# The Thief Orienteering Problem: Formulation and Heuristic Approaches

André G. Santos
*Departamento de Informática*
*Universidade Federal de Viçosa*
Viçosa, MG, Brazil
andre@dpi.ufv.br

Jonatas B. C. Chagas
*Departamento de Computação*
*Universidade Federal de Ouro Preto*
Ouro Preto, MG, Brazil
jonatasbcchagas@gmail.com

***Abstract*—This paper introduces the Thief Orienteering Problem (ThOP), a multi-component problem that combines two classic combinatorial problems: Orienteering Problem (OP) and Knapsack Problem (KP). In this problem, a person (called the thief) carries a capacitated knapsack and has a time limit to collect items distributed in a set of cities. The thief has fixed start and end points, begins his journey with the knapsack empty, and travels with speed inversely proportional to the knapsack weight. While there is time, the thief may visit the cities and collect items. The aim of the problem is to determine the thief's route and items to collect in order to maximize the knapsack profit. We formally describe the ThOP by a mixed integer non-linear programming formulation and present two heuristic approaches based on Iterated Local Search (ILS) and Biased Random-Key Genetic Algorithm (BRKGA) metaheuristics. Our results showed that BRKGA outperformed ILS for a large majority of instances.**

***Index Terms*—Orienteering problem, Knapsack problem, Traveling thief problem, Multi-component problems, Mathematical formulation, Heuristic algorithms.**

## I. Introduction

We introduce here the Thief Orienteering Problem (ThOP), a variation of the Orienteering Problem inspired by the Travelling Thief Problem (TTP). The TTP was proposed recently by Bonyadi, Michalewicz and Barone [1]. It is a combination of two classic well-known problems: the Travelling Salesman Problem (TSP) and the Knapsack Problem (KP). The main argument presented by the authors for proposing the TTP is that the benchmarks of classic NP-hard problems do not reflect the main features of real-world problems, so the effective metaheuristics we have for those benchmarks are not necessarily effective for real-world problems. The authors claim that the complexity of real-world problems is not only due to their size, but mainly because they are a combination of two or more optimization sub-problems, and because those sub-problems are interdependent. This means that the solution of a sub-problem affects the quality of the solution of the others, then they should not be solved independently. In the TTP, a person (the thief) must visit every city of a set of $n$ cities (the TSP component) and during the visit may collect items located on those cities to fill his knapsack (the KP component). However, as items are collected, the knapsack becomes heavier, and the thief walks more slowly. The knapsack was rented, and the price to pay is proportional to the time of rent. The thief must then maximize the total profit of the items collected and also minimize the total time of the route. The authors show that the optimal solution of a TTP instance may not include the optimal solution of the KP component neither the optimal solution of the TSP component, proving the interdependence of the combination.

Since the TTP was introduced, a benchmark of 9,720 instances was made available [2], and several heuristics were proposed [3]–[5]. A comparative study of 21 heuristics was made in [6] to build an algorithm that selects the most suited for each type of instance, and an exact algorithm was proposed in [7] to study the quality of the available heuristics for small instances. The TTP was also subject of competitions on leading events of Computational Intelligence and Evolutionary Computation [8], [9]. The TTP has shown to be a challenging problem, although simple to define and built upon well-known problems.

Inspired by the TTP, we introduce here another multi-component problem, the Thief Orienteering Problem (ThOP), built on the Orienteering Problem (OP) instead of the Traveller Salesman Problem. The Orienteering Problem is based on a sport game of orienteering [10]. In the game, the competitors start on a given point, walk through a region visiting checkpoints, and have to return to a control point within a given time. They have a map of the region and is up to them to decide the route through the checkpoints based on their navigation skills and fitness level. In the OP each checkpoint has a score, then the objective is to find the route which maximizes the total score. i.e., whose sum of scores of the checkpoints visited is maximum. The OP has been extensively investigated in the literature and has many variants. For example, in the Team Orienteering Problem (TOP), a set of competitors work together as a team, and the score of the team is the sum of the scores of the checkpoints visited by any member of the team [11]. There is a survey published in 2011 comparing all approaches already proposed up to that moment [12] . In the ThOP variant, a competitor does not score points by just visiting a checkpoint, but has to collect items at the checkpoints and carry them until the end point. Each competitor has a knapsack with a limited capacity to carry the items. Moreover, the speed of the competitor is directly affected by the weight of the knapsack. As in the TTP, as items

978-1-5090-6017-7/18/$31.00 ©2018 IEEE

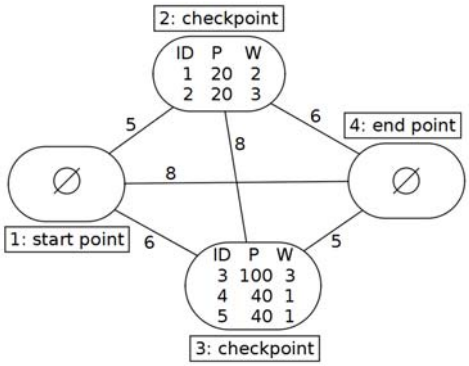



Fig. 1: ThOP example.

are collected, the knapsack becomes heavier, and the speed of the competitor decreases. More formally, let $v_{min}$ and $v_{max}$ denote the minimum and maximum speed of the competitor, when the knapsack of capacity $W$ is respectively full and empty. The speed $v$ of the competitor for a knapsack of weight $w$, $0 \leq w \leq W$ is given by $v = v_{max} - w \cdot (v_{max} - v_{min})/W$. Notice that the value $(v_{max} - v_{min})/W$ is constant and represents the decrease in the speed for each unit of weight of the knapsack.

Figure 1 shows an example with 4 points, the start point (1) and the end point (4) with no items, and 2 checkpoints containing some items. The distances from each pair of points are given in the edges and for each item its profit and weight are shown. Item with $\texttt{ID} = 1$, for example, is located at checkpoint 2, has a profit of 20, and weight 2. For this example, consider $v_{min} = 0.1$, $v_{max} = 1$, $W = 3$ and a time limit $T = 75$ to arrive at the end point.

