sorted in Step 1, stop. Otherwise, select the next task i classified in the first step and go back to Step 2.

3.3. Cluster

The main idea of this heuristic is to balance the strategy of exploiting clusters and the methods that consider tasks in all regions simultaneously. Initially, this heuristic identifies areas with a high concentration of tasks (*clusters*) and assigns values (P) to them. The clusters are ranked in descending order of their P values and a certain number of technicians (n_{tec}) are assigned to them. These technicians are chosen according to their ability of performing tasks with a good benefit for time unit, i.e., the scheduling of tasks with the highest ratio of priority per time unit $\frac{w_i}{p_i}$ is privileged. After that, the other tasks are scheduled through the NT heuristic previously presented. The choice of the NT heuristic was based on preliminary tests where it was observed that its combination with the cluster strategy provides slightly better results than the association of this strategy with the STT heuristic. The main steps of this method are described next.

1. *Defining the number of clusters and identifying their centers:* the number of clusters depends on the number of tasks. If the total number of tasks (n) is greater than 100, then there will be 10 clusters; otherwise, the number of clusters will be $\lceil \frac{n}{10} \rceil$. Tasks are sorted in descending order of $\mu_i = \frac{w_i}{p_i}$ and the first task is the center of the first cluster. The center of the second cluster is the next task with more than $t_{cluster}$ minutes of displacement from the first center. The center of the third cluster is the next task with more than $t_{cluster}$ minutes of displacement from the first and the second clusters, and so on. Nonetheless, if all tasks are analyzed and it is not possible to select a new center, then there will be less than 10 or $\lceil \frac{n}{10} \rceil$ clusters.
2. *Building clusters and calculating their values:* for each task, identify the closest center; if the displacement from this center to the task is less than $t_{cluster}$ minutes, then assign it to this cluster; otherwise, this task is not included in any cluster. Next, calculate the value P of each cluster, given by $\sum_{i \in A} \mu_i$. Set A is composed by the tasks in the evaluated cluster that present a ratio of priority per time unit greater than a pre-established ideal value, i.e. $\mu_i \geq \mu_{ideal}$.
3. *Assigning n_{tec} technicians to clusters:*
   (a) Select cluster C with the highest value P. If all clusters have values equal to zero, go to Step 4.
   (b) For cluster C, calculate the fitness of each technician k not associated with any cluster, given by $\sum_{j \in B} w_j s_{jk}$, where B is the set of all unscheduled tasks in cluster C; however, if a technician is unable to execute all tasks with $\mu_i \geq \mu_{ideal}$ in B, fitness is zero.
   (c) If fitness of all technicians is zero, change the value of cluster C to zero and return to Step 3a. Otherwise, select technician k with the highest fitness.
   (d) Select tasks in Cluster C with $\mu_i \geq \mu_{ideal}$, sort them in decreasing order of μ_i and select task i with the best ranking.
   (e) Evaluate all possible positions of task i in the route of technician k. If there is no feasible solution, go to the next step. Otherwise, choose the position that results in a feasible solution with the greatest z_k.

(f) If task i is the last task sorted in Step 3d, go to the next step. Otherwise, select the next task i and go back to Step 3e.

(g) If n_{tec} technicians are already assigned to clusters, go to Step 4. Otherwise, update the value P of cluster C (given by $\sum_{i \in A} \mu_i$, where A is the set of unscheduled tasks in C with $\mu_i \geq \mu_{ideal}$) and return to Step 3a.

4. *Scheduling the remaining tasks and technicians:* schedule the remaining tasks using the NT constructive heuristic.

Other methods were developed to order tasks according to other rules (processing time, beginning or size of time window, number of technicians able to perform a task, etc.) and to minimize the idle time between tasks, considering the number of skills of each technician, etc. The performance of these strategies was worse than the three heuristics described earlier and their details will be omitted.

4. Biased Random Key Genetic Algorithm (BRKGA)

Introduced by Holland in 1975, the genetic algorithms (GAs) are inspired by Darwin's evolution theory and work with populations of solutions that evolve over successive generations. Each solution of the optimization problem being solved is represented by an individual or a chromosome of the population. The quality of each solution is measured by a fitness function (e.g., objective function value) and the search proceeds over a number of generations, where each individual contribution to the next generation is based on its fitness [25].

Aiming to improve the performance of the traditional GA, Bean [26] proposed a specific GA for combinatorial problems which uses vectors of random keys (real numbers between 0 and 1) to represent the chromosomes. This method examines the solution space of the problem indirectly by searching within the space of random keys (problem-independent) and uses a decoder (problem-dependent) to map solutions of random keys to the solution of the optimization problem [23, 27]. It should be stressed that an essential element of the RKGA is the decoder that should be specified in such way to avoid the generation of unfeasible solutions of the optimization problem. This version of the GA was called Random Key Genetic Algorithm (RKGA) and has been applied to many research studies, providing good results. For instance, Gonçalves et

al. [28] applied the RKGA to the job shop problem and achieved the best known results in the literature in 72% of the tested problems, outperforming seven different versions of GA and other metaheuristics, such as Simulated Annealing and GRASP.

More recently, a new concept was introduced into this GA to increase the prevalence of the best solutions (the best fitness values or elite set) of each generation [27]: in the crossover, one parent is always chosen from the best solutions (in Bean [26] both parents are chosen randomly from the entire previous generation). This new version was called by Biased Random Key Genetic Algorithm (BRKGA), where *Biased* refers to the prevalence of the elite solutions. Comparisons between standard GA, RKGA, and BRKGA for different optimization problems can be found in Gonçalves and Resende [27] and Gonçalves et al. [29]; the authors concluded that the BRKGA is more effective than the RKGA. Noronha et al. [22] studied the problem of Routing and Wavelength Assignment and the results obtained by BRKGA outperformed the best heuristics from the literature (the best fit decreasing heuristic and the partition coloring problem heuristic), both in quality and computational cost. Additionally, in Resende et al. [13], on the problem of Steiner triple covering, the BRKGA reached optimal solutions for all instances in which these solutions are known, and for two large instances, the BRKGA found better results than the literature. Due to the good results obtained by BRKGA in previous works, the current research aims to develop a customized BRKGA for the resolution of the FTSP. It should be pointed out that, to the extent of our knowledge, no application of this approach to the FTSP has been examined in the literature.

According to Morán-Mirabal et al. [23], the BRKGA metaheuristic can be described as follows. The initial population P_0 is composed of n_p vectors (or chromosomes) and each vector, $\lambda_j (j = 1, \ldots, n_p)$, has n_c random numbers (or genes) in the interval $(0, 1]$, represented by $\lambda_j[i]$ $(i = 1, \ldots, n_c)$. A decoder transforms each chromosome λ_j into a feasible solution of the optimization problem and a fitness value $f(\lambda_j)$ is computed. Each generation P_g is divided into two groups; the n_e best (elite) solutions make up the first group (P_g^e) and the remaining (non-elite) solutions are included in the second set $P_g^{\bar{e}}$, where $|P_g^e| < |P_g^{\bar{e}}|$. To develop the next generation $(g+1)$, all elite solutions are copied into population P_{g+1}, a set of n_m new random solutions are generated (called mutant individuals) and the rest of the individuals ($n_0 = n_p - n_e - n_m$)

are created by a parameterized uniform crossover between pairs of individuals from P_g. In this parameterized uniform crossover, two chromosomes are selected at random: one from P_g^e and another from $P_g^{\bar{e}}$. One randomly generated real number between 0 and 1 selects each gene of the offspring: the probability of the elite parent to transmit its gene is equal to or larger than

Algorithm 1 Framework of the BRKGA (based on Morán-Mirabal et al., 2014)

Input: n_p, n_c, n_e, n_m, p_e

Create P_0 with n_p individuals: include solutions provided by constructive heuristics (if there are some) and generate the remaining chromosomes with n_c random-keys $\in (0; 1]$
$g = 0$
while stopping criterion is not satisfied **do**
    Evaluate fitness of each new individual in P_g
    Partition P_g into P_g^e and $P_g^{\bar{e}}$
    Initialize next population: $P_{g+1} \leftarrow P_g^e$
    Generate n_m mutants, P_m, each one having n_c random-keys $\in (0; 1]$
    $P_{g+1} \leftarrow P_{g+1} \cup P_m$
    for $k \leftarrow 1$ to $n_p - n_e - n_m$ **do**
        Select parent a at random from P_g^e
        Select parent b at random from $P_g^{\bar{e}}$
        for $i \leftarrow 1$ to n_c **do**
            Generate a real number p_{random} between 0 and 1
            if $p_{random} \leq p_e$ **then** $c[i] \leftarrow a[i]$
            else $c[i] \leftarrow b[i]$
            end if
        end for
        $P_{g+1} \leftarrow P_{g+1} \cup \{c\}$
    end for
    $g \leftarrow g + 1$
end while
Return $\lambda^* \leftarrow$ **argmax**$\{f(\lambda_g) | \lambda_g \in P_g\}$

the non-elite parent, ensuring that the elite parent has a greater chance to pass along its keys. There are no mutations after crossover; the diversification is guaranteed by the mutants. Once population P_{g+1} is formed, all operations are repeated until a stopping criterion is reached. The best chromosome of the last population is returned as the solution of the optimization problem. Algorithm 1 presents the BRKGA's framework for maximization problems.

