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

Efficiently solving the thief orienteering problem with a max-min ant colony optimization approach

Jonatas B. C. Chagas [1,2,*] · **Markus Wagner** [3]

Received: date / Accepted: date

Abstract We tackle the Thief Orienteering Problem (ThOP), which is academic multi-component problem: it combines two classical combinatorial problems, namely the Knapsack Problem (KP) and the Orienteering Problem (OP). In this problem, a thief has a time limit to steal items that distributed in a given set of cities. While traveling, the thief collects items by storing them in their knapsack, which in turn reduces the travel speed. The thief has as the objective to maximize the total profit of the stolen items while also paying for the rented knapsack. In this article, we present an approach that combines swarm-intelligence with a randomized packing heuristic. Our solution approach outperforms existing works on almost all of the 432 benchmarking instances, with significant improvements.

Keywords Ant Colony Optimization · Multi-Component Problems · Knapsack Problem · Orienteering Problem

1 Introduction

Many studies devoted to solving classical combinatorial optimization problems are directly motivated by real-world problems, such as packing problems, scheduling problems, and vehicle routing problems. While already challenging, many real-world problems exhibit multi-component structures: they

Jonatas B. C. Chagas
E-mail: jonatas.chagas@iceb.ufop.br

Markus Wagner
E-mail: markus.wagner@adelaide.edu.au

[*] Corresponding author
[1] Departamento de Computação, Universidade Federal de Ouro Preto, Ouro Preto, Brazil
[2] Departamento de Informática, Universidade Federal de Viçosa, Viçosa, Brazil
[3] School of Computer Science, The University of Adelaide, Adelaide, Australia

comprise a number of interacting combinatorial optimization problems. Multi-component problems are hard to solve as each component has the potential to influence the feasibility as well as the quality of the other components [4].

Among the studied multi-component problems, vehicle routing problems with loading constraints [15] appear to be very frequently investigated. In these problems, tour are to be created for vehicles while constraints and aims of specific loading policies must be taken into account. One of these problems is the Traveling Thief Problem (TTP), which combines two classic well-known and well-studied problems: the Knapsack Problem (KP) and the Traveling Salesman Problem (TSP). The TTP was proposed in 2013 by Bonyadi et al. [3] in order to provide an academic abstraction of multi-component problems for the scientific community. In the TTP, a thief travels across all cities (TSP component) and steals items along the way (KP component). The stolen items are stored in a rented knapsack, for which the thief has to pay a time-dependent fee. The overall objective of the thief is to maximise the following: the total profit of all stolen items, minus the travel time multiplied with the renting rate. As the travel speed is inversely proportional to the total weight of the items in the knapsack, this non-constant cost sets the TTP apart from many vehicle routing problems, such as the environmental prize-collection problem [26].

In this article, we tackle the Thief Orienteering Problem (ThOP) [23], another academic multi-component problem. The ThOP has been proposed based on the TTP, but with different interactions and constraints in mind. It combines the Orienteering Problem (OP) and the Knapsack Problem (KP). In operational research, the OP has been the subject of many studies [13,14]: a traveler starts at a predetermined location, travels through a region visiting checkpoints, and has to arrive at a control point within a given limited time. Because each checkpoint has a score, the participant's objective is to find a route that maximizes the total score, i.e., where the sum of scores of the visited checkpoints is the highest. In the ThOP context, the participant (i.e., the thief) does not score points by just visiting a checkpoint but has to steal them and carry them in their knapsack until the end of their robbery journey. As in the TTP, in the ThOP, the thief has a capacitated knapsack to carry the items. Moreover, as items are collected, the knapsack becomes heavier, and the speed of the thief decreases. There is no knapsack rental fee and the thief only aims to find a route and a set of items that maximizes their total stolen profit.

Although the ThOP and the TTP are related, the ThOP appears to be more practical due to two key differences: in the ThOP (A) the participant does not have to travel through all cities, and (B) the interaction is not defined by the time-dependent rent for the knapsack, but by the time-constraint. Even though the relaxation of difference (A) may seem to be straightforward, handling this constraint is typically reflected in the design of heuristic [28] and exact [29,19] algorithms, with only Chand and Wagner's [6] Multiple Traveling Thieves Problem (MTTP) being an exception. Regarding (B), applications with routing time limit frequently arise in real-world scenarios, where there is insufficient time and/or capacity to visit/met all possible locations. Examples of this include cash logistics [20], urban crowd-sourcing [7], and concert promotion [16].

Santos and Chagas [23] presented a Mixed Integer Non-Linear Programming formulation for the ThOP (although without results) and two simple heuristics. Afterwards, Faêda and Santos [9] proposed a genetic algorithm that performed better on on most instances. We have also addressed the ThOP in a preceding article [5] with a two-phase approach based on Ant Colony Optimization (ACO) and a greedy heuristic to construct the route and the packing plan (stolen items) of the thief. Our ACO is able to find better solutions than other aforementioned algorithms for most instances because of its focus on creating efficient routes.

Here, we describe a number of improvements that we incorporated into our ACO algorithm, which made it more substantially effective with regard to the quality of the solutions found. In our computation experiments, we have investigated the importance of the parameters of our ACO algorithm considering these improvement changes. In addition, to make a fairer comparison among the other algorithms already proposed for the ThOP, we have also investigated the parameters of those algorithms and then evaluate their performances on a broad set of instances according to the results already presented in the literature.

In this article, we begin in Section 2 with a formal description of the TTP, where we also present a mixed integer non-linear programming formulation. Subsequently, we present our new approach for solving the ThOP in Section 3. Then, we report on our computational experiments in Section 4. Lastly, we summarize our present work and outline future work.

2 Problem definition

2.1 Formal definition

The Thief Orienteering Problem (ThOP) can be formally described as follows. There is a set $I = \{1, 2, \ldots, m\}$ of m items and a set $C = \{1, 2, \ldots, n\}$ of n cities. Each item $k \in I$ has a profit p_k and weight w_k associated. In addition, each item is associated with only a single city, but a city can have multiple items. Let us denoted by I_i the set of items localized at city i. From the foregoing definition, $\bigcup_{i \in C} I_i = I$ and $\bigcap_{i \in C} I_i = \varnothing$. The items are scattered among all cities, except cities 1 and n ($I_1 = I_n = \varnothing$). Cities 1 and n are the cities where the thief starts and ends their journey. Let us denote by $A = \{(i, j),\ \forall i \in C \setminus \{n\},\ \forall j \in C \setminus \{1\} \mid i \neq j\}$ the set of arcs in which the thief can travel. For any pair of cities i and j with $(i, j) \in A$, the distance d_{ij} between them is known. The thief can make a profit throughout their journey by stealing items and storing them in a knapsack with a limited capacity W. Moreover, the thief has a maximum time T to complete their whole robbery journey. As stolen items are put into the rented knapsack, its weight increases and the thief's velocity decreases inversely proportional to the knapsack weight. Specifically, when the knapsack is empty, the thief can move with their highest velocity v_{max}. However, when the knapsack is completely full, the thief moves with the minimum speed v_{min}. In general terms, the thief can move with a speed $v = v_{max} - w \cdot (v_{max} - v_{min}) \,/\, W$, where w is the current weight of their knapsack. The objective of the ThOP is to find a path for the thief that starts from city 1 and ends at city n, as well as a robbery plan, i.e., a set of items chosen from the cities visited that maximizes the total profit stolen, ensuring that the capacity of the knapsack W is not exceeded and that the total traveling time is within the given time limit T.