Although a ThOP solution is composed by a route and a set of items collected along the route, it may be represented by only a sequence of items, as the route may be derived by the location of the items. Collecting the items $\langle 1, 4 \rangle$ has a total profit of $20 + 40 = 60$. It is a feasible solution, as the total weight of these items is 3 and the total walk time is 75, satisfying the limits $W$ and $T$. For the total time we have:

- walk from the start point to point 2 at maximum speed: time is the distance divided by the speed, $5/1.0 = 5$;
- at point 2 collect item 1: the speed decreases to $0.4$;
- walk from point 2 to point 3: walk time is $8/0.4 = 20$;
- at point 3 collect item 4: the speed drops to $0.1$;
- walk from point 3 to the end point: walk time $5/0.1 = 50$.

The solution $\langle 4, 1 \rangle$ has the same items, collected in a different order, and is unfeasible: the total walk time is $77.43$, which is over the limit $T$. The optimal solution for this example is to collect $\langle 3 \rangle$: profit 100, walk time 56. For $T = 20$ the optimal solution would be $\langle 4, 5 \rangle$: profit 80, walk time $18.5$. Notice that for this last case the optimal solution for the knapsack component would be unfeasible for the ThOP.

The ThOP is similar to the TTP, but we point out two main differences: there is no need to visit all the points, and there is an additional constraint, the time limit to arrive at the end point. The last one is a significant difference, because as long as a set of items satisfies the capacity constraint of the knapsack, any permutation of the cities is feasible for the TTP, the difference being the time to travel the route, which impacts the objective function (the price to pay for the knapsack rent). But not for the ThOP, because the travel time is a constraint, as illustrated in the previous example. Due to that difference, the adaptation for the ThOP of methods proposed for the TTP is not trivial. One has to take into account the feasibility of the route. Then, instead of adapting the methods, we propose, as first heuristics for the ThOP, two simple heuristics, that are detailed in Section III.

An immediate application is in reverse logistics, where a company must collect goods from customers for remanufacturing or proper disposal. For each product a benefit is associated. As the vehicle used to collect the products has limited capacity and the driver's working day must be within a limited number of hours, the customers and the items must be chosen properly. The new feature, not present in most works of reverse logistics neither in general transportation problems, is that the speed of the vehicle decreases as it becomes heavier, consequently the time to travel between customers and from them to the depot increases along with the route as items are collected. A dynamic change in the travel time (and/or speed) is considered in time-dependent vehicle routing problems, when congestion and rush hours are taken into account. Here, instead, the change is based on the weight of the load, which is undoubtedly non-linear in real-word, as it depends on several aspects, but can be approximated as linear in some cases in order to be computational tractable.

Besides proposing this new multi-component problem, we propose a method for generating instances for ThOP based on instances for the TTP along with a benchmark with 432 instances, and propose two different heuristics based on the metaheuristic Iterated Local Search (ILS) and on the Biased Random-Key Genetic Algorithm (BRKGA) as starting methods to deal with the problem.

This paper is organized as follows: in Section II we formally describe the problem and give a non-linear mathematical formulation; in Section III we describe in details the two heuristics proposed; then, in Section IV we report computational experiments and analyze the performance of the proposed heuristics; finally, in Section V we present our conclusions and give suggestions for further investigations. In the following we use competitor and thief as terms interchangeable, the same for checkpoints and cities, and collect and steal an item.

## II. Formal definition and mathematical formulation

For the ThOP, we have a set of $n$ points: 1 is the start point, $n$ is the end point, and the remaining $(2, \ldots, n-1)$ are checkpoints. At each checkpoint there is one or more items, and for each item $i$ we have its profit $p_i$ and weight $w_i$. We are also given the capacity $W$ of the knapsack, the time limit $T$ to reach the end point, the thief's maximum and minimum speed, respectively $v_{max}$ and $v_{min}$, and the distance $d_{ij}$ between any pair of points $i$ and $j$. The objective is to find a path from

the start point to the end point filling the knapsack with items carefully chosen at the checkpoints visited on the way, in order to maximize the total profit of knapsack, provided that the capacity of the knapsack is not surpassed and the total walk time is within the time limit.

The following mathematical formulation contains all characteristics of the problem. In order to properly describe it, let $S_i$ be the set of items located at vertex $i$. For each $s \subseteq S_i$, $w_i^s$ represents the total weight of the items contained in $s$ while $p_i^s$ is the total profit of this subset. Moreover, let the constant $\nu = (v_{max} - v_{min})/W$ be the lost in speed for each unit of weight inside the knapsack, and $M'$ and $M''$ be sufficiently large constants. The decisions variables are:

- $x_{ij}^s$ : binary variable that gets 1 if the thief crosses the arc $(i, j)$ after collecting the items of subset $s \subseteq S_i$ and 0 otherwise.
- $q_i$ : variable that reports the weight of the knapsack after collecting the items at city $i$.
- $t_i$ : variable that informs the thief's arrival time at city $i$.

$$\max \sum_{i=1}^{n-1} \sum_{j=2}^{n} \sum_{s \subseteq S_i} p_i^s \cdot x_{ij}^s \tag{1}$$

$$\sum_{j=2}^{n} x_{1j}^{\varnothing} = 1 \tag{2}$$

$$\sum_{i=1}^{n-1} \sum_{s \subseteq S_i} x_{in}^s = 1 \tag{3}$$

$$\sum_{i=1}^{n-1} \sum_{s \subseteq S_i} x_{ij}^s - \sum_{i=2}^{n} \sum_{s \subseteq S_j} x_{ji}^s = 0 \qquad \forall j = 2..n\text{-}1 \tag{4}$$

$$q_j \geq q_i + \sum_{j'=2}^{n} \sum_{s \subseteq S_j} w_j^s \cdot x_{jj'}^s - M' \cdot \left(1 - \sum_{s \subseteq S_i} x_{ij}^s\right) \qquad \begin{array}{l} \forall i = 1..n, \\ \forall j = 1..n \end{array} \tag{5}$$

$$t_j \geq t_i + \frac{d_{ij}}{v_{max} - \nu \cdot q_i} - M'' \cdot \left(1 - \sum_{s \subseteq S_i} x_{ij}^s\right) \qquad \begin{array}{l} \forall i = 1..n, \\ \forall j = 1..n \end{array} \tag{6}$$