Elite population diversification. During the initial computational experiments a premature convergence was observed. Aiming to avoid this undesirable behavior, a similarity measure was proposed to diversify the elite group. Applying this measure, only the best solutions with a predetermined percentage of different genes can be included in the elite set. A similar idea was analyzed by Armentano and Ronconi [30]. For example, Table 1 shows the best 6 chromosomes of one instance in the tenth generation sorted in decreasing order of the objective function. Each row represents one chromosome, while each column represents the random key associated with each task. Assume that the elite group size is 3. Note that there are columns with the same random key for almost all chromosomes. Assuming 60% as the maximum degree of similarity (DS) allowed, only chromosomes with up to 4 identical keys are considered for inclusion in the elite set. Considering that the elite group is empty, chromosome 1 (the best solution) is included in this group. The second and the third chromosomes have 5 and 6 identical keys to the first chromosome and therefore, they are not included. Chromosomes 1 and 4 have the same keys for only 3 tasks, so chromosome 4 is the second element of the elite group. Chromosomes 5

Table 1: Example of the best six individuals of a population in tenth generation

Chromosomes	Random Keys						
1	0.63	0.55	0.88	0.65	0.45	0.97	0.32
2	0.63	0.55	0.88	1.00	0.58	0.97	0.32
3	0.63	0.55	0.88	0.65	0.45	0.34	0.32
4	0.63	0.55	0.17	0.65	0.68	0.37	0.02
5	0.63	0.55	0.88	0.65	0.68	0.37	0.02
6	0.78	0.29	0.88	0.19	0.68	0.97	0.23

and 4 have the same keys in 6 positions, so chromosome 5 is rejected. The sixth chromosome is included in the elite group, because it has only 2 and 1 keys identical to chromosomes 1 and 4, respectively. Observe that, if the three best chromosomes were selected, there would be the same key for 4/7 of the tasks in the elite group. With the application of the similarity measure, chromosomes 1, 4 and 6 were selected and all tasks have two or more different values of keys in the elite group, thus increasing the diversification in the BRKGA.

Algorithm 2 shows the pseudo code of the construction of the elite set using the similarity measure, where *limit_time* is a prefixed time (stop criterion) sufficient for BRKGA to reach convergence; the other parameters and variables are self-explanatory. The elite set is initially formed by the best solution (line 2) and the other elements are selected inside the main loop from lines 4 to 29. Inside this loop, it is necessary to decide if a candidate chromosome can be included or not in the elite group (lines 7 and 21). With this purpose, the number of equal random keys between two chromosomes is calculated (lines 13 to 16) and the algorithm checks if the percentage of equal random keys is superior to DS (lines 17-18). Attempting to reduce the computational cost of this algorithm, two conditions are checked before considering a candidate chromosome: first, if the objective function of the candidate is equal to the previous chromosome, then the candidate is not included in the elite set (line 7-8); secondly, if two chromosomes were in the elite group of the previous generation, then their similarity is less than DS (line 11) and it is not necessary to calculate the number of equal random keys. Finally, it should be emphasized that, if the percentage of chromosomes with similarity less than DS is small, then the elite set cardinality can be less than n_e. However, as the percentage of equal random keys of the P^e_{g-1} is inferior or equal to DS_{g-1}, $P^e_{g-1} \subset P_g$, and $DS_{g-1} \leq DS_g$, then there will certainly be n_e elements in P^e_g.

Algorithm 2 Construction of the elite set using the similarity measure

Input: current population (P_g), elite set of the previous generation (P^e_{g-1}), objective function (f), n_p, n_e, n_c, $limit\_time$, DS^0

Output: elite set of the current population (P^e_g)

Rank P_g in descending order of the value of the objective function
Include the best solution λ_1 in P^e_g ($P^e_g \leftarrow \{\lambda_1\}$)
$j \leftarrow 2$ and $DS \leftarrow DS^0 + (1 - DS^0) \cdot \frac{current\_time}{limit\_time}$
for $i \leftarrow 2$ to n_e **do**
  while $\lambda_j \notin P^e_g$ and $j < n_p$ **do**
    $decision \leftarrow 0$
    if $f(\lambda_j) = f(\lambda_{j-1})$ **then**
      $decision \leftarrow -1$
    else
      for $k \leftarrow 1$ to $j - 1$ **do**
        if $\lambda_k \in P^e_g$ and λ_j or $\lambda_k \notin P^e_{g-1}$ **then**
          $counter \leftarrow 0$
          for $q \leftarrow 1$ to n_c **do**
            if $\lambda_k[q] = \lambda_j[q]$ **then** $counter \leftarrow counter + 1$
            end if
          end for
          if $\frac{counter}{n_c} > DS$ **then** $decision \leftarrow -1$ and $k \leftarrow j$
          end if
        end if
      end for
    end if
    if $decision = -1$ **then** $j \leftarrow j + 1$
    else $P^e_g \leftarrow P^e_g \cup \{\lambda_j\}$
    end if
  end while
  if $j = n_p$ **then** $i \leftarrow n_e$
  else $j \leftarrow j + 1$
  end if
end for

It should be highlighted that a crucial component of the BRKGA is the decoder that transforms each chromosome λ_j into a feasible solution of the problem at hand. Furthermore, it is important to determine an encoder that can convert a feasible solution (e.g. the solutions provided by constructive heuristics) into an encoded chromosome. There are many different ways to determine the decoder and the encoder. Next, we describe the three algorithms proposed in this paper.

4.1. Proposed BRKGA for the FTSP

This section presents three versions of BRKGA and describes their main components. The first version is based on the STT constructive heuristic and the second one is based on the Cluster constructive heuristic. In the third version, random keys are used to assign tasks to technicians and to construct the routes.

4.1.1. BRKGA-STT:

This BRKGA version represents a solution with a vector composed of n elements that associates one random key with each task. The initial population comprises random chromosomes and the solution found by the constructive heuristic STT. The heuristic solution is encoded by the following value in each element of the vector:

$$\lambda[i] = \frac{\rho_i}{\rho_{max}}$$

where ρ_i is defined in (13) and $\rho_{max} = \max\{\rho_i | i = 1, \ldots, n\} + \nu$ and ν is a very small number.

A vector is decoded by sorting the random keys in decreasing order; the first task is tried to be executed by each technician using the second step of the STT heuristic; if at least one feasible insertion position exists, one technician is selected as described in the third step, and these steps are repeated for the second and subsequent tasks, until the last task is tried to be scheduled.

4.1.2. BRKGA-Cluster:

In the Cluster heuristic, some technicians are assigned to a specific cluster to perform profitable tasks and, after that, the remaining tasks are scheduled considering all regions simultaneously. Following this idea, this BRKGA uses n random keys to represent the priority of the

n tasks and one random key (the last random key of the vector) to determine the percentage of technicians assigned to clusters. The initial population of the BRKGA is composed of random chromosomes and the solution found by the constructive heuristic Cluster, which is encoded by the following value in each random key:

$$\lambda[i] = \begin{cases} 0.9 + 0.1 \cdot \frac{\mu_i - \mu_{ideal}}{\mu_{max} - \mu_{ideal}}, & \text{if } \mu_i \geq \mu_{ideal} \\ 0.9 \cdot \frac{\rho_i}{\rho_{max}}, & \text{otherwise} \end{cases}$$

where ρ_i is defined in (13), $\mu_i = \frac{w_i}{p_i}$, $\mu_{max} = \max\{\mu_i | i = 1, \ldots, n\} + \nu$ and ν is a very small number.

A vector λ is decoded using the steps of the Cluster constructive heuristic. In Steps 1 and 2, parameters μ_i and μ_{ideal} are replaced by $\lambda[i]$ and 0.9, respectively (preliminary tests indicated that it is better to fix μ_{ideal} at 0.9 than to let the algorithm find an ideal value for it). The number of technicians assigned to clusters in Step 3 is $n_{tec} = \lceil \lambda[n+1] \cdot m \rceil$. Finally apply Step 4 to assign the remaining tasks. It should be noted that, in this last step, the remaining tasks are initially sorted by decreasing order of $\lambda[i]$ (initial stage of the NT heuristic).

4.1.3. BRKGA-A: assigning tasks and building random routes

In this version of the BRKGA, chromosomes are also vectors composed of n random keys. The initial population comprises randomized solutions and solutions provided by the constructive heuristics. Each decoded solution is transformed into a feasible solution of the problem through the following procedure: each random key of chromosome λ is used to assign one skilled technician to each task; next, to build the route of each technician, tasks are sorted by an index (ψ_i) extracted from the previous random key.

To illustrate this mechanism, Table 2 shows an example with four tasks and three technicians. Two technicians can perform the first task; then, if the key is in the range $(0, 1/2]$, it is assigned to the first technician (or technician number 2); otherwise, the second technician is selected (or technician number 3). Tasks 2 and 3 are similar. For task 4, there are three possibilities of technicians, so ranges are $(0, 1/3]$, $(1/3, 2/3]$, and $(2/3; 1]$. Thus, the technician selected for task i in the chromosome λ is:

$$TC_i = \lceil \lambda[i] \cdot NST_i \rceil$$

Table 2: Skill of technicians in an example with four tasks and three technicians

Task/Technician	1	2	3
1	0	1	1
2	1	1	0
3	1	0	1
4	1	1	1

where TC_i is the n-th technician able to perform task i and NST_i is the number of skilled technicians ($\sum_{k \in K} s_{ik}$) for task i. For example, according to the skills of technicians in Table 2, a chromosome (0.23, 0.94, 0.35, 0.57) assigns task 3 to the first technician, tasks 1, 2, and 4 to the second, and no task is assigned to the third technician.