We can represent any solution for the ThOP through a pair $\langle \pi, z \rangle$, where $\pi = \langle 1, \dots, n \rangle$ is a list of visited cities by the thief, and $z = \langle z_1, z_2, \dots, z_m \rangle$ is a binary vector to represent the packing plan ($z_j = 1$ if item j is collected, and 0 otherwise) adopted by the thief throughout their robbery journey. Note that the first and last cities are fixed for any feasible solution. In addition, the number of cities visited may differ for different solutions.

It is important to note that nothing prevents the thief from visiting a city more than once. In this scenario, the thief should collect all items of a city at once on the last visit from that city to minimize their travel time and consequently has more time to collect other items. One more aspect that should be pointed out is that a solution that visits some cities without collecting any items would only be advantageous if the distances between cities do not respect the triangular inequality because it may be convenient that the thief visits a city just to shorten their route. Nevertheless, as Santos and Chagas [23] have defined the test problems for the ThOP – which we have also used in this work – in such a way that the distances between cities respect triangular inequality, it can be considered as an implicit optimization that any solution is formed by a list of cities π without repetition.

2.2 Mixed Integer Non-Linear Programming (MINLP) formulation

In order to formally describe the ThOP through a mathematical formulation, we have proposed an alternative Mixed Integer Non-Linear Programming (MINLP) formulation to that proposed by Santos and Chagas [23]. In contrast to Santos and Chagas' formulation, the following formulation uses a polynomial number of decision variables in terms of the number of cities and items. These decision variables are detailed bellow:

- x_{ij} : binary variable that gets 1 if the thief crosses arc $(i, j) \in A$, and 0 otherwise.
- y_i : binary variable that gets 1 if the thief visits city $i \in C$, and 0 otherwise.
- z_k : binary variable that gets 1 if the thief collects item $k \in I$, and 0 otherwise.
- q_i : variable that reports the weight of the knapsack after leaving city $i \in C$.
- t_i : variable that informs the thief's arrival time at city $i \in C$.

With these variables, we can describe the following MINLP formulation for the ThOP.

$$\max \sum_{k \in I} p_k \cdot z_k \tag{1}$$

$$\sum_{k \in I} w_k \cdot z_k \leq W \tag{2}$$

$$y_i \geq z_k \qquad i \in C, k \in I_i \tag{3}$$

$$y_i \leq \sum_{k \in I_i} z_k \qquad i \in C \setminus \{1, n\} \tag{4}$$

$$y_1 = y_n = 1 \tag{5}$$

$$\sum_{j:(i,j) \in A} x_{ij} = y_i \qquad i \in C \setminus \{n\} \tag{6}$$

$$\sum_{i:(i,j) \in A} x_{ij} = y_j \qquad j \in C \setminus \{1\} \tag{7}$$

$$q_j \geq \left(q_i + \sum_{k \in I_j} w_k \cdot z_k \right) \cdot x_{ij} \qquad (i,j) \in A \tag{8}$$

$$t_j \geq \left(t_i + \frac{d_{ij}}{v_{max} - \nu \cdot q_i} \right) \cdot x_{ij} \qquad (i,j) \in A \tag{9}$$

$$x_{ij} \in \{0,1\} \qquad (i,j) \in A \tag{10}$$

$$y_i \in \{0,1\} \qquad i \in C \tag{11}$$

$$z_k \in \{0,1\} \qquad k \in I \tag{12}$$

$$0 \leq q_i \leq W \qquad i \in C \tag{13}$$

$$0 \leq t_i \leq T \qquad i \in C \tag{14}$$

The objective (1) is to maximize the total profit of items collected. Constraint (2) ensures that the total weight of items collected does not exceed the knapsack capacity. While constraints (3) guarantee that the thief must visit a city to collect any item from it, constraints (4) ensure that the thief does not visit cities where no items are selected. Note that constraints (4) are not needed for the model to produce feasible solutions. However, these constraints strengthen the model by removing unprofitable route combinations. Constraint (5) simply imposes that cities 1 and n have to be visited once they are, respectively, the fixed start and end points of any feasible route. Constraints (6) and (7) guarantee route connectivity. Constraints (8) and (9) guarantee that the knapsack weight and the route time is properly increasing along the route according to the items, which also avoid subcycles. Note that constraints (8) and (9) are non-linear. Finally, constraints (10)-(14) define the scope and domain of the decision variables. Note that, as constraints (13) ensure that the knapsack weight must be always less than knapsack capacity W throughout the route, constraint (2) could be removed. However, constraints (13) may be weaker for the purpose expressed by constraint (2) due to the multiplication by the variable x_{ij}, which allows that weak fractional solutions to be considered during the resolution of the formulation.

It is worth mentioning that constraints (8) can be linearized by rewriting them using Big-M constants as shown in (15). On the other hand, we cannot linearize constraints (9) as they involve divisions and multiplications of decision variables. Nevertheless, we have also rewritten them using Big-M constants, as shown in (16), to remove their non-linear multiplications. In constraints (15) and (16), we have used different Big-M constants M'_j and M''_{ij}, which should assume any sufficiently large number that is greater than or equal to $W + \sum_{i \in I_j} w_i$ and $T + d_{ij}/v_{min}$, respectively.

$$q_j \geq q_i + \sum_{k \in I_j} w_k \cdot y_k - M'_j \cdot \big(1 - x_{ij}\big) \qquad (i,j) \in A \qquad (15)$$

$$t_j \geq t_i + \frac{d_{ij}}{v_{max} - \nu \cdot q_i} - M''_{ij} \cdot \big(1 - x_{ij}\big) \qquad (i,j) \in A \qquad (16)$$

Although the foregoing mathematical formulation may be used for solving the ThOP from a mathematical solver, we have not considered it in our experiments due to its complexity. As constraints (16) are still non-linear, they greatly increase the complexity of the formulation, making it impracticable to solve even the smallest-size instance defined in the literature for the ThOP [23]. In fact, our formulation cannot find even reasonable bounds for the problem due to its non-linearity. Therefore, we have bet in a heuristic strategy for helping the thief in their robbery, leaving an improved mathematical formulation and exact algorithms for future investigation. As our MINLP formulation might be used as a starting point for other investigations, we have made it publicly available at https://github.com/jonatasbcchagas/minlp_thop[1], which has been implemented using PySCIPOpt [18], a Python interface for the SCIP Optimization Suite [12].