$$x_{ij}^s \in \{0, 1\} \qquad \begin{array}{l} \forall i = 1..n, \\ \forall j = 1..n, \\ \forall s \subseteq S_i \end{array} \tag{7}$$

$$0 \leq q_i \leq W \qquad \forall i = 1..n \tag{8}$$

$$0 \leq t_i \leq T \qquad \forall i = 1..n \tag{9}$$

The objective (1) is to maximize the total profit of items collected. The thief must start at city 1 carrying an empty knapsack (2) and reach city $n$ (3), visiting any other city on the way (4). After visiting a city, the thief must leave it after collecting a subset of its items (4), which consequently increases the weight of his knapsack (5) and decreases his speed accordingly, affecting the time to reach the next city (6). The weight of the knapsack and the total walk time should be always within the given limits (8)-(9).

Notice that this set of constraints is enough to avoid subcycles on the route, as (4) assures the route flow and (5)-(6) guarantee that the knapsack weight and the route time is increasing along the route.

Although complete, the formulation as it is cannot be used to solve the problem due to its complexity: the number of variables is exponential in the number of items of a given city, because of the number of possible subsets; and constraint (6) is non-linear, the distance is divided by a continuous variable. We then develop some heuristics, leaving an improved mathematical formulation and exact algorithms for future investigation.

## III. Heuristic solutions

In this section, we describe two heuristic algorithm approaches based on the metaheuristic Iterated Local Search (ILS) and on the Biased Random-Key Genetic Algorithm (BRKGA) in order to obtain high quality solutions. The proposed heuristics and their components are detailed below.

### A. Solution representation and evaluation procedure

In order to facilitate the exploration of the feasible region, we have decided to represent a solution $s$ of the problem by a permutation $\pi = \langle \pi_1, \pi_2, \cdots, \pi_m \rangle$, where $m$ is the number of items available in the cities.

The permutation defines a priority order for the items to be stolen by the thief. From the permutation $\pi$, Algorithm 1 describes how the route $\mathcal{R}$ traveled by the thief is constructed and which items will be stolen and stored in the knapsack $\mathcal{K}$. Initially, the route $\mathcal{R}$ and knapsack $\mathcal{K}$ are empty. For each item $\pi_i$, following the order defined by the permutation, we first verify if after including the item $\pi_i$ the solution remains viable, i.e., if the capacity of the knapsack will not be extrapolated and if the remaining time will be enough for the thief to arrive in the final city (city $n$). If any of the above constraints is not satisfied, the thief will not steal item $\pi_i$. Otherwise, the thief will steal item $\pi_i$. To steal item $\pi_i$, first the thief needs to travel to the city where item $\pi_i$ is located, which we represent by `CITY(`$\pi_i$`)`. If the thief has already stolen another item in the city `CITY(`$\pi_i$`)`, we remove from route $\mathcal{R}$ the previous occurrence of `CITY(`$\pi_i$`)`, and add `CITY(`$\pi_i$`)` at the end of route $\mathcal{R}$. In this way, all items stolen in city `CITY(`$\pi_i$`)` will be inserted into the knapsack only on the last visit to city `CITY(`$\pi_i$`)`, in order to minimize the time spent by the thief.

### B. Local search

Algorithm 2 describes a local search procedure used as subroutines in the main algorithms proposed for the ThOP. It consists of exploring the neighbors of a current solution in order to find a neighbor solution of better quality. If a better solution is found, this solution becomes the current solution, and the process is repeated until it is not possible to improve that solution, i.e., until the search reaches a local optimum.

**Algorithm 1** Evaluation procedure

```
1: procedure f(s)
2:     Let π be the permutation of m items represented in s
3:     R ← K ← ∅
4:     for i ← 1 to m do
5:         if stealing π_i does not violate any constraint then
6:             if CITY(π_i) ∈ R then
7:                 remove city CITY(π_i) from route R
8:             end if
9:             add city CITY(π_i) at the end of route R
10:            add item π_i to the collection plan K
11:            update travel time and knapsack capacity
12:        end if
13:    end for
14:    return total profit value of K
15: end procedure
```

**Algorithm 2** Local search procedure

```
1: procedure LOCALSEARCH(s)
2:     s' ← s
3:     loop
4:         get the best neighbor solution s'' ∈ N(s')
5:         if f(s'') > f(s') then
6:             s' ← s''
7:         else
8:             return s'
9:         end if
10:    end loop
11: end procedure
```

**Algorithm 3** Iterated local search

```
1: procedure ILS()
2:     s ← generate a random solution
3:     s ← LOCALSEARCH(s)
4:     k ← 2
5:     repeat
6:         s' ← SHAKE(s, k)
7:         s'' ← LOCALSEARCH(s')
8:         if f(s'') > f(s) then
9:             s ← s''
10:            k ← 2
11:        else
12:            k ← k + 1
13:        end if
14:    until time limit is reached
15:    return s
16: end procedure
```

**Algorithm 4** Shake procedure

```
1: procedure SHAKE(s, k)
2:     s' ← s
3:     k' ← 0
4:     repeat
5:         get a random neighbor solution s'' ∈ N(s')
6:         s' ← s''
7:         k' ← k' + 1
8:     until k' = k
9:     return s'
10: end procedure
```

In our local search procedure, we use a simple neighborhood structure that consists of exchanging two positions of the permutation $\pi$ of a solution $s$. The set $\mathcal{N}(s)$ represents all neighbors of a solution $s$.

### C. Iterated local search algorithm

Iterated Local Search (ILS) is a metaheuristic that explores the solution space through a local search method and a method that applies perturbations to each local optimum found. The perturbation method is capable and responsible for escaping from local optima and thus of exploring other regions of the solution space (see [13] for more details on ILS metaheuristic).