After assigning all tasks to technicians, tasks of each technician are sorted by an index (ψ_i), which is the percentage of the distance between the random key and the beginning of the subinterval that corresponds to the technician for each task. In this example, task 2 is assigned to technician 2, because the key is in the interval $(1/2, 1]$, so the index (ψ_i) is $\frac{0.94-0.5}{0.5} = 0.88$. The general formula of the index (ψ_i) can be defined as follows:

$$\psi_i = \frac{\lambda[i] - \frac{TC_i - 1}{NST_i}}{\frac{1}{NST_i}} = NST_i \cdot \lambda[i] - TC_i + 1$$

The indices (ψ_i) of the four tasks of the example are, respectively: 0.46, 0.88, 0.70 and 0.71. Therefore, tasks are inserted in the route of technician 2 in the following order: 1, 4 and 2. Starting from the first to the last task, all possible positions in the route of the technician are evaluated for each task and the position that results in a feasible solution with longest idle time after returning to the base (z_k) is chosen. If there is no feasible solution for a task, then it is not performed.

The solutions of the constructive heuristics (STT, NT and Cluster) are added to the initial population of the BRKGA-A with the following value in each key:

$$\lambda[i] = \frac{TC_i - \frac{\rho_i}{\rho_{max}}}{NST_i}$$

If a task was not assigned to any technician by the constructive heuristics, then a technician

(TC_i) is randomly selected and the value of the key is:

$$\lambda[i] = \frac{TC_i - \frac{\nu}{\rho_{max}}}{NST_i}$$

where ν is a very small number.

5. Generation of upper bounds

In order to evaluate the quality of heuristic solutions in medium and large FTSP problem instances, we propose to generate upper bounds by relaxing the constraints of the original model. The first upper bound model is an improvement of the best model proposed by Xu and Chiu [9], while the second model is our proposal to address this problem.

5.1. Model 1: Simplifying travel times and time windows

A possible strategy to obtain an upper bound for the problem at hand involves the construction of a basic MILP model, based on the model presented in Section 2, that disregards travel times and time windows. This model can be generated by deleting all restrictions of the original model, except restriction (2), and including a new restriction to ensure that the sum of the processing times of the tasks assigned to each technician does not overcome his/her daily working hours. However, to improve this model, the travel time and time window concepts can be reintroduced in a simplified form. For time windows, a set E was defined, containing pairs of conflicting tasks, i.e., tasks i and j that can not be performed by the same technician due to the time window restrictions. Set E is defined as follows:

$$E = \{(i,j) | i \neq j \in J, e_i + p_i + c_{ij} > l_j - p_j \text{ and } e_j + p_j + c_{ji} > l_i - p_i\} \tag{14}$$

To estimate travel times, for each technician k a minimum previous time δ_{ik} immediately before the execution of task i was calculated, which can be the real travel time or the waiting time (difference between the time window of task i and its potential predecessor j). Potential predecessors are the base and tasks j that technician k can execute ($s_{jk} = 1$) before task i. The

previous time of the base is $\tau_{0ik} = \max\{c_{0i}, e_i - a_k\}$ and the previous time of a task j $(1 \leq j \leq n)$ is:

$$\tau_{jik} = \begin{cases} \max\{c_{ji}, e_i - l_j\}, & \text{if } s_{ik} = s_{jk} = 1 \text{ and } e_j + p_j + c_{ji} \leq l_i - p_i \\ b_k - a_k, & \text{otherwise} \end{cases} \tag{15}$$

Therefore, δ_{ik} is defined as follows:

$$\delta_{ik} = \min\{\tau_{jik} \,|\, 0 \leq j \leq n, j \neq i\} \tag{16}$$

Finally, to determine the minimum travel time of each technician k to return to the base, parameter ϕ_k was established as follows:

$$\phi_k = \min\{c_{i0} \,|\, i \in J, s_{ik} = 1\} \tag{17}$$

The complete model follows:

$$\max \sum_{i \in J} \sum_{k \in K | s_{ik}=1} \frac{w_i y_{ik}}{MW} + \frac{1}{MZ} \cdot \left(\sum_{k \in K} (b_k - a_k) - \sum_{i \in J} \sum_{k \in K | s_{ik}=1} (p_i + \delta_{ik}) y_{ik} \right) \tag{18}$$

s.t.

$$\sum_{\substack{k \in K \\ | s_{ik}=1}} y_{ik} \leq 1 \qquad i \in J \tag{19}$$

$$\sum_{\substack{i \in J \\ | s_{ik}=1}} (p_i + \delta_{ik}) y_{ik} + \phi_k \leq b_k - a_k \qquad k \in K \tag{20}$$

$$y_{ik} + y_{jk} \leq 1 \qquad (i,j) \in E, k \in K, s_{ik} = s_{jk} = 1 \tag{21}$$

$$y_{ik} \in \{0,1\} \qquad i \in J, k \in K, s_{ik} = 1 \tag{22}$$

The objective function (18) is similar to the MILP model presented in Section 2. The difference is the secondary objective: instead of idle time of technicians (z_k), we considered the sum of the total number of working hours and subtracted the processing time (p_i) and the minimum previous time (δ_{ik}) of all assigned tasks. Constraints (19) ensure that no task is assigned to more than one technician. Constraints (20) guarantee that the sum of processing

times, the minimum previous time of the tasks, and the minimum time to return to the base does not exceed the working hours of the technician. If it is impossible for the same technician to execute two tasks, this pair belongs to set E and restrictions (21) ensure that they are not assigned to the same technician.

It should be highlighted that, although this model is based on the best model proposed by [9] to obtain an upper bound, four improvements were carried out: *i)* in set E, we included the travel time (c_{ij}) to determine whether the two tasks are conflicting; *ii)* when calculating the minimum previous time for each task i, we took into account only the tasks j that can indeed be executed before (as shown by equation (15)); *iii)* the minimum time for each technician k to return to base (ϕ_k) was included in restriction (20); and *iv)* we generated variables y_{ik} only when $s_{ik} = 1$.

5.2. Model 2: Limiting the working hours of technicians and excluding unfeasible combinations of tasks

In this model, instead of restrictions (20) and (21) presented in Section 5.1, another approach was adopted to represent the travel time and time window concepts. First, for each technician k, a complete enumeration of all eligible tasks, i.e. $s_{ik} = 1$, was performed to determine the maximum sum of processing times ($pmax_k$) that can be executed by this technician respecting his/her work time restriction, the travel times, and the time windows of tasks. Similarly, the maximum feasible weighted processing time ($w_i p_i$) for each technician k ($pwmax_k$) was determined. To avoid the generation of solutions that exceed these limits, restrictions (25) and (26) were added to the model.

To eliminate unfeasible solutions, instead of set E (see equation (14)), new restrictions were included. For each technician k, considering the eligible tasks, all possible groups with up to four tasks (A_1, A_2, A_3, and A_4) were generated. For each one of these groups, a complete enumeration was performed to verify whether a feasible solution could be found. If there is no feasible solution (due to the paths that violate the time window constraints), restrictions (27)-(30) were added to exclude that specific group. A similar idea can be found in Ascheuer et al. [31].

The MILP model follows:

$$\max \sum_{i \in J} \sum_{\substack{k \in K \\ |s_{ik}=1}} \frac{w_i y_{ik}}{MW} + \frac{1}{MZ} \cdot \left(\sum_{k \in K} (b_k - a_k) - \sum_{i \in J} \sum_{\substack{k \in K \\ |s_{ik}=1}} (p_i + \delta_{ik}) y_{ik} \right) \tag{23}$$

Subject to:

$$\sum_{\substack{k \in K \\ |s_{ik}=1}} y_{ik} \leq 1 \qquad i \in J \tag{24}$$

$$\sum_{\substack{i \in J \\ |s_{ik}=1}} p_i y_{ik} \leq pmax_k \qquad k \in K \tag{25}$$

$$\sum_{\substack{i \in J \\ |s_{ik}=1}} w_i p_i y_{ik} \leq pwmax_k \qquad k \in K \tag{26}$$

$$y_{ik} \leq 0 \qquad (i,k) \in A_1 \tag{27}$$

$$y_{i_1 k} + y_{i_2 k} \leq 1 \qquad (i_1, i_2, k) \in A_2 \tag{28}$$

$$y_{i_1 k} + y_{i_2 k} + y_{i_3 k} \leq 2 \qquad (i_1, i_2, i_3, k) \in A_3 \tag{29}$$

$$y_{i_1 k} + y_{i_2 k} + y_{i_3 k} + y_{i_4 k} \leq 3 \qquad (i_1, i_2, i_3, i_4, k) \in A_4 \tag{30}$$

$$y_{ik} \in \{0, 1\} \qquad i \in J, k \in K \tag{31}$$

Where:

$A_1 = \{(i,k) | i \in J, k \in K, s_{ik} = 1, a_k + c_{0i} + p_i > l_i \text{ or } e_i + p_i + c_{i0} > b_k\}$

$A_2 = \{(i_1, i_2, k) | i_1, i_2 \in J, k \in K, s_{i_1 k}, s_{i_2 k} = 1, (i_1,k), (i_2,k) \notin A_1, \text{all possible sequences of tasks } i_1 \text{ and } i_2 \text{ are infeasible for the technician } k\}$

$A_3 = \{(i_1, i_2, i_3, k) | i_1, i_2, i_3 \in J, k \in K, s_{i_1 k}, s_{i_2 k}, s_{i_3 k} = 1, (i_1,k), (i_2,k), (i_3,k) \notin A_1, (i_1,i_2,k), (i_1,i_3,k), (i_2,i_3,k) \notin A_2 \text{ and all possible sequences of tasks } i_1,\ i_2 \text{ and } i_3 \text{ are infeasible for the technician } k\}$

$A_4 = \{(i_1, i_2, i_3, i_4, k) | i_1, ..., i_4 \in J, k \in K, s_{i_1 k}, ..., s_{i_4 k} = 1, (i_1,k), ..., (i_4,k) \notin A_1, (i_1,i_2,k), ..., (i_3,i_4,k) \notin A_2, (i_1,i_2,i_3,k), ..., (i_2,i_3,i_4,k) \notin A_3 \text{ and all possible sequences of tasks } i_1,\ i_2,\ i_3 \text{ e } i_4 \text{ are infeasible for the technician } k\}$

Other models to obtain a stronger upper bound were tested, such as replacing integer variables with real ones in the original model (Section 2) or using intermediate models between the

models presented in this paper, but they did not show superior performance.

6. Numerical experiments

This section describes the instances and presents the results obtained by the proposed approaches. Codes were written in C programming language and tests were conducted on a 2.93 GHz Intel(R) 870 with 16 GB of RAM memory. All instances and results are available at http://www.pro.poli.usp.br/professores/dronconi/.

6.1. Instances

A total of 1040 instances were generated in the experiments, with a considerable diversity of the parameters that can impact on the behavior of the heuristics: geographical distribution of customers, size of time windows of tasks, percentage of tasks with time windows, processing time, priority of tasks, number of tasks and technicians. Four different geographical distributions of the tasks were used: random uniform distribution (R), clustered distribution (C), semi-clustered distribution (RC), and radial (RAD) distribution. The first three distributions were suggested by Solomon [32] and the last one is a new distribution proposal where there is a higher population density in the central part of the city, which is reduced towards the outskirts of the city. For each geographical distribution, 20 instances were generated for 13 different amounts (or cases) of tasks and technicians. Table 3 shows the number of tasks and technicians for each case.

Instance parameters were generated in the following range of discrete uniform distribution:

- Priority of tasks (w_i): 1 (low), $2, \ldots, 10$ (high).
- Processing time of each task (p_i): $30, 35, \ldots, 120$ minutes.
- Percentage of tasks with time windows: 25, 50, 75 or 100%.
- Beginning of time windows of tasks (e_i): $7, \ldots, 19$ hours.
- Size of time windows of tasks: $2, \ldots, 8$ hours.
- End of time windows of tasks (l_i): e_i plus the size of the window.

Table 3: Dimension of the instances generated

Case	#tasks (n)	#technicians (m)
1	16	2
2	26	2
3	30	3
4	39	3
5	45	7
6	64	5
7	80	13
8	100	10
9	120	20
10	150	25
11	200	33
12	500	83
13	999	166

- Beginning of time windows of technicians (a_k): $7, 7.5, 8, \ldots, 12$ hours.
- End of time windows of technicians (b_k): $a_k + 9$ hours.
- Skill of technician k to perform task i (s_{ik}): 0 or 1.
- Travel time: all tasks are located in a region where the travel time between any two locations is less than or equal to 1.5 hour.

Taillard's random number generator and seeds (see [33]) were used to generate random numbers for the discrete uniform distribution for the parameters described above. The radial geographical distribution was obtained by a continuous uniform distribution, also generated by the system proposed by Taillard.

6.2. The MILP model

The MILP model presented in Section 2 was solved using the ILOG CPLEX software, version 12.6, with limited processing time of one hour. The optimum result was reached for 52 problems in case 1, 10 problems in case 2, 6 problems in case 3, and 4 problems in case 4. In cases 5 to 7, the CPLEX software found feasible solutions for all instances, and in cases 8 to 13 no feasible solutions were obtained.

Table 4 shows the percentage difference between the lower bound (LB_{cplex}) and the upper bound (UB_{cplex}) of the objective function provided by the CPLEX software, i.e.:

$$Diff\% = 100 \cdot \frac{UB_{cplex} - LB_{cplex}}{UB_{cplex}}$$

The first column identifies the case. The second, third and fourth columns present the minimum, the average, and the maximum percentage difference in each case, respectively. Table 4 suggests that the difference between the lower and the upper bound depends on the ratio $\frac{n}{m}$ (the average difference is higher when the ratio is higher). Note that, for a similar ratio, the average percentage difference grows as problem size increases. This behavior was expected because, as mentioned in Section 1, the FTSP is a generalization of the vehicle routing problem with time windows, which is NP-hard.

Table 4: Percentage difference ($Diff\%$) for each case

Case	Min.	Avg.	Max.
1	0.0	5.2	25.8
2	0.0	33.3	50.6
3	0.0	24.9	37.5
4	0.0	38.2	51.6
5	0.0	11.9	20.7
6	0.0	42.0	52.8
7	1.0	20.8	33.7

6.3. Upper bounds

In order to measure the quality of the proposed upper bounds, the results of solving Model 1 (UB_1) and Model 2 (UB_2) were compared with an upper bound from the literature (UB_0). This upper bound was proposed by [9] and was obtained through the resolution of Model 1 without restriction (21) and the δ parameter. All models were solved using the software CPLEX 12.6 with a limited processing time of one hour.

For cases 1-4 ($n < 40$), a comparison with optimal solutions was also conducted. The MILP model described in Section 2 was solved for these instances using the CPLEX solver with an execution time limit of ten hours (starting from the best final solution obtained by the BRKGA versions). A complete enumeration was also applied aiming to achieve the maximum number of optimal values. All instances of cases 1 and 2, 58 from case 3, and 18 from case 4, totaling 236 instances, were optimally solved.

Table 5 shows the average CPU time to solve the models that obtained UB_0, UB_1, and UB_2, the number of times that the best upper bound was given by the resolution of the analyzed model ($\#best$), and the average percentage difference between the value of the upper bound ($UB_\#$) (where # identifies the upper bound) and the optimal value, i.e.:

$$GAP_{Opt} = 100 \cdot \frac{Opt - UB_\#}{Opt} \tag{32}$$

The bold numbers in Table 5 and in the following tables indicate the best result of each

Table 5: Comparisons between the analysed upper bounds and the optimal values.

Case	UB_0			UB_1			UB_2		
	CPU (s)	GAP_{Opt}	#best	CPU (s)	GAP_{Opt}	#best	CPU (s)	GAP_{Opt}	#best
1	0.2	-20.6	0	0.3	-10.7	3	0.3	**-3.3**	**80**
2	0.3	-28.4	0	0.3	-15.9	1	0.6	**-8.0**	**79**
3	0.4	-26.8	0	0.5	-15.6	0	1.0	**-6.2**	**80**
4	0.7	-36.0	0	1.0	-20.3	0	3.3	**-9.2**	**80**
Avg.	0.4	-27.9		0.5	-15.6		1.3	**-6.7**	

Table 6: Comparisons between the proposed upper bounds and UB_0.

Case	UB_0		UB_1			UB_2		
	CPU (s)	#best	CPU (s)	$Diff_{UB0}$	#best	CPU (s)	$Diff_{UB0}$	#best
5	0.1	0	144.1	1.8	2	396.5	**3.9**	**80**
6	148.2	0	674.8	6.7	0	1086.0	**11.9**	**80**
7	0.2	0	10.8	1.2	16	152.3	**1.8**	**80**
8	1260.7	0	3326.5	4.4	0	1703.3	**7.7**	**80**
9	0.3	0	0.9	**0.8**	**80**	-	-	-
10	0.3	0	1.1	**0.5**	**80**	-	-	-
11	0.6	0	2.3	**0.4**	**80**	-	-	-
12	3.5	0	54.8	**0.1**	**80**	-	-	-
13	16.8	18	335.2	**0.0**	**80**	-	-	-
Avg.	159.0		505.6	1.8		834.5	6.3	

row. As it can be seen, for all considered cases the upper bounds provided by the proposed models were tighter than the one obtained using the model of the literature (UB_0). It can also be observed that the upper bound provided by Model 2 is tighter than the one obtained using Model 1.