3 Problem-solving methodology

Throughout this section, we describe our solution approach, called ACO++, for solving the ThOP. Our ACO++ is an improved version of the ACO algo-

[1] In addition to the code of our mathematical model, it also contains some experiments and results on smaller instances that we have created in order to solve the model and establish a benchmarking with exact results. However, the experiments have shown that even for instances with 15 cities and 1 item per city, our model cannot be solved, and it also is not able to find tight mathematical bounds within a reasonable computational time.

rithm previously presented in [5]. At the end of this section, we highlight the differences between both algorithms.

3.1 The overall algorithm

Our solution approach has been loosely based on Wagner's TTP approach [27], with Ant Colony Optimization (ACO) [8] as the central component.

Following [27], we have used the ACO for determining the thief's route, while another algorithm for determining their packing plan for each route found by the ants. We have used the MAX-MIN ant system [24], which limits all pheromones to an interval. In our implementation, we have used Stützle's ACOTSP 1.0.3 framework[2] for constructing the thief's route. The ACOTSP is an efficient framework that implements several ACO algorithms for the symmetric TSP. Most of the ACOTSP framework remains untouched in our approach, with only a few minimal modifications necessary to adapt it to the ThOP specifications. First, to construct the feasible routes for the thief from a given city to a given destination, and without the need to visit all cities, we have made an adaptation so that the ants built their routes until the thief's destination city, that is, city n has been visited. Thus, the ants are able to build routes of varying sizes.

Second, the pheromone trail updates are performed based on the quality of the TSP routes. Because the TSP's objective is to find the shortest possible route over all cities, the fitness of a given route is inversely proportional to its total distance. In contrast to this, in our ACOTSP adaptation, the fitness of each route is set based on the profit of the stolen items, because stolen items define the quality of ThOP solutions, which are defined by our proposed packing routine (to be described later): the fitness of a ThOP's route π is inversely proportional to $UB + 1 - p(z)$, where UB is an upper bound for the ThOP and $p(z)$ is the total profit of packing plan z. Thus, the fitness behaves similar to that of the TSP and we do not need to modify the ACOTSP any further. To set the upper bound UB, we use the optimal solution for the KP version that allows the selection of fractions of items, which can be solved in $\mathcal{O}(m \log_2 m)$ [25].

In Algorithm 1, we show the simplified overview of our ACO++. At the beginning (Line 1), the best ThOP solution (route and packing plan) found by the algorithm is initialized as an empty solution. The algorithm performs its iterative cycle (Lines 2 to 18) as long as the stopping criterion is not met. At Line 3, each ant constructs a route for the thief, and then packing plans are created (Line 4 and 5). The ACOTSP framework allows us to apply several classic local search heuristics: 2-*opt*, 2.5-*opt*, and 3-*opt* [1]. If any local search is enabled in our algorithm (Line 6), that local search procedure is performed on each route π, thus generating routes π' (Line 7), which may be better than π regarding the distance costs. In the next step, a packing plan z' is created

[2] Publicly available online at http://www.aco-metaheuristic.org/aco-code

from π' (Line 8). If z' is better than z (Line 9), π and z are replace by π' and z' (Line 10). At Lines 13 to 15, we update the best solution. Note that, to achieve more efficient routes, we remove from π all those cities from which no items are stolen (Line 14). As we have stated before, this is only true as all ThOP instances use distances that preserve the triangular inequality. After every route has been considered, the pheromones are updated according to the quality of the ThOP solutions (Line 17). In the end, the best solution found is returned.

Algorithm 1: ACO++ algorithm for the ThOP

```
1  $\pi^{best} \leftarrow \varnothing, z^{best} \leftarrow \varnothing$
2  repeat
3      $\Pi \leftarrow$ construct routes using ants
4      foreach route $\pi \in \Pi$ do
5          $z \leftarrow$ construct a packing plan from $\pi$
6          if local search procedure is activated then
7              $\pi' \leftarrow$ perform a local search procedure on route $\pi$
8              $z' \leftarrow$ construct a packing plan from $\pi'$
9              if profit of $z'$ is higher than profit of $z$ then
10                 $\pi \leftarrow \pi'$, $z \leftarrow z'$
11             end
12         end
13         if profit of $z$ is higher than profit of $z^{best}$ then
14             $\pi^{best} \leftarrow \zeta(\pi)$, $z^{best} \leftarrow z$
15         end
16     end
17     update ACO statistics and pheromone trail
18 until stopping condition is fulfilled
19 return $\pi^{best}$, $z^{best}$
```

$\zeta(\pi)$ removes from π all cities where no items are stolen according to the packing plan z.

3.2 Randomized packing heuristic

To construct a packing plan in our proposed ACO++ for a given route π, we use our previously developed, randomized heuristic for solving the ThOP [5]:[3] in brief, the randomization varies the relative influence of weights, profits, and distances in the (originally deterministic) heuristic PackIterative, which is an efficient packing algorithm developed for the TTP [10]. This non-deterministic strategy lets us explore the packing plan space even for a fixed

[3] As stated by Polyakovskiy and Neumann [22], determining the optimal packing plan is $\mathcal{NP}$-hard, even when the route of the thief is kept fixed.

route, which can lead to overall better configurations. This was also necessacitated by the observation that the ants would often find near-identical routes during an optimization run.

3.3 Differences between the ACO++ and ACO approaches

Compared to the ACO described in [5], we have incorporated two new features into our ACO++. These features are described in the following:

1. The ants of ACO++ construct routes that do not necessarily visit all cities, while in our previous ACO the ants always construct complete TSP tours, i.e., all cities are visited. Note that, although both algorithms remove from the route all cities in which no item is selected, there is now a higher consistency with respect to the ThOP's definition, because ants do not have to visit all cities. Note that this allows ants to construct routes that might not be easily constructed from our previous ACO. In addition, when a route visits fewer cities fewer items are available to be collected from that route, which reduces the search space to find a packing plan, thus making our packing routines more computationally efficient. It is also because of this last point, that we had to adapt our randomized packing heuristic algorithm to consider only the items that can be selected from each constructed route. However, its core idea remains unchanged.
2. There is now the possibility to apply different local searches on each route constructed by the ants in our ACO++. While this results in a TSP-bias toward shorter routes, finding shorter routes may help the thief to better plan their robbery journey.

4 Computational study

In the following, we present the experiments performed to study the performance of the proposed algorithm against other algorithms proposed for the ThOP [23, 9, 5]. As the computational budget of all ThOP algorithms are based on wallclock time, in order to enable a fair comparison, we have rerun all ThOP codes, except for Faêda and Santos [9]'s algorithm because we have not had access to their code. In our experiments, we have used a machine with Intel(R) Xeon(R) CPU X5650 @ 2.67GHz, running CentOS 7.4.

