Ants can orienteer a thief in their robbery

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

[1] Dep. de Computação, Universidade Federal de Ouro Preto, Ouro Preto, Brazil
[2] Dep. de Informática, Universidade Federal de Viçosa, Viçosa, Brazil
jonatas.chagas@{iceb.ufop.br, ufv.br}
[3] School of Computer Science, The University of Adelaide, Adelaide, Australia
markus.wagner@adelaide.edu.au

Abstract. We address the Thief Orienteering Problem (ThOP), a multi-component problem that combines features of two classic combinatorial optimization problems, namely the Orienteering Problem and Knapsack Problem. Due to the given time constraint and the interaction of the load-dependent movement speed with the chosen route, the ThOP is complex and challenging. We propose a two-phase, swarm-intelligence based approach together with a new randomized packing heuristic. To identify the impact of the respective components, we use automated algorithm configuration. The resulting configurations outperform existing work on more than 90% of the benchmarking instances, with an average improvement of over 300%.

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

1 Introduction

Optimization problems are frequently investigated due to their theoretical and practical relevance. Many studies devoted to solving classical combinatorial optimization problems can be directly applied to solving real-world problems, such as vehicle routing problems, scheduling problems, and packing problems. While this is true, many other real-world problems exhibit multi-component structures, i.e., they consist of several combinatorial optimization problems that interact with each other. Multi-component problems are hard to solve as each of their components influences the quality and feasibility of the other components [6].

Among the real-world multi-component problems, vehicle routing problems with loading constraints [18] subjectively appear to be very prominent. There, routes need to be planned for vehicles while constraints and aims of specific loading policies. A well-studied (but academic) multi-component problem is the Traveling Thief Problem (TTP), which combines two classic problems: the Traveling Salesman Problem (TSP) and the Knapsack Problem (KP). Bonyadi et al. [5] have proposed the TTP to provide an academic abstraction of multi-component problems for the scientific community. In brief, in the TTP, a single thief has to visit all cities (TSP component) and can make a profit by stealing items and

storing them in a rented knapsack (KP component). As stolen items are stored in the knapsack, it becomes heavier, and the thief travels more slowly, with a velocity inversely proportional to the knapsack weight. The thief's objective is to maximize the total profit of the stolen items while considering the price to pay for the knapsack, which is proportional to the rent time.

The Thief Orienteering Problem (ThOP) [25] has been designed as an academic multi-component problem with different interactions and constraints in mind: it combines the Orienteering Problem (OP) and the Knapsack Problem (KP). The OP is a well-studied problem in operational research (see, e.g., [15,27,17]), where a participant starts on a given point, travels through a region visiting checkpoints, and has to arrive at a control point within a given time. Each checkpoint has a score, and the objective of the participant is to find the route that maximizes the total score, i.e., whose sum of scores of the checkpoints visited is maximal. Recent real-world examples of the OP include tourists planning their sight-seeing trips [11], rescue teams planning the visit of safe places in case of emergencies [3], and politicians or music bands planning their tours [2,13].

For the ThOP, Santos and Chagas [25] have proposed a Mixed Integer Non-Linear Programming formulation for it, but no computational results have been presented due to the formulation's complexity. Instead, two simple heuristic algorithms have been proposed, i.e., one based on Iterated Local Search (ILS) [21] and one based on a Biased Random-Key Genetic Algorithm (BRKGA) [16]. The BRKGA outperformed ILS on large instances, and the authors have attributed this to the diversification introduced of the mutant individuals.

In this work, we propose the use of a two-phase swarm intelligence approach based on Ant Colony Optimization (ACO) and a new greedy heuristic, to construct, respectively, the route and the packing plan (stolen items) of the thief. We investigate the importance of the components via automated algorithm configuration and then evaluate our approach on a broad set of instances.

The remainder of this paper is structured as follows. In Section 2, we formally describe the ThOP. In Section 3, we present our solution approach proposed for addressing the ThOP. Section 4 reports the experiments and analyzes the performance of the proposed solution approach against previous ones already proposed in the literature. We conclude in Section 5 with a summary and outline possible future work.

2 Problem description

As stated by Santos and Chagas [25], in the ThOP, there is a set of n cities, labeled from 1 to n, where the cities 1 and n are, respectively, the cities where the thief starts and ends their journey. A set of m items scattered among the other cities $(2, \ldots, n-1)$, and each city has one or more items. Each item $i \in \{1, \ldots, m\}$ has a profit p_i and weight w_i associated. For any pair of cities i and j, the distance d_{ij} between them is known.