For cases 5-13, a comparison among upper bounds was conducted. Table 6 presents the average CPU time required to solve each model, the number of times each bound is the tighest bound, and the average percentage difference between UB_1 or UB_2 and UB_0 ($Diff_{UB0}$). For cases 9-13, CPLEX was not able to solve Model 2, probably due to the significant increase in the number of constraints [1]. As expected, figures in this table shows that there is a trade-off between the quality of the bound obtained and the computational effort required to compute the bound. Model 2 delivers tighter bounds but cannot be solved when applied to instances of cases 9-13; while Model 1 can be solved for all instances which deliver good quality bounds. It is worth noting that the bounds obtained by solving Model 1 are always better than or equal

[1] For this reason, in the resolution of cases 7 and 8 the restrictions (30) was excluded.

to UB_0 with a reasonable running time. It must also be highlighted that, when considering the 640 instances for which UB_0, UB_1, and UB_2 were successfully computed, the bound computed by solving Model 2 was strictly the best bound in 618 instances.

6.4. Constructive heuristics

The Cluster heuristic achieved the best results with the following parameters: $\mu_{ideal} = 10$, $t_{cluster} = 12$ minutes, and $n_{tec} = 0.8m$. The heuristics STT and NT had no parameters to be calibrated [2] The results of constructive heuristics were analyzed by the gap between the upper bound (UB) and the solution value (SV) provided by the method being analyzed, i.e.:

$$GAP_{UB} = 100 \cdot \frac{UB - SV}{UB}$$

Table 7 shows the minimum, average, and standard deviation of GAP_{UB} considering the 80 instances of each case for the three proposed constructive heuristics and the heuristic developed by Xu and Chiu [9], henceforth denoted by XC. The NT constructive heuristic is similar to the constructive heuristic proposed by Xu and Chiu, but there are two differences: first, in Step 1, the authors sort the tasks in order of decreasing ratio of priority per processing time ($\frac{w_i}{p_i}$) and, in case of a tie, the second criterion is the decreasing value of $l_i - p_i$; secondly, there is no second decision criterion in Step 3.

Considering the average GAP_{UB}, the proposed heuristics outperformed the XC heuristic in all cases. The Cluster heuristic provided the best average GAP_{UB} and the best average result in 6 cases. For the two largest instances (with 500 and 999 tasks) in the STT heuristic, the upper bound gaps were, on average, 3.5 and 2.6%: it is a noteworthy result, since the maximum run time of this constructive heuristics to solve an instance was 0.21 seconds. Regarding the known optimal solutions for cases 1 to 4, the average GAP_{Opt} (given by equation (32)) were 6.7% (STT), 6.6% (NT), 6.4% (Cluster), and 8.0% (XC). It should be highlighted that the STT and Cluster heuristics achieved 36% of the best results among all constructive heuristics, while NT achieved 29% and XC only 10%.

[2] In calculating the parameter ρ, the time unit adopted was hours.

Table 7: Minimum (Min.), Average (Avg.), and Standard Deviation (σ) of GAP_{UB} of the constructive heuristics

Case	STT			NT			Cluster			XC		
	Min.	Avg.	σ	Min.	Avg.	σ	Min.	Avg.	σ	Min.	Avg.	σ
1	0.2	8.1	5.1	0.2	**7.8**	4.5	0.2	8.4	4.6	0.2	9.7	4.6
2	0.3	13.3	4.9	0.3	13.5	4.5	0.3	**13.0**	5.2	0.4	14.2	5.3
3	0.2	12.5	4.1	1.3	12.6	4.0	1.3	**11.9**	4.0	3.9	13.5	3.9
4	4.7	15.8	4.4	1.3	**15.4**	4.5	1.3	15.7	4.9	6.1	17.8	5.1
5	2.4	9.6	3.1	1.6	**9.1**	3.0	2.8	9.2	2.7	1.6	10.9	3.1
6	4.7	17.7	3.8	4.6	17.7	3.7	4.6	**17.2**	3.7	9.9	18.9	3.3
7	4.0	8.4	2.6	2.5	**8.2**	2.8	2.7	8.3	3.0	3.8	9.8	3.1
8	11.0	16.7	2.9	9.7	16.4	2.7	10.1	**16.1**	2.8	10.3	17.6	3.1
9	2.3	7.1	2.9	1.3	**6.8**	2.9	2.5	6.9	2.9	2.5	8.5	3.3
10	1.5	**6.2**	2.6	1.6	**6.2**	2.7	1.9	**6.2**	2.6	2.1	7.5	3.3
11	1.1	6.0	3.2	1.0	6.0	2.9	1.3	**5.7**	2.9	1.7	7.2	3.3
12	0.8	**3.5**	2.3	1.0	3.9	2.2	0.9	3.8	2.2	1.5	4.7	2.6
13	0.3	**2.6**	2.3	0.6	3.2	2.2	0.5	3.0	2.2	0.9	3.8	2.5
Avg. 1-4 (%)	1.3	12.4	4.6	0.8	**12.3**	4.4	0.8	**12.3**	4.7	2.6	13.8	4.7
Avg. 5-13 (%)	3.1	8.6	2.8	2.7	8.6	2.8	3.0	**8.5**	2.8	3.8	9.9	3.1

6.5. BRKGA

6.5.1. Parameter calibration

Initially, the parameters of the BRKGA should be set, namely: elite group size, number of mutant solutions, probability of the parameterized crossover, population size, and maximum number of generations. Based on Gonçalves and Resende [27], the following parameter values were analyzed:

- elite group size (n_e): 15%, 20%, and 25% of the population.
- new mutant solutions (n_m): 5%, 10%, and 15% of the population.
- probability of the parametrized crossover (p_e): 50%, 60%, 70%, and 80%.

- population size (n_p): up to 1500 chromosomes.

Several tests were conducted to calibrate the degree of similarity (DS). The best results were obtained with an initial value (DS^0), which increased linearly over generations (very similar solutions are accepted in the last generations). The parameter DS^0 was set considering candidate values ranging from 50% to 90%.

Table 8: Calibrated parameters of the BRKGA versions

	BRKGA-STT	BRKGA-Cluster	BRKGA-A
n_e	15%	25%	15%
n_m	15%	15%	10%
p_e	60%	60%	70%
DS^0	80%	60%	55%

All BRKGA versions with different combinations of parameter values were applied to 104 instances (10% of the instances of each case). Table 8 shows the configuration of n_e, n_m, p_e, and DS^0 that provided the best results for each BRKGA. The population size (n_p) adopted was 1000 (for cases 1 to 11), 500 (case 12), and 200 (case 13). Note that, as the problem size increases, the cost of evaluation and manipulation of each chromosome increases significantly. Due to this fact, the n_p value for cases 12 and 13 was limited to guarantee an acceptable execution time. Finally, the maximum number of generations was identified for each BRKGA version to reach "convergence" for all tested instances. The convergence of the BRKGA was characterized as the lack of improvement (or an improvement smaller than 0.01%) in the last generations. In Appendix A, Table A.11 presents these values for each BRKGA.

Regarding the elite population diversification, Figure 1 illustrates the BRKGA performance using the usual elite set and the elite set updated by the similarity measure. The average $Diff_{CH}$ of eight instances in case 8 (with 100 tasks and 10 technicians) is presented, where:

$$Diff_{CH} = 100 \cdot \frac{BR - CH}{CH} \tag{33}$$

where BR and CH are the solution values obtained by the BRKGA and Cluster heuristics, respectively. Figure 1(a) shows that the average $Diff_{CH}$ increased from 7.3% to 10.5% in the

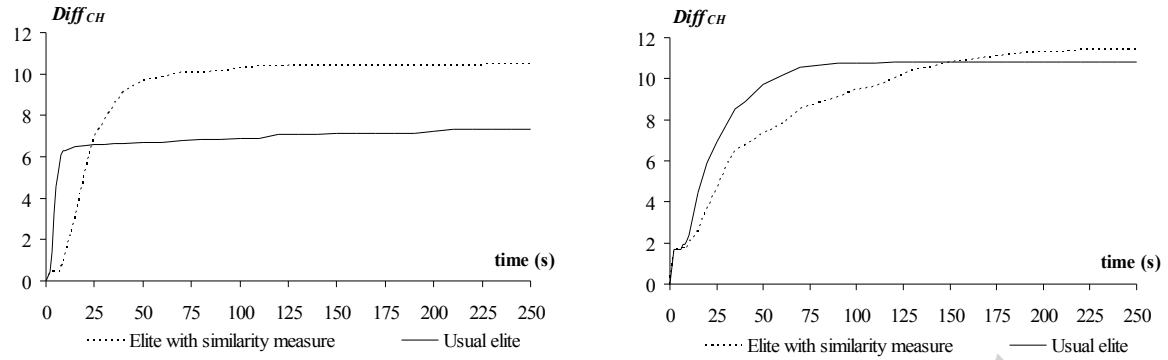

Figure 1: Performance of the BRKGA with the similarity measure and the usual elite: a) BRKGA-A; b) BRKGA-STT.

BRKGA-A using the similarity measure. Likewise, in Figure 1(b), the average improvement of BRKGA-STT increased from 10.8% to 11.4%. Despite this smaller improvement, it should be observed that this version of the BRKGA-STT continues to improve the solution after 125 seconds, unlike the BRKGA-STT with usual elite. The behavior of the similarity measure in the BRKGA-Cluster was similar to BRKGA-STT (the average $Diff_{CH}$ increased from 10.9% to 11.0%) and its chart was omitted.