All raw results and solutions (tours and packing plans), as well as the code are publicly available at https://github.com/jonatasbcchagas/acoplusplus_thop.

4.1 Benchmarking instances

To evaluate the different ThOP approached, we use all 432 ThOP instances from [23]. These have been created based on the TTP instances [21] by removing the items in city n and by adding a maximum travel time. The instances have the following characteristics:

- numbers of cities: 51, 107, 280, and 1000 (TSP instances: *eil51*, *pr107*, *a280*, *dsj1000*);
- numbers of items per city: *01*, *03*, *05*, and *10*;
- sizes of knapsacks: *01*, *05* and *10* times the size of the smallest knapsack;
- types of knapsacks: values and weights of the items are either uncorrelated (*unc*), uncorrelated with similar weights (*usw*), or bounded and strongly correlated (*bsc*);
- maximum travel times: *01*, *02*, and *03* classes. These values refer to 50%, 75%, and 100% of instance-specific references times defined in the original ThOP paper [23].

In the remainder of this article, each instance will be identified as `XXX_YY_ZZZ_WW_TT`, where `XXX`, `YY`, `ZZZ`, `WW` and `TT` indicate the different characteristics of the instance at hand. For example, `pr107_05_bsc_01_01` identifies the instance with 107 cities (TSP instance *pr107*), 5 items per city with their weights and values bounded and strongly correlated with each other, and the smallest knapsack and time limit defined.

4.2 Parameter tuning

In the following, we tune a variety of ACOTSP parameters: *ants* defines the number of ants; *alpha* controls the relative importance of pheromone values in the construction of routes; *beta* defines the influence of distances between cities; *rho* is the evaporation rate of the pheromones; and *localsearch* controls whether and what local search procedure to apply. Moreover, we vary the number of attempts of our randomized packing heuristic *ptries*. To stop the algorithm, as in previous work on the ThOP, we limit the execution time to $\lceil 0.1m \rceil$ seconds, which is determined based on the number of items m of each particular instance.

To find well-performing configurations, we have followed the same tuning experiments used in [5], i.e., we have used the Irace package [17] for the automatic configuration of algorithms [2], in order to determine the influence of parameter values across different instance sets. We have divided all 432 instances into 48 groups and then executed Irace on each of them to achieve tuned configurations that (1) perform well and (2) that can be analyzed to learn about the problem domain. Each instance group is identified as `XXX_YY_ZZZ`: `XXX` denotes the TSP base group, `YY` the number of items per city and `ZZZ` the knapsack type. Each group `XXX_YY_ZZZ` contains all nine instances defined with different knapsack sizes and maximum travel time.

In Table 1 we show the parameters as well as their ranges; the ranges were determined in preliminary experiments. We have used Irace with its default settings, except for the parameter *maxExperiments*, which we have set to 5000.

Table 1: Parameter values considered during the tuning experiments.

Parameter	Investigated values
ants	$\{10, 20, 50, 100, 200, 500, 1000\}$
alpha	$\{0.00, 0.01, 0.02, \ldots, 10.00\}$
beta	$\{0.00, 0.01, 0.02, \ldots, 10.00\}$
rho	$\{0.00, 0.01, 0.02, \ldots, 1.00\}$
ptries	$\{1, 2, 3, 4, 5\}$
localsearch	{no local search, 2-*opt*, 2.5-*opt*, 3-*opt*}

In Figure 1, we show for each group of instances the configurations returned by Irace. Because Irace can return more than one configuration, we can sometimes see several configurations originating from the same instance (shown in the left-most columns). Each axis stands for a parameter and each parameter configuration is described by a line that intersects each parallel axis in its corresponding value. We can see which parameter values have been most selected among all tuning experiments by looking for "concentrations" of lines. We use different styles and colors to emphasize the results obtained for each individual group. All logfiles for these experiments can be found at the GitHub repository along with our code.

We can make several observations. First, we notice that the number of ants is typically between 50 and 200. The importance of the pheromone trail (α) is typically low, especially for the groups of instances that consider the TSP bases *eil51*, and *a280*. In turn, the importance of distances between cities (β) varies depending on the underlying TSP instance. This is to be expected, because the underlying TSP instances are different in nature and not normalized, hence requiring different values of beta. The evaporation rate of the pheromone trail has had a behavior more spread, although it seems to have a compensation correlation between the parameter *beta*: the higher the influence of distances between cities, the lower the evaporation rate of the pheromone trail. We can also observe that nearly all tuned configurations require the multiple invocation of our randomized packing heuristic, with the number of packing attempts widely spread among each other. Regarding the application of local searches on routes, one can note that for most groups of instances the use of 2-opt moves produces better ThOP solutions. However, some configurations do not include the use of any local search. Note that there are no configurations that indicate the use of 2.5-opt and 3-opt moves. Potentially, this is because high-quality TSP routes do not necessarily result in high-quality ThOP routes. Therefore, there may be no need to use local searches with larger neighborhood moves based solely on route distances as an improvement phase for ThOP routes.

4.3 Comparison of ThOP solution approaches

We compare the quality of the solutions obtained by ACO++ with the quality of the solutions obtained by other algorithms (ILS [23], BRKGA [23], GA [9],