Since ILS metaheuristic is widely used in the literature to solve combinatorial problems [14], we also have been motivated to propose a method based on it to solve the ThOP. This method is described in detail in the Algorithm 3, which starts with an initial solution $s$ that is generated choosing any random permutation $\pi$ of all $m$ items. Then, it applies a local search (Algorithm 2) in $s$. While the execution time does not reach the maximum established execution time, the algorithm performs a perturbation (Algorithm 4) of size $k$ (= 2, initially) in the current solution $s$, that produces a new solution $s'$. Then, a local search (Algorithm 2) is applied on $s'$, resulting in $s''$. If the solution $s''$ is better than $s$, $s''$ becomes the current solution and the perturbation size is reset to 2; otherwise, the perturbation size increases by one unit. At the end of the process, the best solution found is returned.

### D. Biased random-key genetic algorithm

Biased Random-Key Genetic Algorithm (BRKGA) [15] has been proposed based on the Random-Key Genetic Algorithm (RKGA) [16]. Both are evolutionary metaheuristics that mimic the processes of Darwinian Evolution [17]. Their mechanisms, as well as classic Genetic Algorithms (GAs) (see [18] and [19] for good references), are based on the evolution of a population of individuals, where each individual encodes a solution to the problem at hand. Throughout the evolution, characteristics of individuals with greater fitness tend to survive, thus guiding the algorithm to explore more promising regions of the solutions space.

In a RKGA, each individual is represented as a vector of random-keys, i.e., a vector of real numbers that assume values in the interval $[0, 1]$. A deterministic algorithm is responsible for decoding a feasible solution from a vector of random-keys. RKGA maintains a population of individuals $\mathcal{P}$ of size P, that is, a population of P vectors of random-keys, throughout the evolution of the algorithm. The initial population is created completely randomly, where each random-key of each indi-

vidual is a random number chosen in the interval $[0, 1]$. At each generation of the algorithm, all individuals are decoded and the population $\mathcal{P}$ is divided into two groups: $\mathcal{P}_e$ and $\mathcal{P}_{\bar{e}}$. Group $\mathcal{P}_e$ is formed by the $\mathbf{P_e}$ individuals with greater fitness, while $\mathcal{P}_{\bar{e}}$ is formed by the $\mathrm{P} - \mathbf{P_e}$ others ones, i.e., $\mathcal{P}_{\bar{e}} = \mathcal{P} \setminus \mathcal{P}_e$, where $\mathbf{P_e} < \mathrm{P}/2$ . From any generation $k$ of the RKGA algorithm, a new population $\mathcal{P}^+$ is formed based on the current population $\mathcal{P}$. First, all elite individuals $\mathcal{P}_e$ of generation $k$ are copied to new population $\mathcal{P}^+$ (generation $k + 1$) without any modification. In order to maintain a high diversity of the population, $\mathbf{P_m}$ mutant individuals are added into the population $\mathcal{P}^+$, which are randomly generated, in the same way as the individuals of the initial population are created. To complete the number of individuals P in the population $\mathcal{P}^+$, $\mathrm{P} - \mathbf{P_e} - \mathbf{P_m}$ new individuals are added in $\mathcal{P}^+$ from the uniform crossover operator, which combines pairs of parent individuals from the $\mathcal{P}$ population to produce offspring ones. The uniform crossover operator generates an offspring individual $c$ from the recombination of the vectors of random-key of two parent individuals ($a$ and $b$) randomly selected in $\mathcal{P}$. For each position $i$ of the vector of random-key, the $i$-th random-key of $c$ assumes the value of $i$-th random-key of $a$ with probability 0.5 (50%); otherwise it assumes the value of $i$-th random-key of $b$.

BRKGAs differ from RKGAs in the way parent individuals are selected for mating and also in the way that the mating is made [15]. In a BRKGA, each offspring individual is generated combining one individual selected at random from the elite group $\mathcal{P}_e$ and one from the non-elite group $\mathcal{P}_{\bar{e}}$. In addition, BRKGAs use the parameterized uniform crossover, which generates an offspring with probability $\rho > 0.5$ of choosing the random-keys of the elite parent individual. Algorithm 5 demonstrates how this crossover is performed, where $p_e$ is an elite individual and $p_{\bar{e}}$ is a non-elite one.

**Algorithm 5** Crossover procedure

```
1: procedure CROSSOVER(p_e, p_ē, ρ)
2:     for i ← 1 to m do
3:         if rand (0, 1) ≤ ρ then
4:             p^i ← p_e^i
5:         else
6:             p^i ← p_ē^i
7:         end if
8:     end for
9:     return p
10: end procedure
```

These two mentioned differences in respect to the RKGAs allow the BRKGAs to propagate with greater probability the characteristics of the best individuals for the next generations, increasing the chances of finding better solutions.

Motivated by the fact that the BRKGAs have been applied successfully to solve complex optimization problems [20]–[23], in the remainder of this section, we describe in detail a BRKGA to solve the ThOP.

In our BRKGA, each individual has $m$ random-keys, that is, each individual is represented by a vector of random-keys of size $m$ (remember that $m$ is the number of items available in the cities). We use the decoding proposed in [16], which consists of sorting the vector of random-keys in non-decreasing order and uses the indices of the sorted keys to represent a sequence. Fig. 2 shows an example of decoding a vector of random-keys $p$ of size 6. The random-keys 4, 2, 1, 3, 5 and 6 are sequentially ordered in the $p'$. Thus the sequence $s = \langle 4, 2, 1, 3, 5, 6 \rangle$ is decoded from $p$. This sequence can then be evaluated by the procedure described in the Algorithm 1 to determine the route traveled by the thief and the items stolen by him. In the course of the paper we will refer to this decoding procedure by `DECODE(p)`.

$$p = \langle \underset{1}{0.32}, \underset{2}{0.10}, \underset{3}{0.62}, \underset{4}{0.05}, \underset{5}{0.89}, \underset{6}{0.93} \rangle$$

$$p' = \langle \underset{4}{0.05}, \underset{2}{0.10}, \underset{1}{0.32}, \underset{3}{0.62}, \underset{5}{0.89}, \underset{6}{0.93} \rangle$$

$$s = \langle 4, 2, 1, 3, 5, 6 \rangle$$

Fig. 2: Decoding example.

Algorithm 6 summarizes the previously mentioned steps of our BRKGA. In addition, in order to find local optimal solutions possibly not found only by the evolution mechanism, at every L evolutionary cycles we apply a local search procedure (Algorithm 2) to each individual of the current population.