In the ThOP, there is a single thief to steal items scattered among cities. The thief has a knapsack with a limited capacity W to carry the items. Moreover, the thief has a maximum time T to complete their whole robbery plan. The speed of the thief is inversely proportional to their knapsack weight. When the knapsack is empty, the thief can move with their maximum speed v_{max}. However, when the knapsack is full, the thief moves with the minimum speed v_{min}. The speed v of the thief when the knapsack weight is $w \leq W$ is given by $v = v_{max} - w \times (v_{max} - v_{min}) / W$.

The objective of the ThOP is to provide a path from the start city 1 to the end city n, as well as a set of items chosen from the cities visited throughout the route so that to maximize the total profit stolen, ensuring that the capacity of the knapsack W is not surpassed and the total traveling time of the thief is within the time limit T. The thief does not need to visit all cities.

Note that, while the ThOP and the TTP appear to be similar due to the KP as a component, it has been argued that the ThOP is more practical due to two key differences: in the ThOP there is (A) no need to visit all the cities, and (B) the interaction is not through a time-dependent rent for the knapsack, but through a constraint that imposes on the thief a time limit to complete the tour – at the very least for the aforementioned real-world examples, speed typically remains constant, but time constraints have to be fulfilled, and only worthwhile places have to be visited. While the relaxation of difference (A) might appear trivial, the consideration of this constraint, i.e., to visit all cities, is typically reflected in the design of heuristic [29] and exact [30,22] approaches to the TTP, with Chand and Wagner [7]'s Multiple Traveling Thieves Problem (MTTP) being the only exception known to us. Regarding the difference (B) and the ThOP in general, applications of it can arise when there is no enough time and capacity to visit all possible cities. For an overview of time-dependent routing problems, we refer the interested reader to Gendreau et al. [14].

3 Stealing items with ants

In the following, we describe our heuristic approach for the ThOP. It is loosely based on Wagner's TTP study [28]. As in [28], we propose in this work the use of swarm intelligence based on Ant Colony Optimization (ACO) [10] in order to solve ThOP's tour part, while a novel heuristic will be responsible for solving the ThOP's packing part, i.e., to select the set of stolen items.

ACO algorithms consist of an essential class of probabilistic search techniques that are inspired by the behavior of real ants. These algorithms have proven to be efficient in solving a range of combinatorial problems [9]. The basic idea behind ACOs is that ants construct solutions for a given problem by carrying out walks on a so-called construction graph. These walks are influenced by the pheromone values that are stored along the edges of the graph. During the optimization process, the pheromone values are updated according to good solutions found during the optimization, which should then lead the ants to better solutions in

further iterations of the algorithm. We refer the interested reader to the book by Dorigo and Birattari [8] for a comprehensive introduction.

In order to define the thief's route, we use Stützle's ACOTSP 1.0.3 framework[4]. This framework implements several ACO algorithms for the symmetric TSP, i.e., its found solutions are tours that visit all cities. While this may not be efficient or even feasible for the thief due to the time limit to conclude their journey, we will adapt the output according to the solution found by the packing algorithm in order to determine efficient solutions for the ThOP.

We note that ACOTSP builds complete TSP tours, not OP tours, hence possibly affecting the algorithm performance. We decided against the OP-tour approach: assuming that we have an OP tour and then consider the packing, we may (based on our two-phase approach) end up skipping cities anyhow if there are no interesting items to pick up; hence, a further dropping of cities may be required anyhow. Of course, this would be different if we would have an algorithm to solve the OP and KP parts simultaneously.

3.1 ACO framework and adjustments

The ACOTSP framework allows us to choose which ant colony optimization approach to be used. As in [28], we use the MAX-MIN ant system by Stützle and Hoos [26], which restricts all pheromones to a bounded interval in order to prevent pheromones from dropping to arbitrarily small values. In Algorithm 1, we show the simplified overview of the proposed swarm intelligence approach, combined with the packing heuristic algorithm.