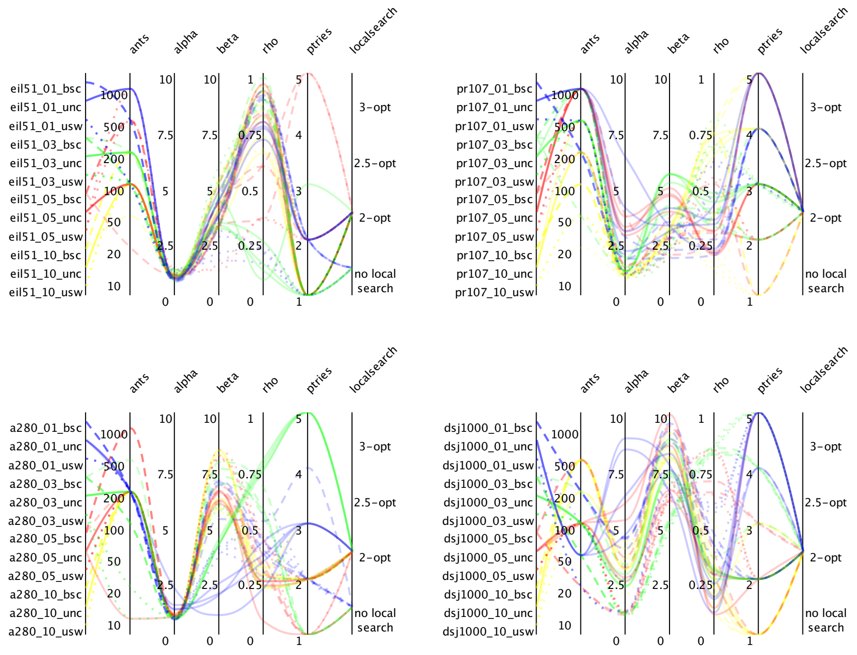
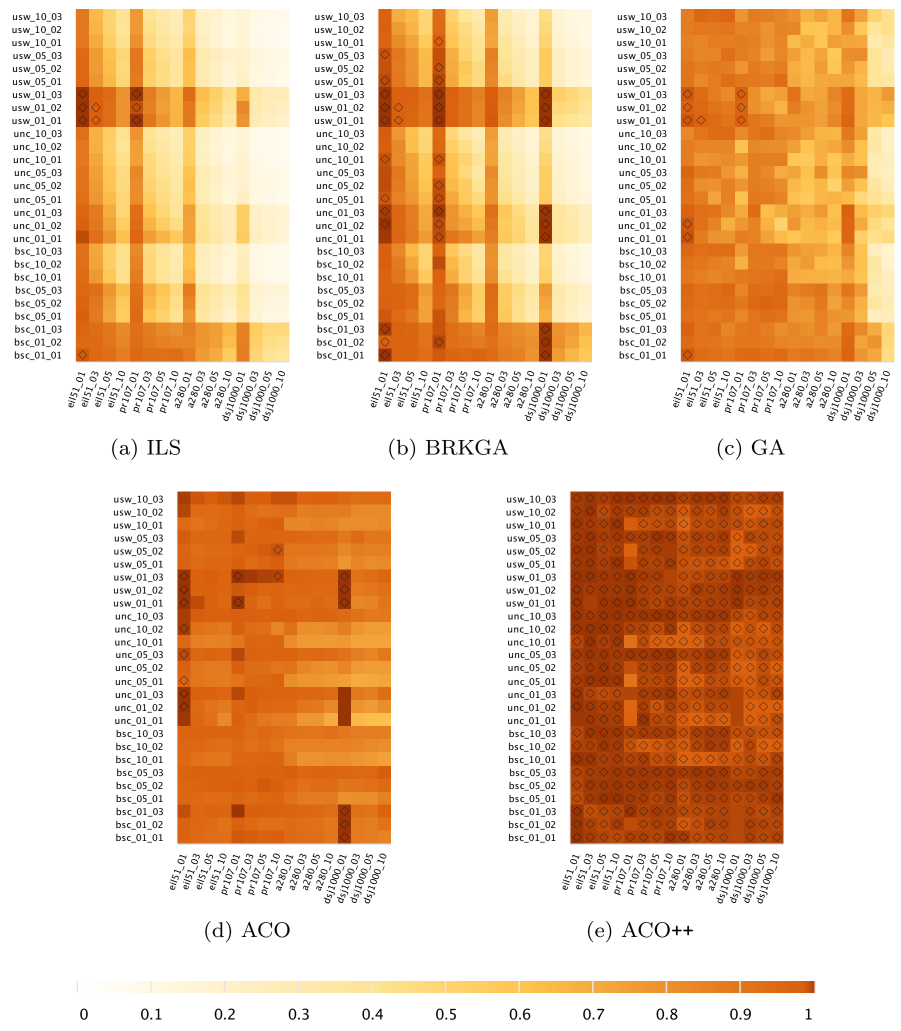


Fig. 1: Irace results for the 48 instance groups. Dashed, solid, and dotted lines are used, respectively, to emphasize the groups of instances with items where their weights and values are uncorrelated (*unc*), uncorrelated with similar weights (*usw*), and bounded and strongly correlated (*bsc*). Blue, green, red, and yellow lines represent, respectively, groups of instances with 1, 3, 5, and 10 items-per-city.

ACO [5]) already proposed for the ThOP. To enable a fair comparison, we also tuned the parameters of the BRKGA and ACO following the same process for ACO++, i.e., we have individually executed the Irace for each 48 different groups of instances. The ILS algorithm has no parameters to be tuned [23], while for the GA, we have not investigated its parameters because we have not had access to its code. Therefore, for the GA, we have made our analysis based on the results reported in [9].

Because all algorithms are randomized, we have performed 30 independent runs per instance. Each run has been executed with the parameter values with the best mean performance among those returned by Irace. ILS, BRKGA, and ACO codes, as well as their tuned configurations, raw results and solutions found, are also available at the GitHub link along with our ACO++.

In the first analysis, we compare the performance of the solutions obtained by measuring for each instance the achieved approximation ratio: for each instance and algorithm, we take the average objective value obtained considering the independent runs of that algorithm and compute the ratio between that average objective value and the best objective value found among all algorithms. Note that the higher the approximation ratio, the higher the average performance of that particular algorithm. In Figure 2, we plot for every instance and algorithm the approximation ratio as a heatmap in order to highlight larger differences. Moreover, we use diamond symbols to highlight the instances for which each algorithm has found the best known solutions.



(a) ILS (b) BRKGA (c) GA

(d) ACO (e) ACO++

Fig. 2: Approximation ratio of the solution approaches. Diamond symbols highlight in which the instances each algorithm has found the best solutions.

From Figure 2, we can make several observations. As stated by Santos and Chagas [23], we can also confirm that their BRKGA has outperformed their ILS for most instances, with higher prominence on the larger-size instances. In addition, one can note that their algorithms perform better for instances that involve only one item per city. Regarding the best-known solutions, we can see that their algorithms have not been able to find many of them. The GA proposed by Faêda and Santos [9] has outperformed, in general, both BRKGA and ILS solution approaches. Although BRKGA has found more best-known solutions, the GA has a more uniform behavior regarding the dimensions of the instances. Note that our previous ACO algorithm [5] has reached a better approximation ratio for almost all instances when compared to GA and also to ILS and BRKGA. In turn, our current ACO++ algorithm has presented a better or equal performance regarding the other algorithms for almost all instances. Similarly, it has typically found the best solutions for most of the instances.

In order to compare each pair of algorithms as to the best solutions found by them, we show in Table 2 the percentage of the number of instances in which every algorithm found better or equal quality solutions than another algorithm. The results shown in this table corroborate with those shown in Figure 2. In addition to showing that the ACO++ algorithm outperformed all other algorithms by more than 96% of the total of instances, we can also see that our previous ACO also is more efficient than ILS, BRKGA, and GA by over 88% of instances. In turn, GA is more efficient than ILS and BRKGA, and BRKGA overcomes ILS.

Table 2: Percentage of the number of instances in which algorithm i found better or equal quality solutions than algorithm j.