## IV. Computational experiments

Our two heuristic algorithms (ILS and BRKGA) have been implemented in C/C++ language and compiled with GNU version 4.8.2. All experiments have been sequentially (nonparallel) performed on an Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz x 40 computer with 384GB of RAM, running CentOS Release 6.8 (Final) Kernel Linux 2.6.32-642.1.1.el6.x86_64.

In order to evaluate the quality of the algorithm ILS and BRKGA, we have performed tests with different instance sizes and settings. All instances were created based on the benchmark of instances for the Traveling Thief Problem (TTP) [2]. The only additional parameter is the walk time limit $T$. In order to have a challenging instance, this parameter should be set carefully. If it is too high, it would be useless, as the knapsack capacity $W$ would make the walk time constraint redundant. On the other hand, if it is too small, it would greatly limit the number of items, turning the knapsack capacity constraint useless. A good value would be something less than (or a fraction of) the route time of the optimal solution for the corresponding TTP instance. However, optimal solutions for the TTP are not available. We then derive the parameter $T$ from the best solution we know for the TTP instances, which were found by a *General Variable Neighborhood Search* (GVNS) we proposed for the TTP competition at the GECCO 2017 conference [9]. As far as we know, the best solutions received by the organizers were not made available, but our

**Algorithm 6** Biased random-key genetic algorithm

**procedure** BRKGA ( P, $P_e$, $P_m$, $\rho$, L)
  $s^{best} \leftarrow \varnothing$
  $\mathcal{P} \leftarrow$ generate P vector of random-keys
  $num\_ger \leftarrow 1$
  **repeat**
    **for each** $p \in \mathcal{P}$ **do**
      $s \leftarrow$ DECODE $(p)$
      **if** $num\_ger$ **mod** L = 0 **then**
        $s \leftarrow$ LOCALSEARCH $(s)$
      **end if**
      **if** $f(s) > f(s^{best})$ **then** $s^{best} \leftarrow s$ **end if**
    **end for**
    $\mathcal{P}_e \leftarrow$ select the $P_e$ best individuals (elite) from $\mathcal{P}$
    $\mathcal{P}_{\bar{e}} \leftarrow \mathcal{P} \setminus \mathcal{P}_e$
    $\mathcal{P}^+ \leftarrow \mathcal{P}_e$
    $\mathcal{P}_m \leftarrow$ generate $P_m$ vector of random keys
    $\mathcal{P}^+ \leftarrow \mathcal{P}^+ \cup \mathcal{P}_m$
    **for** $i \leftarrow 1$ **to** $P - P_e - P_m$ **do**
      $p_e \leftarrow$ randomly choose an individual from $\mathcal{P}_e$
      $p_{\bar{e}} \leftarrow$ randomly choose an individual from $\mathcal{P}_{\bar{e}}$
      $\mathcal{P}^+ \leftarrow \mathcal{P}^+ \cup \{$CROSSOVER $(p_e, p_{\bar{e}}, \rho)\}$
    **end for**
    $\mathcal{P} \leftarrow \mathcal{P}^+$
    $num\_ger \leftarrow num\_ger + 1$
  **until** time limit is reached
  **return** $s^{best}$
**end procedure**

method was awarded for having improved the solutions of about 2,500 out of 9,720 instances. We then set different maximum travel times ($T$) as different proportions of the route time of the best TTP solutions found by our GVNS, in order to verify the behavior of the resolution methods according to this characteristic. Therefore, we consider the following features:

- TSP base instance groups: `eil51`, `pr107`, `a280` and `dsj1000`
- number of items per city: `01`, `03`, `05`, and `10`
- item relation type: bounded-strongly-correlated (`bsc`), uncorrelated (`unc`), uncorrelated-similar-weights (`usw`)
- knapsack capacity class: `01`, `05`, `10`
- maximum travel time class: `01`, `02`, `03`

The TSP base groups were arbitrarily chosen from the groups available from the TTP benchmark, from small to large sizes. The number of items per city, the item relation type and the knapsack capacity class are the same defined for the TTP. The maximum travel time classes `01`, `02`, `03` use for parameter $T$ respectively 50%, 75% and 100% of the route time of our solution for the TTP competition. All instances are public available at http://www.dpi.ufv.br/~andre/thop/.

Regarding the parameters of the algorithms, we defined as stopping criteria the execution time equal to $\lceil \frac{m}{10} \rceil$ seconds, which is given in terms of the number of items $m$ of each instance. The ILS algorithm has no other parameter, whereas the BRKGA has five other ones: population size P, elite population size $P_e$, number of mutant individuals $P_m$ inserted in each evolutionary cycle, elite inheritance probability $\rho$, and the parameter L that indicates how often the local search procedure is applied. All parameters were empirically tuned following the parameters value settings recommended in [15]. Our final experiments were performed with $P = 100$, $P_e = 25$, $P_m = 30$, $\rho = 0.6$ and $L = 50$.

Since the ILS algorithm and BRKGA have random components, for each instance, we run 10 times each algorithm with different random seeds and we use the average value of the objective function and the best one found in these 10 runs to compare the algorithms. Therefore, we have two results (average and best) for two algorithms and 432 instances, a total of 1,728 results. The complete results, including details of the solutions, can be found in a supplementary material in Appendix A. In this paper, we present and analyze them grouped by instances and types, in charts shown in Figure 4 and Tables III, IV, II and I.

Before going into details on the analysis of the results, we illustrate solutions of a ThOP instance for different maximum travel times. Figure 3 shows the best result of ILS and BRKGA for instances `eil_01_bsc_01_TT` for increasing travel time limit (from left to right, `TT = 01` to `03`). The green triangle is the start point and the red square the end point. The line represents the route and becomes thicker as the knapsack becomes heavier. Notice how the route becomes longer as the thief has more time, and how the profit improves, sometimes visiting more checkpoints, sometimes changing the checkpoints visited (in search for more profitable items). Notice also how BRKGA tends to find routes with less edges crossing.