Algorithm 1: ACO for the ThOP

```
1  π^best ← ∅, z^best ← ∅
2  repeat
3  |  Π ← construct TSP tours using ants
4  |  foreach TSP tour π ∈ Π do
5  |  |  z ← construct a packing plan from π using PACK(π, ptries)
6  |  |  if local search procedure is activated then
7  |  |  |  π' ← perform local search on tour π
8  |  |  |  z' ← construct a packing plan from π' using PACK(π', ptries)
9  |  |  |  if profit of z' is higher than profit of z then
10 |  |  |  |  π ← π', z ← z'
11 |  |  if profit of z is higher than profit of z^best then
12 |  |  |  π^best ← ζ(π), z^best ← z
13 |  update ACO statistics and pheromone trail
14 until stopping condition is fulfilled
15 return π^best, z^best
```

$\zeta(\pi)$ returns a tour for the thief from the TSP tour π by removing all cities where no item is stolen according to the packing plan z.

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

Initially (line 1), the best ThOP solution (tour and packing plan) found by the algorithm is initialized as an empty solution. The algorithm performs its iterative cycle (lines 2 to 14) as long as the stopping criterion is not fulfilled. At line 3, each ant constructs a TSP tour. For each TSP tour π (line 4), we apply our heuristic algorithm for defining a packing plan (line 5, Algorithm 2). The ACO framework allows us to apply on the tours found by the ants some classic local search procedures: 2-*opt*, 2.5-*opt*, and 3-*opt* [1]. If any local search is enabled (line 6) in our algorithm, we perform that local search procedure on each tour π, thus generating a new TSP tour π' (line 7). From π' we construct a packing plan z' using our packing algorithm (line 8). If z' is better than z (line 9), we replace π by π' and z by z' (line 10). At lines 11 to 12, the best solution found is possibly updated. Note that we remove from π all cities where no items have been stolen according to the packing plan z (line 12) in order to get a more efficient ThOP's tour (all ThOP instances use Euclidean distances rounded up). After all tours have been considered, ACO statistics and the pheromone values are updated according to the quality of the ThOP solutions found (line 13). At the end of the algorithm (line 17), the best solution found is returned.

Implementation Notes. The overall logic of the ACOTSP framework remains unchanged in our proposed algorithm. Some minimal modifications have been performed to adapt it to the ThOP specifications. To construct the TSP tours, we just established that the first and last cities must be those where the thief begins and ends their robbery journey. In the ACOTSP framework, the pheromone trail update performs based on the quality of the TSP tours found by ants. Since the objective of the TSP is to find the shortest possible tour visiting each city, the fitness of a given tour is inversely proportional to its total distance. On the other hand, in our ACOTSP adaptation, the fitness of each tour is set in terms of the quality of the stolen items throughout the tour, which are defined by the heuristic packing plan. As the ACOTSP framework is developed explicitly for the TSP, a minimization problem where its solutions have positive objective values, we consider that the fitness of a ThOP's tour 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. Note that in this way we can maintain the same behavior of fitness of the TSP solutions, without modifying the ACO framework structure. The upper bound UB is defined as the optimal solution for the KP version that allows selecting fractions of items. This KP version can be solved in $O(m \log_2 m)$.

3.2 ThOP packing heuristic

In Algorithm 2, we describe our heuristic strategy for constructing a packing plan from a fixed tour. Note that even when the tour of the thief is kept fixed, finding the optimal packing configuration is NP-hard [24].

Our packing heuristic algorithm seeks to find a good packing plan z from multiple attempts for the same tour π. The number of attempts is defined by *ptries*. Each attempt is described between lines 2 to 16. At the beginning of

Algorithm 2: Packing Algorithm: PACK(π, *ptries*)

```
1  z ← ∅, try ← 1
2  repeat
3  |  choose a real number for each parameter θ, δ, and γ from a uniform
   |   distribution in the range [0, 1], so that θ + δ + γ = 1
4  |  foreach i ← 1 to m do
5  |  |  compute score s_i for item i                                  // Eq. 1
6  |  z' ← ∅
7  |  for j ← 1 to m do
8  |  |  i ← get item with the j-th highest score
9  |  |  z' ← z' ∪ {i}
10 |  |  if weight of z' is higher than W then  z' ← z' \ {i}
11 |  |  else
12 |  |  |  t ← compute the required time to steal z' by visiting only cities
   |  |  |   with selected items following the order of the TSP tour π
13 |  |  |  if t is longer than T then  z' ← z' \ {i}
14 |  if profit of z' is higher than profit of z then  z ← z'
15 |  try ← try + 1
16 until try > ptries
17 return z
```

each attempt (line 3), we uniformly select three random values (θ, δ, and γ) between 0 and 1, and then normalize them so that their sum is equal to 1. These values are used to compute a score s_i for each item $i \in \{1, \ldots, m\}$ (lines 4 to 5), where θ, δ, and γ define, respectively, exponents applied to profit p_i, weight w_i, and distance d_i in order to manage their impact. The distance d_i is calculated according to the tour π by sum all distances from the city where is the item i to the end city. Equation 1 shows as the score of item i is calculated.