$i\downarrow \quad j\rightarrow$	ILS	BRKGA	GA	ACO	ACO++
ILS	-	3.01%	20.37%	4.63%	2.55%
BRKGA	99.54%	-	37.50%	14.58%	8.56%
GA	81.71%	64.58%	-	2.78%	2.31%
ACO	96.99%	88.89%	98.61%	-	5.32%
ACO++	99.54%	96.06%	99.54%	98.15%	-

As both algorithms based on ACO metaheuristics have had the best and most similar performances, we statistically compare the quality of their solutions using the Wilcoxon signed-rank test. At a significance level of 5%, the performance of ACO++ has been statistically worse than ACO in only 11 instances, in 12 instances there is no difference between the performance of both algorithms, while in 409 instances (about 95% of total) ACO++ has been better than ACO.

In Table 3, we summarize the results obtained with a closer analysis of the solutions found by ACO and ACO++. For each TSP base instance (`XXX`) and number of items per city (`YY`), which resulted in 27 instances each, we show averaged information concerning all the best solutions achieved by both approaches. Column $\mathcal{D}$ shows the ratio between the total distance traveled and the number of cities visited by the thief, while columns $\%T$ and $\%W$ report, respectively, the percentage spent of the time limit and the percentage used of the knapsack capacity. If values in these last two columns are close to 100%, then these indicate limiting factors. Note that both algorithms have a similar use of the time limit. On the other hand, the solutions found by ACO++ have used more the knapsack capacity, especially for instances with more cities and items. From the values in column $\mathcal{D}$, we can understand this behavior. Note that the ratio between the total distance traveled and the number of cities visited of the solutions found by ACO is higher than those found by ACO++. Note that the solutions found by ACO have a ratio between the total distance traveled and the number of cities visited higher than those solutions found by ACO++, which indicates that ACO has found the most spread-out routes and/or with more edge crossings. Therefore, as the routes found by ACO++ are more condensed and/or efficient, the thief is able to travel more effectively and, consequently, uses better the knapsack capacity, thus managing to collect a better set of items. To illustrate this behavior, Figure 3 shows for some instances, where the resulting quality differs significantly, the best solution found by each algorithm.

In summary, we can see that our ACO++ has been able to find significantly more efficient routes, which allows achieving better packing plans, and, consequently, achieving higher profits.

4.4 Solving the classical Orienteering Problem

One can note that the classical Orienteering Problem (OP) [13] is a subproblem of the ThOP, i.e., its definition is contained within the definition of the ThOP. Indeed, if (i) there is only one item per city ($|I_i| = 1$, $\forall i \in C \setminus \{1, n\}$), (ii) the thief travels at a constant speed ($v_{max} = v_{min}$), and (iii) their knapsack has unlimited capacity ($W = \infty$), we have the same constraints and objective on both problems. Therefore, we can use any ThOP algorithm for solving the classic OP, simply by *fixing* some information in the ThOP instances in order to convert them into OP instances.

To investigate the efficiency of BRKGA, ILS, ACO, and ACO++ algorithms proposed for the ThOP in solving the OP, we have adapted the ThOP instances that have only one item per city, setting their $v_{max} = v_{min} = 1$ and W to a large number so that the knapsack capacity is greater than the sum of all item profits. These instances will be identified as `XXX_YY_ZZZ_WW_TT`, as already mentioned in Section 4.1, with the emphasis now that `WW` = *inf*.

For our analysis, we have performed 30 independent runs per instance and algorithm. Then, for each instance, we have taken the average objective value obtained by each algorithm to compute the ratio between that value and the optimal objective value of that instance. We have determined the optimal value for each instance from the branch-and-cut algorithm proposed

Table 3: Information on the structure of the best solutions found. $\mathcal{D}$ is the ratio between the total distance traveled and the number of cities visited; $\%T$ and $\%W$ denote the percentage spent of the time limit and the percentage used of the knapsack capacity.

TSP base (`XXX`)	Number of items per city (`YY`)	ACO			ACO++		
		$\mathcal{D}$	%T	%W	$\mathcal{D}$	%T	%W
eil51	01	10.91	98.60	78.06	10.46	99.00	79.55
	03	9.46	99.60	83.85	8.63	99.55	85.34
	05	9.26	99.62	83.39	8.53	99.78	85.48
	10	9.12	99.84	85.26	8.20	99.88	86.57
pr107	01	718.92	99.66	79.58	680.85	99.74	81.82
	03	498.71	99.85	81.78	476.67	99.85	83.84
	05	471.77	99.93	83.22	445.23	99.95	83.79
	10	449.09	99.95	84.95	417.47	99.93	84.40
a280	01	16.52	99.61	79.57	14.23	99.80	83.94
	03	12.60	99.74	81.83	10.45	99.76	85.68
	05	11.84	99.95	82.72	9.61	99.93	86.27
	10	11.17	99.92	83.02	9.22	99.92	86.26
dsj1000	01	44632.08	74.72	79.31	37015.71	72.08	86.16
	03	26635.61	99.90	82.91	18943.46	99.89	89.57
	05	25648.23	99.85	82.43	18064.09	99.82	89.91
	10	23795.22	99.92	84.03	17700.59	99.68	90.35

by Fischetti et al. [11], which has been executed without time limitation. In Table 4, we show all computed ratios, where the highest ratio for each instance is highlighted in bold. Note that the higher the ratio, the higher the average performance of that particular ThOP algorithm in solving the OP instance.

From Table 4, we can verify the same behavior among the algorithms when solving OP instances as seen in the previous experiments: better efficiency of the algorithms based on the ACO strategy, with a significant better performance in our ACO++ algorithm. Note that our algorithm is able to find high-quality solutions for all instances. For almost all instances, the ratio between the objective values of solutions found by the ACO++ and their optimal values is larger than 0.95. On average, our algorithm has reached a ratio of 0.98. Note that for all instances with the TSP base *dsj1000*, both algorithms based on ACO has found the optimal solutions (ratio = 1.00). Analyzing such solutions, we see that the thief is able to visit all cities and, consequently, steal all items. This has occurred because without the speed reduction and with an unlimited knapsack, these instances become easier once the thief has enough time and space in their knapsack to steal and carry all items along a path that does not need to be very optimized. Anyway, it is noteworthy that the other algorithms have not been able to find such solutions.