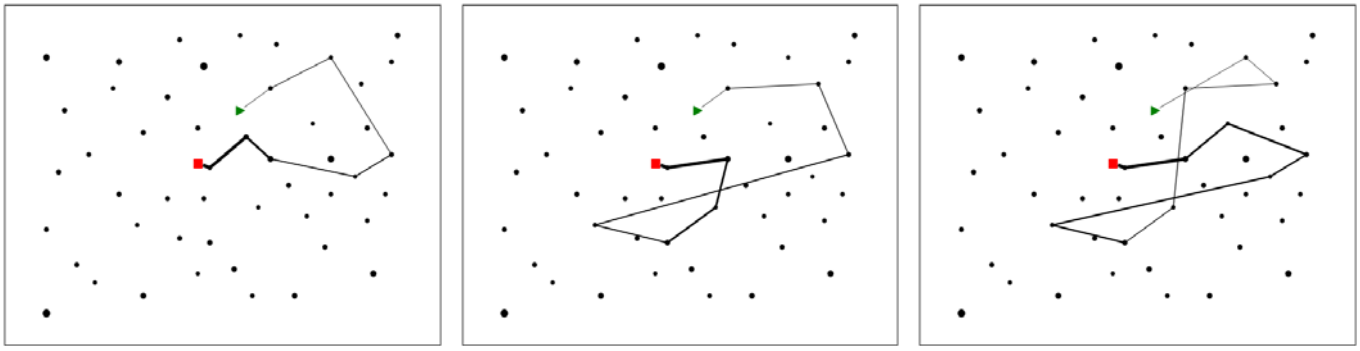
We define the metric $\chi$ that establishes a measure of the convergence quality of the algorithms. Equations (10) and (11) show how the metric $\chi$ is calculated for each algorithm, where $\text{ILS}^{avg}$ and $\text{BRKGA}^{avg}$ refer to the average value of the objective function found by the ILS algorithm and BRKGA, respectively, for the corresponding group, and $\text{ILS}^{best}$ and $\text{BRKGA}^{best}$ are the best solution found by each algorithm. Note that the greater the metric $\chi$, the higher the convergence of the algorithm to the best known solution.

$$\chi^{\text{ILS}} = \frac{\text{ILS}^{avg}}{\max\left(\text{ILS}^{best}, \text{BRKGA}^{best}\right)} \cdot 100\% \tag{10}$$

$$\chi^{\text{BRKGA}} = \frac{\text{BRKGA}^{avg}}{\max\left(\text{ILS}^{best}, \text{BRKGA}^{best}\right)} \cdot 100\% \tag{11}$$

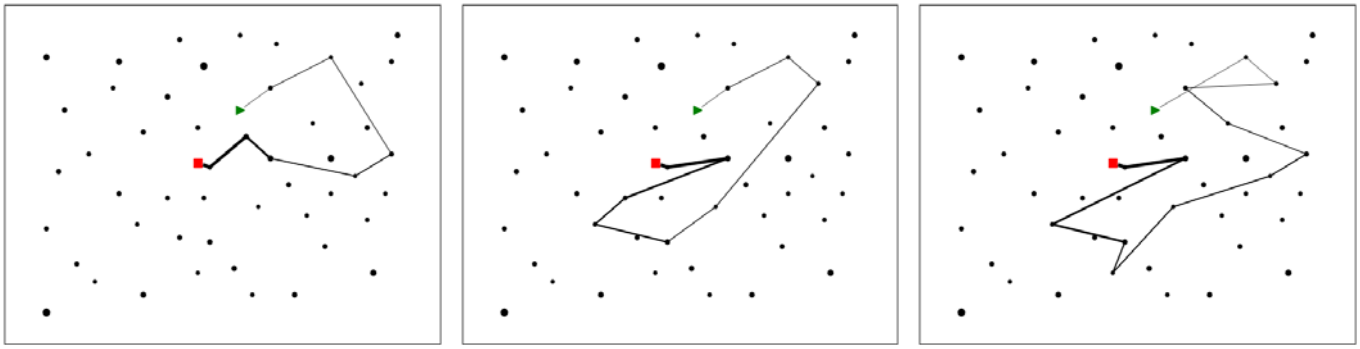
Figure 4 has four charts, which report the results of each TSP base instance group (`eil51`, `pr107`, `a280` and `dsj1000`). The vertical axis (y-axis) of all charts shows instance subgroup in the format `XXX_YY_ZZZ`, where `XXX` informs the TSP base instance group, `YY` the number of items per city and `ZZZ` the relation items type. The other characteristics of the instances (knapsack capacity class and maximum travel time class) were grouped.

ILS



$T$ = 163, Profit = 5554 | $T$ = 244, Profit = 6152 | $T$ = 325, Profit = 6602

BRKGA



$T$ = 163, Profit = 5554 | $T$ = 244, Profit = 6276 | $T$ = 325, Profit = 6825

Fig. 3: Solutions found by ILS and BRKGA for instances eil_01_bsc_01 for different time limits.

TABLE I: Results grouped by number of items per city.

| TSP base group | ILS | | | | BRKGA | | | |
|---|---|---|---|---|---|---|---|---|
| | 01 | 03 | 05 | 10 | 01 | 03 | 05 | 10 |
| eil51 | 95.9 | 92.8 | 81.8 | 68.3 | 97.2 | 94.5 | 93.6 | 92.3 |
| pr107 | 95.1 | 84.5 | 81.6 | 76.1 | 95.0 | 92.4 | 93.1 | 94.3 |
| a280 | 92.4 | 76.3 | 70.7 | 56.0 | 88.8 | 90.9 | 92.1 | 91.2 |
| dsj1000 | 79.3 | 68.3 | 62.2 | 59.6 | 91.3 | 91.7 | 91.5 | 93.2 |

TABLE II: Results grouped by item relation type.

| TSP base group | ILS | | | BRKGA | | |
|---|---|---|---|---|---|---|
| | bsc | unc | usw | bsc | unc | usw |
| eil51 | 86.4 | 82.7 | 84.9 | 95.3 | 93.4 | 94.5 |
| pr107 | 86.1 | 82.5 | 84.4 | 94.8 | 93.1 | 93.2 |
| a280 | 74.5 | 74.1 | 72.9 | 91.8 | 90.0 | 90.5 |
| dsj1000 | 68.9 | 67.3 | 65.9 | 92.2 | 92.4 | 91.1 |