The performance of the similarity measure was analyzed by an experiment involving 140 instances with the number of tasks ranging from 45 to 200 (for cases 5 to 11, respectively). All geographical distributions were considered: R, C, RC and RAD. On average, BRKGAs that used the similarity measure improved the results of the BRKGAs with traditional elite by 2% (BRKGA-A), 0.3% (BRKGA-STT), and 0.2% (BRKGA-Cluster). This analysis can be improved by calculating the number of instances where this new version of BRKGA obtained strictly better results than BRKGA for the traditional elite: 100% (BRKGA-A), 79% (BRKGA-STT), and 71% (BRKGA-Cluster). Therefore, it was concluded that the similarity function improved the results of all BRKGA versions, especially BRKGA-A, which had a higher premature convergence tax.

6.5.2. Performance of the BRKGA

To compare the BRKGA versions proposed here, a prefixed time (*limit_time*) that would be sufficient for all BRKGAs to reach the convergence was adopted as the stopping criterion.

Table 9: Comparisons among the BRKGA values and the optimal values

Case	# Opt	BRKGA-STT		BRKGA-Cluster		BRKGA-A	
		#Opt.	GAP_{Opt}	*#Opt.*	GAP_{Opt}	*#Opt.*	GAP_{Opt}
1	80	**79**	**0.01**	**79**	**0.01**	**79**	**0.01**
2	80	**79**	**0.01**	78	0.60	77	0.78
3	58	**58**		**58**		56	0.36
4	18	**18**		17	0.02	**18**	
Total (Avg.)	236	**234**	**0.01**	232	0.31	230	0.51

The last column of Table A.11 shows the *limit_time* applied for each case. All BRKGA versions were run once on each instance [34, 35].

The average improvement of the metaheuristics in relation to the best constructive heuristic, Cluster, was 6.0% by BRKGA-STT and BRKGA-Cluster and 5.5% by BRKGA-A.

A comparative study was conducted with optimal solutions through the analysis of the gap of optimality presented by each heuristic. Considering the heuristic methods, this gap is calculated using equation (32). Table 9 shows the number of optimal solutions *(#Opt.)* for each case achieved by the exact methods and by the BRKGA versions and the average GAP_{Opt} considering only the instances for which the optimal solution was not achieved by the heuristic method. As shown in last line of this table, BRKGA-STT and BRKGA-Cluster achieved 99% and 98% of the optimal solutions, respectively, and BRKGA-STT obtained the smallest average GAP_{Opt} for the remaining instances (0.01%).

For medium- and large-sized cases, the quality of heuristic solutions was evaluated through a comparison with the best upper bounds, as it can be seen in Table 10. Columns 2–4 of the table show, respectively, the minimum, average, and standard deviation of GAP_{UB} of the results of BRKGA-STT; the following columns show the results for the BRKGA-Cluster and BRKGA-A; and the last three columns present, respectively, the minimum, average, and standard deviation of GAP_{UB} considering the known optimum values (italic numbers). The bold numbers in Table 10 indicate the best average among all BRKGA versions. Analyzing the results of BRKGA-

Table 10: Average (Avg.), Minimum (Min.), and Standard Deviation (σ) GAP_{UB} of the heuristic methods and of the known optimal values

Case	BRKGA-STT			BRKGA-Cluster			BRKGA-A			Optimum		
	Min.	Avg.	σ	Min.	Avg.	σ	Min.	Avg.	σ	Min.	Avg.	σ
1	0.1	**3.1**	2.5	0.1	**3.1**	2.5	0.1	**3.1**	2.5	*0.1*	*3.1*	*2.5*
2	0.2	**7.3**	3.1	0.2	**7.3**	3.1	0.2	**7.3**	3.1	*0.2*	*7.3*	*3.1*
3	0.1	**5.9**	2.7	0.1	**5.9**	2.7	0.1	**5.9**	2.7	*0.1*	*5.8*[a]	*2.7*
4	0.1	**8.6**	2.9	0.1	**8.6**	2.9	0.1	**8.6**	2.9	*0.1*	*8.3*[b]	*3.1*
5	0.1	**2.9**	1.4	0.1	**2.9**	1.5	0.1	3.2	1.5	-	-	-
6	0.1	**9.6**	2.5	0.1	**9.6**	2.5	0.1	9.8	2.6	-	-	-
7	0.0	2.5	1.3	0.0	**2.4**	1.3	0.0	3.0	1.4	-	-	-
8	1.7	8.6	2.1	1.1	**8.4**	2.2	2.6	9.2	2.1	-	-	-
9	0.0	1.9	1.2	0.0	**1.8**	1.2	0.5	2.7	1.4	-	-	-
10	0.0	1.7	1.1	0.0	**1.6**	1.1	0.4	2.6	1.3	-	-	-
11	0.2	**2.0**	1.5	0.2	**2.0**	1.5	0.5	3.0	1.8	-	-	-
12	0.0	**1.7**	1.3	0.2	1.9	1.5	0.3	2.6	1.7	-	-	-
13	0.0	**1.7**	1.8	0.1	2.2	1.9	0.3	2.4	2.1	-	-	-
Avg. 1-4 (%)	0.1	**6.2**	2.8	0.1	**6.2**	2.8	0.1	**6.2**	2.8	*0.1*	*6.1*	*2.8*
Avg. 5-13 (%)	0.2	**3.6**	1.6	0.2	**3.6**	1.6	0.5	4.3	1.8	-	-	-

[a] refers to the average of 58 instances.
[b] refers to the average of 18 instances.

STT and BRKGA-Cluster, in almost all small-sized cases the minimum, average, and standard deviation of GAP_{UB} were similar to results of the optimal value, as expected. For cases 5 to 13, the average GAP_{UB} of these methods is 3.6%, suggesting a good performance of these BRKGA versions. Observe that the medium- and large-sized cases can be grouped into two categories: low ratio (6 to 8) and high ratio of tasks per technician (10 to 13). In the first group (cases 5, 7, 9, 10, 11, 12, and 13), the average GAP_{UB} is 2.1% for BRKGA-STT and BRKGA-Cluster, while an average GAP_{UB} of 9.1% and 9.0%, respectively, was obtained for the second group (cases 6 and 8). This fact is probably related to the quality of the upper bound when this ratio

increases. In the instances where the optimum value is known, problems with this ratio greater than or equal to 10 (cases 2 to 4) presented an average gap from the optimal solution equal to 7.1%, while problems with lower ratio (case 1) showed an average gap of 3.1%.

It should be highlighted that other versions of BRKGA were developed: based on the NT (Nearest Technician) constructive heuristic, using complete enumerations to find the best route for each technician, applying local search to improve the route of each technician, etc. The results of these other versions were omitted, because they did not present superior performance compared to the three versions described in this paper. Another aspect that should be mentioned is that BRKGA showed similar performance in four different geographical distributions of tasks and in different settings of other problem parameters (size of time windows, percentage of tasks with time windows, etc.), i.e., on average, different problem parameters have no great impact on the results.

Aiming to compare the proposed metaheuristic with other methods from the literature, the performance of the BRKGA-STT when applied to the problem addressed by Kovacs et al. [4] was analyzed. In [4] an Adaptive Large Neighborhood Search (ALNS) heuristic was proposed to solve a parallel problem that contains tasks with identical priority but with different time windows. Moreover, in [4] the objective function intends to minimize the costs of displacement and outsourcing. Note that the minimization of outsourcing cost can be understood as the maximization of total priority of the performed tasks; while the idle time of the technicians at the end of the day can be seen as the minimization of displacement cost. However, minor adjustments were made to the STT decoder in order to tackle the different objective function being considered. In Step 2, when analyzing the insertion of a task into a given technician's route, the position of the task is given by the position that implies the shortest total travel time. Moreover, in Step 3, where it is determined in which route the task is actually inserted, the heuristic selects the technician's route with the smallest increase in displacement cost. The numerical experiments were conducted on a 2.93 GHz Intel(R) E7500 with 4 GB of RAM memory.

We considered all "no team" instances from [4] that are instances to which our approach applies, i.e. instances in which tasks can be performed by a single technician. Namely, we

considered the sets "small", "complete", and "reduced" that have 35, 36, and 36 instances, respectively. The term "complete" refers to instances where, due to the large number of available technicians, all tasks can be performed without the use of outsourcing. On the other hand, "reduced" means that, due the limited number of technicians, it is not likely that all tasks will be scheduled (see [4] for details). The BRKGA-STT was applied five times to each instance. The average solution value was compared with the optimal value for the instances in the "small" data set and, for the remaining instances, with the average value found by the ALNS heuristic.

The detailed results for each instance are given in Appendix B (Tables B.12, B.13, and B.14). In the instances of the data set "small", the BRKGA-STT found the optimal solution in all runs, outperforming the ALNS heuristic, which achieved the optimal solutions in all five runs only for 71.43% of the instances (as shown in Table 6 of [4, p.591]). The average run times per instance of the ALNS and BRKGA-STT were 3.72 and 2.99 seconds, respectively. Regarding the "complete" and the "reduced" data sets, the average objective function values found by BRKGA-STT were 1332.32 and 3528.19 (with average run times 108.91 and 116.82 seconds, respectively). As it can be seen in Table 3 (column from 6 to 9) of [4, p.589], the BRKGA-STT presented a competitive performance when compared to eight different versions of the LNS (large neighborhood search) that include the ALNS method. The obtained results are 2.78% and 0.91% above the average objective function value achieved by the ALNS (see Tables 13 and 14 in [4, pp.596–597]). This better performance of the BRKGA-STT in the "reduced" data set was expected since the proposed algorithm was originally designed for a problem where there is a restricted number of technicians and, consequently, a selection of which tasks will be executed will be required. It is important to observe that the numerical results being compared were run on different machines, using probably different compilers and compiling options.