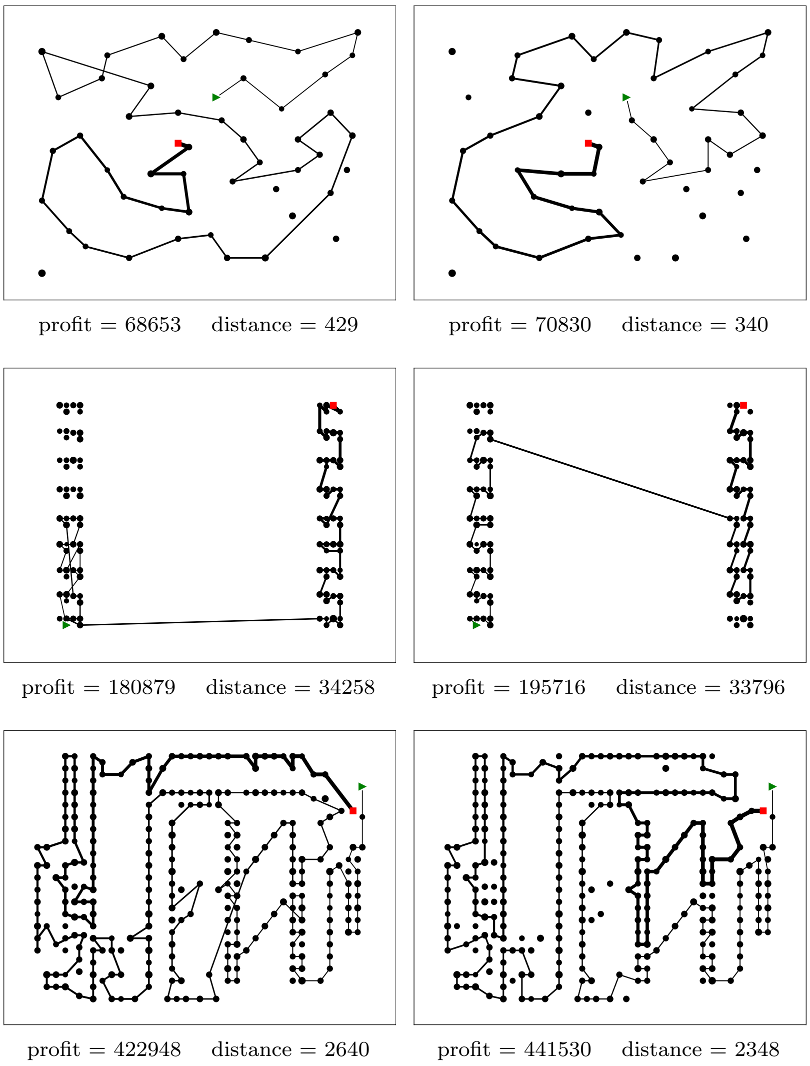

profit = 68653 distance = 429
profit = 70830 distance = 340
profit = 180879 distance = 34258
profit = 195716 distance = 33796
profit = 422948 distance = 2640
profit = 441530 distance = 2348

Fig. 3: The graphical representation of the solutions found by ACO (left) and ACO++ (right) for instances *eil51_10_bsc_01_03* (top), *pr107_05_usw_10_02* (middle), and *a280_10_unc_01_03* (bottom). The initial and final cities are highlighted using a green triangle and a red square, respectively. The lines connecting pairs of cities represent the route traveled by the thief, with the increasing line thickness corresponding to the thief's knapsack weight.

Table 4: Ratios between the average objective value obtained by each algorithm and the optimal objective value for OP instances.

Instance	ILS	BRKGA	ACO	ACO++
`eil51_01_bsc_inf_01`	0.922	0.953	0.921	**0.965**
`eil51_01_bsc_inf_02`	0.912	0.961	0.977	**0.995**
`eil51_01_bsc_inf_03`	0.879	0.964	0.968	**0.988**
`eil51_01_unc_inf_01`	0.872	0.963	0.965	**0.997**
`eil51_01_unc_inf_02`	0.856	0.956	0.989	**0.996**
`eil51_01_unc_inf_03`	0.905	0.979	0.999	**1.000**
`eil51_01_usw_inf_01`	**1.000**	0.989	**1.000**	**1.000**
`eil51_01_usw_inf_02`	0.979	0.996	0.969	**1.000**
`eil51_01_usw_inf_03`	0.917	0.958	0.966	**1.000**
`pr107_01_bsc_inf_01`	0.721	0.852	0.923	**0.940**
`pr107_01_bsc_inf_02`	0.709	0.892	0.967	**0.999**
`pr107_01_bsc_inf_03`	0.791	0.969	**1.000**	**1.000**
`pr107_01_unc_inf_01`	0.689	0.954	0.935	**0.955**
`pr107_01_unc_inf_02`	0.668	0.904	0.924	**0.961**
`pr107_01_unc_inf_03`	0.718	0.915	0.999	**1.000**
`pr107_01_usw_inf_01`	0.861	**0.985**	0.945	0.965
`pr107_01_usw_inf_02`	0.732	0.943	0.961	**0.979**
`pr107_01_usw_inf_03`	0.708	0.908	0.940	**0.971**
`a280_01_bsc_inf_01`	0.537	0.683	0.781	**0.882**
`a280_01_bsc_inf_02`	0.516	0.655	0.869	**0.954**
`a280_01_bsc_inf_03`	0.533	0.675	0.981	**0.997**
`a280_01_unc_inf_01`	0.484	0.639	0.774	**0.927**
`a280_01_unc_inf_02`	0.476	0.642	0.932	**0.973**
`a280_01_unc_inf_03`	0.522	0.709	**1.000**	**1.000**
`a280_01_usw_inf_01`	0.637	0.750	0.833	**0.956**
`a280_01_usw_inf_02`	0.500	0.657	0.814	**0.932**
`a280_01_usw_inf_03`	0.476	0.627	0.828	**0.949**
`dsj1000_01_bsc_inf_01`	0.669	0.882	**1.000**	**1.000**
`dsj1000_01_bsc_inf_02`	0.836	0.952	**1.000**	**1.000**
`dsj1000_01_bsc_inf_03`	0.965	0.995	**1.000**	**1.000**
`dsj1000_01_unc_inf_01`	0.589	0.904	**1.000**	**1.000**
`dsj1000_01_unc_inf_02`	0.759	0.961	**1.000**	**1.000**
`dsj1000_01_unc_inf_03`	0.956	0.995	**1.000**	**1.000**
`dsj1000_01_usw_inf_01`	0.272	0.656	**1.000**	**1.000**
`dsj1000_01_usw_inf_02`	0.363	0.770	**1.000**	**1.000**
`dsj1000_01_usw_inf_03`	0.438	0.882	**1.000**	**1.000**
Average	0.705	0.863	0.949	**0.980**

5 Conclusions

In this article, we have proposed an improvement to a swarm-based approach to the academic Thief Orienteering Problem (ThOP): we have combined a heuristic approach based on Ant Colony Optimization with a randomized packing heuristic and with local searches. Using extensive tuning on groups of instances, we have studied the effects of our algorithmic components. Furthermore, we have evaluated the performance of the algorithm on the complete set of instances available in the literature. The experiments show that our solution strategy is able to find better solutions with large improvements when compared to the other solution methods proposed for the problem. Based on our analysis, the efficiency of our algorithm is due to the fact that ants have been able to find more efficient routes, which has allowed our packing heuristic to select a better set of items. In addition, we have shown that our algorithm is able to find high-quality solutions for the classic Orienteering Problem without any modification in its design.