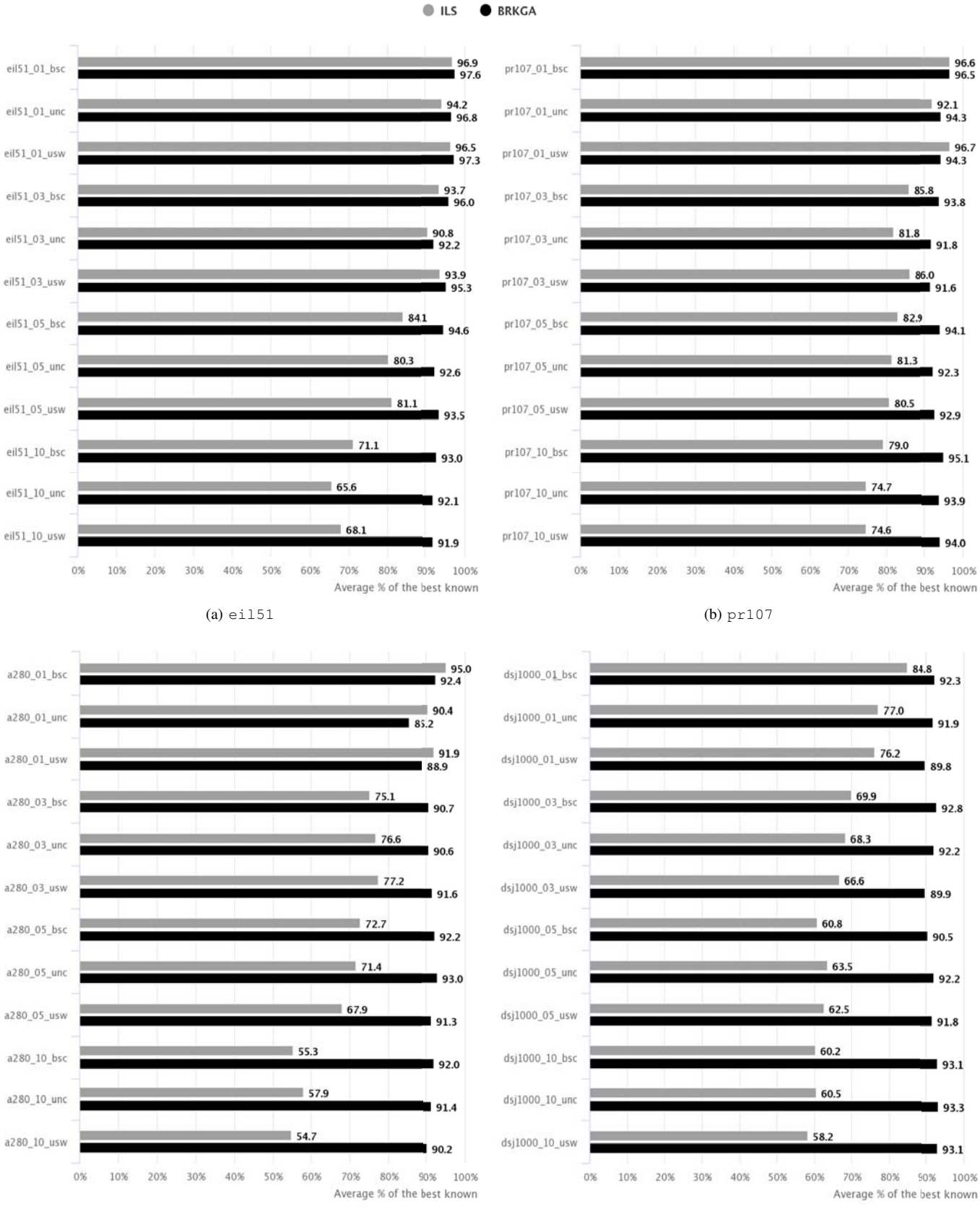
TABLE III: Results grouped by maximum travel time class.

| TSP base group | ILS | | | BRKGA | | |
|---|---|---|---|---|---|---|
| | 01 | 02 | 03 | 01 | 02 | 03 |
| eil51 | 86.7 | 83.7 | 83.6 | 93.9 | 94.6 | 94.8 |
| pr107 | 85.2 | 84.9 | 82.8 | 94.1 | 93.4 | 93.5 |
| a280 | 72.1 | 73.2 | 76.3 | 89.7 | 90.0 | 92.7 |
| dsj1000 | 64.8 | 67.5 | 69.8 | 91.3 | 92.1 | 92.4 |

TABLE IV: Results grouped by knapsack capacity class.

| TSP base group | ILS | | | BRKGA | | |
|---|---|---|---|---|---|---|
| | 01 | 05 | 10 | 01 | 05 | 10 |
| eil51 | 93.2 | 82.7 | 78.2 | 96.1 | 93.8 | 93.3 |
| pr107 | 90.8 | 82.3 | 79.9 | 96.4 | 93.0 | 91.7 |
| a280 | 78.4 | 71.8 | 71.3 | 92.3 | 90.3 | 89.7 |
| dsj1000 | 69.5 | 65.6 | 67.0 | 93.4 | 90.8 | 91.5 |

We may see that the performance of BRKGA remains stable (convergence around 90%) for all groups and all instances, while the convergence of ILS decreases as the number of checkpoints increases and as the number of items increases for the same number of checkpoints, going from 97% to less than 60%. ILS had a better convergence only in the smallest groups of a280 and p107. For a clearer view, see Table I, that shows the results grouped by number of items per city. For instance a280, for example, the convergence of ILS goes from around 92% to 76%, 71% and 56% as the number of items per city goes from 1 to 3, 5, and 10 respectively.




(a) `eil51`

(b) `pr107`

(c) `a280`

(d) `dsj1000`

Fig. 4: Comparative analysis of the convergence of ILS algorithm and BRKGA on the four instance groups.

The behavior also changes for different item relation type, as seen on Table II. Both methods perform better for instances of type `bsc`. And worst for type `unc`, except for instances `dsj1000`, where `usw` had the worst convergence.