7. Conclusion

This paper presented constructive heuristics and three customized Biased Random Key Genetic Algorithm for the Field Technician Scheduling Problem (FTSP).

For large instances (500 and 999 tasks), the best constructive heuristics provided beneficial results with reduced computational effort. This good performance can be attributed to the

fact that constructive heuristics explore specific characteristics of the FTSP (task priorities, geographic clusters, technician skills, and travel times) and demonstrate the potential of these heuristics to solve larger problems.

The best customized Biased Random Key Genetic Algorithms, BRKGA-STT and BRKGA-Cluster, found 99% and 98% of the optimal solutions for small instances, respectively. For medium- and large-sized cases, the average results were within 3.6% of the upper bound. The fact that these two BRKGA versions outperformed BRKGA-A, the most random version, indicates that the use of more elaborate decoders is valid.

The similarity measure of the elite set has reached the desired effect by preventing premature convergence. The average results of all BRKGA versions was improved (up to 2% on average) and the number of instances where the BRKGAs with the similarity measure obtained strictly better results than the BRKGAs with the traditional elite ranged between 100% and 71%, depending on the BRKGA version. It should be highlighted that the most significant improvements were observed in the BRKGA-A version, which had a critical problem of premature convergence.

As future work, we believe that the similarity measure can be used to improve the performance of customized BRKGAs for other combinatorial problems.

Acknowledgements

The authors would like to thank the anonymous referees whose comments helped a lot to improve this paper. This research has been partially supported by FAPESP (Grants 2010/10133-0 and 2013/07375-0) and CNPq (Grant 141876/2012-3).

Appendix A. Number of generations and stop criterion

Table A.11: Number of generations required to reach convergence (BRKGAs with the similarity measure of the elite set) and the *limit_time* (stop criterion) for each case.

Case	Number of generations			*limit_time*
	BRKGA-STT	BRKGA-Cluster	BRKGA-A	(seconds)
1	10	10	10	3
2	20	30	20	5
3	110	100	150	10
4	200	200	1000	50
5	400	500	1300	75
6	400	600	1400	80
7	400	700	2200	180
8	400	700	2200	250
9	400	1100	4400	480
10	500	1000	5000	600
11	500	1000	5000	900
12	300	200	8000	1800
13	300	200	11000	3600

Appendix B. Detailed results for the instances proposed by Kovacs

The average results obtained by the analyzed methods for the data sets "small", "complete", and "reduced" are given in Tables B.12, B.13, and B.14, respectively. In these tables, the first column presents the names of the instances; the second column, the average solution values obtained by ALNS proposed by Kovacs et al. (2012); the third column, the average solution values found by BRKGA; and in the last column the average CPU times are reported. The bold numbers in Table B.12 refer to the optimal values. In Tables B.13 and B.14, the fourth column shows the percentage difference between the average objective function value obtained

by ALNS (fo_A) and the average objective function value found by BRKGA (fo_B) ($Diff_k\% = 100 \cdot \frac{fo_B - fo_A}{fo_A}$).

Table B.12: Performance of the ALNS and BRKGA in *small no-team* instances

Instance	ALNS	BRKGA	
	Avg.	Avg.	CPU (s)
C101_5x4	**271.70**	**271.70**	0.26
C101_6x6	**927.35**	**927.35**	0.00
C101_7x4	**789.08**	**789.08**	0.07
C103_7x4	**671.06**	**671.06**	32.31
C101_5x4	838.11	**830.00**	15.49
C101_6x6	1181.31	**1154.84**	12.22
C101_7x4	1367.75	**1356.54**	8.51
C201_5x4	**863.08**	**863.08**	0.00
C201_6x6	**1217.10**	**1217.10**	0.00
C201_7x4	**738.35**	**738.35**	0.00
C203_5x4	**835.83**	**835.83**	0.04
C203_6x6	**930.60**	**930.60**	0.03
C203_7x4	**684.98**	**684.98**	0.00
C201_5x4	**859.54**	**859.54**	2.82
C201_6x6	**1203.93**	**1203.93**	0.36
C201_7x4	**1312.21**	**1312.21**	0.16
R101_5x4	**2195.04**	**2195.04**	0.06
R101_6x6	2977.63	**2857.05**	0.06
R101_7x4	**2447.74**	**2447.74**	0.03
R101_5x4	4540.34	**4507.87**	10.57
R101_6x6	5362.77	**5190.32**	10.63
R101_7x4	4573.12	**4463.80**	1.73
R201_5x4	1092.41	**1091.07**	0.17
R201_6x6	**1377.42**	**1377.42**	0.00
R201_7x4	**959.51**	**959.51**	0.02
R203_7x4	**849.47**	**849.47**	0.13
R201_5x4	**1107.51**	**1107.51**	3.34
R201_6x6	**1647.70**	**1647.70**	3.34
R201_7x4	**1553.23**	**1553.23**	2.00
RC101_5x4	950.81	**862.21**	0.07
RC101_6x6	1557.44	**1361.80**	0.02
RC101_7x4	**1669.63**	**1669.63**	0.03
RC201_5x4	**465.25**	**465.25**	0.03
RC201_6x6	**1228.89**	**1228.89**	0.11
RC201_7x4	**967.60**	**967.60**	0.07
Avg.	1480.53	1469.98	2.99

Table B.13: BRKGA applied to *no-team/reduced* instances

Instance	ALNS	BRKGA		
	Avg.	Avg.	$Diff_k\%$	CPU (s)
C101_5x4_noTeam	5733.75	5763.44	0.52	74.41
C103_5x4_noTeam	2782.20	2971.30	6.80	156.49
C201_5x4_noTeam	2755.52	2755.52	0.00	72.00
C203_5x4_noTeam	2392.50	2386.03	-0.27	101.77
R101_5x4_noTeam	5895.38	5636.46	-4.39	90.35
R103_5x4_noTeam	1845.25	1932.31	4.72	106.10
R201_5x4_noTeam	2854.30	2869.57	0.53	134.29
R203_5x4_noTeam	2332.23	2332.27	0.00	104.00
RC101_5x4_noTeam	5164.84	5129.91	-0.68	99.82
RC103_5x4_noTeam	2348.06	2524.94	7.53	152.91
RC201_5x4_noTeam	3091.67	3096.08	0.14	140.42
RC203_5x4_noTeam	2540.35	2536.76	-0.14	154.79
C101_6x6_noTeam	7762.94	7755.07	-0.10	100.77
C103_6x6_noTeam	5028.83	5084.84	1.11	139.38
C201_6x6_noTeam	3299.56	3378.22	2.38	43.62
C203_6x6_noTeam	2465.90	2471.91	0.24	144.17
R101_6x6_noTeam	6152.29	6170.52	0.30	130.97
R103_6x6_noTeam	2329.25	2330.16	0.04	121.08
R201_6x6_noTeam	3536.70	3615.64	2.23	146.89
R203_6x6_noTeam	2446.18	2463.73	0.72	155.31
RC101_6x6_noTeam	5466.44	5508.04	0.76	131.78
RC103_6x6_noTeam	2349.57	2586.16	10.07	146.97
RC201_6x6_noTeam	4519.95	4619.96	2.21	153.52
RC203_6x6_noTeam	2673.72	2711.80	1.42	148.66
C101_7x4_noTeam	5257.90	5275.30	0.33	108.62
C103_7x4_noTeam	2117.44	2091.45	-1.23	152.86
C201_7x4_noTeam	2779.37	2776.38	-0.11	55.86
C203_7x4_noTeam	2282.15	2288.40	0.27	120.31
R101_7x4_noTeam	5381.35	5336.78	-0.83	85.25
R103_7x4_noTeam	2215.84	2210.41	-0.24	109.69
R201_7x4_noTeam	2679.38	2671.42	-0.30	118.49
R203_7x4_noTeam	2209.80	2201.93	-0.36	124.95
RC101_7x4_noTeam	5799.77	5547.86	-4.34	64.77
RC103_7x4_noTeam	2674.54	2753.69	2.96	85.46
RC201_7x4_noTeam	2936.28	2914.19	-0.75	101.45
RC203_7x4_noTeam	2285.17	2316.57	1.37	127.17
Avg.	3510.73	3528.19	0.91	116.82