For future research, we may investigate a variant of the current problem that generalizes to multiple thieves. Another interesting direction will be to approach the problem in a bi-objective version, where are to maximize the total profit and minimize the total distance traveled. By combining both foregoing directions, it will create a very interesting, challenging, and general problem with potential applications in real-world scenarios with routing problems under time-dependent limitations.

Acknowledgements The authors would like to thank the following institutions and funding bodies: Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance code 001; Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG); Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq); Universidade Federal de Ouro Preto (UFOP); Universidade Federal de Viçosa (UFV); and Australian Research Council Project DP200102364.

References

1. Aarts, E., Aarts, E.H., Lenstra, J.K.: Local search in combinatorial optimization. Princeton University Press (2003)
2. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-race and iterated f-race: An overview. In: Experimental methods for the analysis of optimization algorithms, pp. 311–336. Springer (2010)
3. Bonyadi, M.R., Michalewicz, Z., Barone, L.: The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In: IEEE Congress on Evolutionary Computation, pp. 1037–1044. IEEE (2013)
4. Bonyadi, M.R., Michalewicz, Z., Wagner, M., Neumann, F.: Evolutionary computation for multicomponent problems: opportunities and future directions. In: Optimization in Industry, pp. 13–30. Springer (2019)
5. Chagas, J.B., Wagner, M.: Ants can orienteer a thief in their robbery. Operations Research Letters **48**(6), 708 – 714 (2020)
6. Chand, S., Wagner, M.: Fast heuristics for the multiple traveling thieves problem. In: Genetic and Evolutionary Computation Conference (GECCO), pp. 293–300. ACM (2016)
7. Chen, C., Cheng, S.F., Gunawan, A., Misra, A., Dasgupta, K., Chander, D.: Traccs: A framework for trajectory-aware coordinated urban crowd-sourcing. In: J.P.

Bigham, D.C. Parkes (eds.) Second AAAI Conference on Human Computation and Crowdsourcing (HCOMP). AAAI (2014). URL `http://www.aaai.org/Library/HCOMP/hcomp14contents.php`
8. Dorigo, M., Di Caro, G.: Ant colony optimization: a new meta-heuristic. In: IEEE Congress on Evolutionary Computation (CEC), vol. 2, pp. 1470–1477. IEEE (1999)
9. Faêda, L.M., Santos, A.G.: A genetic algorithm for the thief orienteering problem. In: 2020 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8. IEEE (2020)
10. Faulkner, H., Polyakovskiy, S., Schultz, T., Wagner, M.: Approximate approaches to the traveling thief problem. In: Genetic and Evolutionary Computation Conference (GECCO), pp. 385–392. ACM (2015)
11. Fischetti, M., Gonzalez, J.J.S., Toth, P.: Solving the orienteering problem through branch-and-cut. INFORMS Journal on Computing **10**(2), 133–148 (1998)
12. Gamrath, G., Anderson, D., Bestuzheva, K., Chen, W.K., Eifler, L., Gasse, M., Gemander, P., Gleixner, A., Gottwald, L., Halbig, K., Hendel, G., Hojny, C., Koch, T., Le Bodic, P., Maher, S.J., Matter, F., Miltenberger, M., Mühmer, E., Müller, B., Pfetsch, M.E., Schlösser, F., Serrano, F., Shinano, Y., Tawfik, C., Vigerske, S., Wegscheider, F., Weninger, D., Witzig, J.: The SCIP Optimization Suite 7.0. ZIB-Report 20-10, Zuse Institute Berlin (2020). URL `http://nbn-resolving.de/urn:nbn:de:0297-zib-78023`
13. Golden, B.L., Levy, L., Vohra, R.: The orienteering problem. Naval Research Logistics **34**, 307–318 (1987)
14. Gunawan, A., Lau, H.C., Vansteenwegen, P.: Orienteering problem: A survey of recent variants, solution approaches and applications. European Journal of Operational Research **255**(2), 315 – 332 (2016)
15. Iori, M., Martello, S.: Routing problems with loading constraints. Top **18**(1), 4–27 (2010)
16. Kim, H., Kim, B.I., jin Noh, D.: The multi-profit orienteering problem. Computers and Industrial Engineering **149**, 106808 (2020)
17. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
18. Maher, S., Miltenberger, M., Pedroso, J.P., Rehfeldt, D., Schwarz, R., Serrano, F.: PySCIPOpt: Mathematical programming in python with the SCIP optimization suite. In: Mathematical Software – ICMS 2016, pp. 301–307. Springer International Publishing (2016). DOI 10.1007/978-3-319-42432-3_37
19. Neumann, F., Polyakovskiy, S., Skutella, M., Stougie, L., Wu, J.: A fully polynomial time approximation scheme for packing while traveling. In: Y. Disser, V.S. Verykios (eds.) Algorithmic Aspects of Cloud Computing, pp. 59–72. Springer (2019)
20. Orlis, C., Bianchessi, N., Roberti, R., Dullaert, W.: The team orienteering problem with overlaps: An application in cash logistics. Transportation Science **54**(2), 470–487 (2020)
21. Polyakovskiy, S., Bonyadi, M.R., Wagner, M., Michalewicz, Z., Neumann, F.: A comprehensive benchmark set and heuristics for the traveling thief problem. In: Genetic and Evolutionary Computation Conference (GECCO), pp. 477–484. ACM (2014)
22. Polyakovskiy, S., Neumann, F.: Packing while traveling: Mixed integer programming for a class of nonlinear knapsack problems. In: International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems (CPAIOR), pp. 332–346. Springer (2015)
23. Santos, A.G., Chagas, J.B.: The thief orienteering problem: Formulation and heuristic approaches. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1191–1199. IEEE (2018)
24. Stützle, T., Hoos, H.H.: Max–min ant system. Future generation computer systems **16**(8), 889–914 (2000)
25. Toth, P., Martello, S.: Knapsack problems: Algorithms and computer implementations. Wiley (1990)
26. Trachanatzi, D., Rigakis, M., Marinaki, M., Marinakis, Y.: A firefly algorithm for the environmental prize-collecting vehicle routing problem. Swarm and Evolutionary Computation p. 100712 (2020)
27. Wagner, M.: Stealing items more efficiently with ants: a swarm intelligence approach to the travelling thief problem. In: International Conference on Swarm Intelligence (ANTS), pp. 273–281. Springer (2016)

28. Wagner, M., Lindauer, M., Mısır, M., Nallaperuma, S., Hutter, F.: A case study of algorithm selection for the traveling thief problem. Journal of Heuristics **24**(3), 295–320 (2018)
29. Wu, J., Wagner, M., Polyakovskiy, S., Neumann, F.: Exact approaches for the travelling thief problem. In: Asia-Pacific Conference on Simulated Evolution and Learning, pp. 110–121. Springer (2017)