Regarding the maximum travel time $T$, the methods tends to have a better convergence as this parameter increases, as reported in Table III, except for instances `p107`, in which the behavior is the opposite. And finally, convergence of both methods are better for instances of knapsack capacity class `01` and worst for `10`, with only two exceptions: worst for `05` in the two largest TSP based groups, `a280` and `dsj1000`.

## V. Conclusions and further investigations

In this paper, we introduced a new combinatorial problem referred to as the Thief Orienteering Problem (ThOP), and proposed two heuristic algorithms (ILS and BRKGA) to solve it. We have tested our approaches for the ThOP with 432 instances created based on instances proposed and available in the literature for the Traveling Thief Problem (TTP).

The results showed a superiority of the BRKGA when compared to the ILS algorithm, mainly for the large instances. We think that this superiority is due to the diversification introduced by the mutant individuals and mainly by the recombination of the individuals that share good characteristics of different individuals, generating other ones with greater fitness and, consequently, better solutions.

As future work, we would like to analyze the performance of heuristics based on other metaheuristics, and generate upper bounds on the optimal solution, using an improved version of the proposed formulation, in order to better attest the quality of the heuristics.

## Acknowledgment

The authors thank Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) and Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) for the financial support of this project.

## Appendix
## Supplementary material

Supplementary material associated with this paper can be found, in the online version, at the following url: http://www.dpi.ufv.br/~andre/thop/

## References

[1] M. R. Bonyadi, Z. Michalewicz, L. Barone. “The travelling thief problem: the first step in the transition from theoretical problems to realistic problems”. In Proceedings of the 2013 IEEE Congress on Evolutionary Computation, pp. 1037–1044, 2013.

[2] S. Polyakovskiy, M. R. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann, “A comprehensive benchmark set and heuristics for the traveling thief problem”. In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 477–484, 2014, ACM.

[3] M. Wagner, “Stealing items more efficiently with ants: a swarm intelligence approach to the travelling thief problem”. In Proceedings of the 2016 International Conference on Swarm Intelligence, pp.273–281, 2016.

[4] M. El Yafrani, B. Ahiod B. “Population-based vs. single-solution heuristics for the travelling thief problem”. In: Proceedings of the 2016 Genetic and Evolutionary Computation Conference, pp 317-324, 2016.

[5] H. Faulkner H, S. Polyakovskiy, T. Schultz, M. Wagner, “Approximate approaches to the traveling thief problem. In: Proceedings of the 2015 Genetic and Evolutionary Computation Conference, pp 385-392, 2015

[6] M. Wagner, M. Lindauer, M. Misir, S. Nallaperuma, F. Hutter. “A case study of algorithm selection for the traveling thief problem”. Journal of Heuristics, 2017.

[7] J. Wu, M. Wagner, S. Polyakovskiy, F. Neumann. “Exact approaches for the travelling thief problem”. In Proceedings of the 11th International Conference on Simulated Evolution and Learning, pp. 110–121, 2017.

[8] “Optimisation of Problems with Multiple Interdependent Components”. Internet: https://cs.adelaide.edu.au/ optlog/CEC2014Comp Jul. 9, 2014 [Jan. 15, 2018]

[9] “Optimisation of Problems with Multiple Interdependent Components”. Internet: http://gecco-2017.sigevo.org/index.html/Competitions [Jan. 15, 2018]

[10] I-M. Chao, B. L. Golden, E. A. Wasil. “A fast and effective heuristic for the orienteering problem”. European Journal of Operational Research, 88(3), pp. 475–489, 1996a.

[11] I-M. Chao, B. L. Golden, E. A. Wasil. “The team orienteering problem”. European Journal of Operational Research, 88(3), pp. 464–474, 1996a.

[12] P. Vansteenwegen, W. Souffriau, D. Van Oudheusden. “The orienteering problem: A survey”. European Journal of Operational Research, 209(1), pp. 1–10, 2011.

[13] H. R. Lourenço, O. C. Martin, and T. Sttzle, “Iterated local search”. International Series in Operations Research and Management Science, pp. 321–354, 2003.

[14] H. R. Lourenço, O. C. Martin, and T. Sttzle, “Iterated local search: Framework and applications”. In Handbook of Metaheuristics, pp. 363–397, 2010.

[15] J. F. Gonçalves, and M. G. Resende, “Biased random-key genetic algorithms for combinatorial optimization”. Journal of Heuristics, pp. 487–525, 2011.

[16] J. C. Bean, “Genetic algorithms and random keys for sequencing and optimization”. ORSA Journal on Computing, pp. 154–160, 1994.

[17] C. Darwin, and G. De Beer, “The Origin of Species by Means of Natural Selection, Or, The Preservation of Favored Races in the Struggle for Life”, 1956.

[18] L. Davis, “Handbook of genetic algorithms”, 1991.

[19] M. Srinivas, and L. M. Patnaik, “Genetic algorithms: A survey”, Computer, pp. 17–26, 1994.

[20] C. Martinez, I. Loiseau, M. G. Resende, and S. Rodriguez, “BRKGA algorithm for the capacitated arc routing problem”. Electronic Notes in Theoretical Computer Science, pp. 69–83, 2011.

[21] A. Castro, M. Ruiz, L. Velasco, G. Junyent, and J. Comellas, “Path-based recovery in flexgrid optical networks”. In 14th International Conference on Transparent Optical Networks, pp. 1–4, 2012, IEEE.

[22] R. M. D. A. Silva, M. G. Resende, P. M. Pardalos, and J. L. Facó, “Biased random-key genetic algorithm for linearly-constrained global optimization”. In 15th Annual Conference Companion on Genetic and Evolutionary Computation, pp. 79–80, 2013, ACM.

[23] A. A. Chaves, L. A. N. Lorena, E. L. F. Senne, and M. G. Resende, “Hybrid method with CS and BRKGA applied to the minimization of tool switches problem”. Computers & Operations Research, pp. 174–183, 2016.

[24] G. Reinelt, “TSPLIB - A traveling salesman problem library. ORSA Journal on Computing”, pp. 376–384, 1991.