Table B.14: BRKGA applied to *no-team/complete* instances

Instance	ALNS	BRKGA		
	Avg.	Avg.	$Diff_k$%	CPU (s)
C101_5x4_noTeam	1111.08	1201.68	8.15	92.32
C103_5x4_noTeam	1037.33	1124.55	8.41	138.14
C201_5x4_noTeam	1180.93	1157.56	-1.98	50.05
C203_5x4_noTeam	1049.30	1092.09	4.08	79.35
R101_5x4_noTeam	1685.85	1718.85	1.96	130.60
R103_5x4_noTeam	1249.91	1315.80	5.27	114.57
R201_5x4_noTeam	1448.93	1450.12	0.08	131.01
R203_5x4_noTeam	1106.12	1111.87	0.52	138.59
RC101_5x4_noTeam	1716.07	1737.49	1.25	109.89
RC103_5x4_noTeam	1354.11	1481.46	9.40	127.88
RC201_5x4_noTeam	1607.25	1605.69	-0.10	74.26
RC203_5x4_noTeam	1166.50	1180.38	1.19	81.83
C101_6x6_noTeam	1004.82	1066.40	6.13	102.11
C103_6x6_noTeam	897.86	1076.75	19.92	143.52
C201_6x6_noTeam	821.55	821.54	0.00	32.58
C203_6x6_noTeam	703.10	689.68	-1.91	95.80
R101_6x6_noTeam	1667.43	1677.54	0.61	154.93
R103_6x6_noTeam	1231.49	1249.34	1.45	137.30
R201_6x6_noTeam	1270.26	1300.61	2.39	123.44
R203_6x6_noTeam	951.84	949.89	-0.20	146.46
RC101_6x6_noTeam	1683.96	1703.77	1.18	145.63
RC103_6x6_noTeam	1310.95	1378.68	5.17	150.37
RC201_6x6_noTeam	1403.95	1427.44	1.67	112.59
RC203_6x6_noTeam	1016.71	1034.83	1.78	145.11
C101_7x4_noTeam	1398.95	1456.84	4.14	94.61
C103_7x4_noTeam	1239.22	1309.31	5.66	128.69
C201_7x4_noTeam	1282.18	1256.30	-2.02	29.84
C203_7x4_noTeam	1151.27	1153.28	0.17	64.19
R101_7x4_noTeam	1793.95	1826.76	1.83	105.46
R103_7x4_noTeam	1375.09	1412.75	2.74	137.37
R201_7x4_noTeam	1410.90	1429.36	1.31	79.23
R203_7x4_noTeam	1166.94	1171.01	0.35	126.38
RC101_7x4_noTeam	1844.37	1897.61	2.89	133.94
RC103_7x4_noTeam	1455.33	1528.59	5.03	86.57
RC201_7x4_noTeam	1701.25	1719.80	1.09	113.91
RC203_7x4_noTeam	1241.65	1247.77	0.49	62.41
Avg.	1298.29	1332.32	2.78	108.91

References

[1] C. E. Cortés, M. Gendreau, L. M. Rousseau, S. Souyris, A. Weintraub, Branch-and-price and constraint programming for solving a real-life technician dispatching problem, European Journal of Operational Research 238 (1) (2014) 300–312.

[2] H. Tang, E. Miller-Hooks, R. Tomastik, Scheduling technicians for planned maintenance of geographically distributed equipment, Transportation Research Part E: Logistics and Transportation Review 43 (5) (2007) 591–609.

[3] Y. Li, A. Lim, B. Rodrigues, Manpower allocation with time windows and job-teaming constraints, Naval Research Logistics (NRL) 52 (4) (2005) 302–311.

[4] A. A. Kovacs, S. N. Parragh, K. F. Doerner, R. F. Hartl, Adaptive large neighborhood search for service technician routing and scheduling problems, Journal of Scheduling 15 (5) (2012) 579–600.

[5] H. Hashimoto, S. Boussier, M. Vasquez, C. Wilbaut, A grasp-based approach for technicians and interventions scheduling for telecommunications, Annals of Operations Research 183 (1) (2011) 143–161.

[6] J.-F. Cordeau, G. Laporte, F. Pasin, S. Ropke, Scheduling technicians and tasks in a telecommunications company, Journal of Scheduling 13 (4) (2010) 393–409.

[7] E. Tsang, C. Voudouris, Fast local search and guided local search and their application to british telecom's workforce scheduling problem, Operations Research Letters 20 (3) (1997) 119–127.

[8] V. Pillac, C. Gueret, A. L. Medaglia, A parallel matheuristic for the technician routing and scheduling problem, Optimization Letters 7 (7) (2013) 1525–1535.

[9] J. Xu, S. Y. Chiu, Effective heuristic procedures for a field technician scheduling problem, Journal of Heuristics 7 (5) (2001) 495–509.

[10] A. Dohn, E. Kolind, J. Clausen, The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach, Computers & Operations Research 36 (4) (2009) 1145–1157.

[11] D. L. Overholts II, J. E. Bell, M. A. Arostegui, A location analysis approach for military maintenance scheduling with geographically dispersed service areas, Omega 37 (4) (2009) 838–852.

[12] J. Valente, J. F. Gonçalves, A genetic algorithm approach for the single machine scheduling problem with linear earliness and quadratic tardiness penalties, Computers & Operations Research 36 (10) (2009) 2707–2715.

[13] M. G. Resende, R. F. Toso, J. F. Gonçalves, R. M. Silva, A biased random-key genetic algorithm for the steiner triple covering problem, Optimization Letters 6 (4) (2012) 605–619.

[14] J. S. Brandao, T. F. Noronha, M. G. Resende, C. C. Ribeiro, A biased random-key genetic algorithm for single-round divisible load scheduling, International Transactions in Operational Research 22 (2015) 823–839.

[15] J. F. Gonçalves, P. S. A. Sousa, A genetic algorithm for lot sizing and scheduling under capacity constraints and allowing backorders, International Journal of Production Research 49 (9) (2011) 2683–2703.

[16] A. Duarte, R. Martí, M. Resende, R. Silva, Improved heuristics for the regenerator location problem, International Transactions in Operational Research 21 (4) (2014) 541–558.

[17] C. E. de Andrade, R. F. Toso, M. G. Resende, F. K. Miyazawa, Biased random-key genetic algorithms for the winner determination problem in combinatorial auctions, Evolutionary computation 23 (2) (2015) 279–307.

[18] J. F. Gonçalves, M. G. Resende, A biased random key genetic algorithm for 2d and 3d bin packing problems, International Journal of Production Economics 145 (2) (2013) 500–510.

[19] E. Lalla-Ruiz, J. L. González-Velarde, B. Melián-Batista, J. M. Moreno-Vega, Biased random key genetic algorithm for the tactical berth allocation problem, Applied Soft Computing 22 (2014) 60–76.

[20] J. F. Gonçalves, M. G. Resende, A biased random-key genetic algorithm for the unequal area facility layout problem, European Journal of Operational Research 246 (1) (2015) 86–107.

[21] L. S. Buriol, M. J. Hirsch, P. M. Pardalos, T. Querido, M. G. Resende, M. Ritt, A biased random-key genetic algorithm for road congestion minimization, Optimization Letters 4 (4) (2010) 619–633.

[22] T. F. Noronha, M. G. Resende, C. C. Ribeiro, A biased random-key genetic algorithm for routing and wavelength assignment, Journal of Global Optimization 50 (3) (2011) 503–518.

[23] L. Morán-Mirabal, J. González-Velarde, M. Resende, Randomized heuristics for the family traveling salesperson problem, International Transactions in Operational Research 21 (1) (2014) 41–57.

[24] A. Grasas, H. Ramalhinho, L. S. Pessoa, M. G. Resende, I. Caballé, N. Barba, On the improvement of blood sample collection at clinical laboratories, BMC Health Services Research 14 (1) (2014) 12.

[25] D. E. Goldberg, J. H. Holland, Genetic algorithms and machine learning, Machine Learning 3 (2) (1989) 95–99.

[26] J. C. Bean, Genetic algorithms and random keys for sequencing and optimization, ORSA Journal on Computing 6 (2) (1994) 154–160.

[27] J. F. Gonçalves, M. G. Resende, Biased random-key genetic algorithms for combinatorial optimization, Journal of Heuristics 17 (5) (2011) 487–525.

[28] J. F. Gonçalves, J. J. de Magalhães Mendes, M. G. Resende, A hybrid genetic algorithm for the job shop scheduling problem, European Journal of Operational Research 167 (1) (2005) 77–95.

[29] J. Gonçalves, M. Resende, R. Toso, An experimental comparison of biased and unbiased random-key genetic algorithms, Pesquisa Operacional 34 (2014) 143–164.

[30] V. A. Armentano, D. P. Ronconi, Tabu search for total tardiness minimization in flowshop scheduling problems, Computers & Operations Research 26 (3) (1999) 219–235.

[31] N. Ascheuer, M. Fischetti, M. Grötschel, Solving the asymmetric travelling salesman problem with time windows by branch-and-cut, Mathematical Programming 90 (3) (2001) 475–506.

[32] M. M. Solomon, Algorithms for the vehicle routing and scheduling problems with time window constraints, Operations Research 35 (2) (1987) 254–265.

[33] E. Taillard, Benchmarks for basic scheduling problems, European Journal of Operational Research 64 (2) (1993) 278–285.

[34] M. Birattari, On the estimation of the expected performance of a metaheuristic on a class of instances, Tech. rep. (2004).

[35] M. S. Hussin, T. Stützle, Tabu search vs. simulated annealing as a function of the size of quadratic assignment problem instances, Computers & Operations Research 43 (2014) 286–291.