$$s_i = \frac{{p_i}^{\theta}}{{w_i}^{\delta} \times {d_i}^{\gamma}} \tag{1}$$

Note that each score s_i incorporates a trade-off between a distance that item i has to be carried over, its weight, and its profit. Equation 1 is based on the heuristic PACKITERATIVE that has been developed for the TTP [12]. However, unlike in [12], we consider an exponent for the term of distance to vary the importance of its influence. Furthermore, the values of all exponents are randomly selected drawn between 0 and 1 for each attempt (and then normalized) to search the space for greedy packing plans.

After computing scores for all items, we use their values to define the priority of each item in the packing strategy. The higher the score of an item, the higher its priority. Between lines 7 and 13, we create the packing plan for the current attempt by considering the items according to their priorities. If an item violates the constraints of the ThOP (lines 10 and 13), it is not selected. Note that we calculate travel time (line 12) from the cities listed on tour π, but we ignore those cities where no items are selected. After completing the current attempt's

packing plan, its quality is compared to the best packing plan so far (line 14), which is then possibly updated (line 14). At the end of all attempts, the best packing plan found is returned (line 17).

Note that our packing algorithm is non-deterministic (in contrast to the deterministic PackIterative [12]'s), as it has randomized components. In our preliminary experiments, we have observed that ants find identical or very similar routes throughout the iterations of the ACO algorithm. For this reason, we decided to design our packing algorithm in a non-deterministic way in order to increase the explore the packing plan space more broadly. Moreover, via the parameter *ptries*, we can control the number of attempts needed to reached good packing plans.

4 Computational experiments

We now present the experiments performed to study the performance of the proposed framework concerning the quality of its solutions. We have rerun Santos and Chagas [25]'s ThOP code to enable a fair comparison as the computational budget is based on wallclock time.

Our framework has been implemented based on Thomas Stützle's ACOTSP 1.0.3 framework, which has been implemented in C programming language. In our experiments, each run of the proposed algorithm has been sequentially (non-parallel) performed on an Intel(R) Xeon(R) E5-2660 (2.20GHz), running under CentOS Linux 7 (Core). Our code, as well as all results and solutions, can be found at https://github.com/jonatasbcchagas/aco_thop.

4.1 Benchmarking instances

To assess the quality of the proposed algorithm, we have used all ThOP instances defined by Santos and Chagas [25]. As stated by the authors, these instances have been created upon a benchmark of TTP instances [23] by removing the items on city n and by adding a maximum travel time. They have created 432 instances with the following characteristics:

- numbers of cities: 51, 107, 280, and 1000 (TSP instances (`XXX`): *eil51*, *pr107*, *a280*, *dsj1000*);
- numbers of items per city (`YY`): *01*, *03*, *05*, and *10*;
- types of knapsacks (`ZZZ`): weights and values of the items are bounded and strongly correlated (*bsc*), uncorrelated (*unc*), or uncorrelated with similar weights (*usw*);
- sizes of knapsacks (`WW`): *01*, *05* and *10* times the size of the smallest knapsack;
- maximum travel times (`TT`): *01*, *02*, and *03* classes. These values refer to 50%, 75%, and 100% of instance-specific references times defined in the original ThOP paper [25].

All 432 ThOP instances can be obtained by combining the different characteristics described above. Each instance is identified as `XXX_YY_ZZZ_WW_TT.thop`.

4.2 Parameter tuning to gain insights

Our first study analyzes the influence of the values of the main parameters of our algorithm. As in the previous work [25] on the ThOP, we have 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 particular instance.

The ACOTSP framework allows setting a large number of parameters. We consider the following: *ants* defines the number of ants used; *alpha* controls the relative importance of pheromone trails in the construction of tours; *beta* defines the influence of distances between cities for construction the tours; *rho* sets the evaporation rate of the pheromone trail; and *localsearch* controls whether and which local search procedure is applied to tours. Besides, we analyze the influence of our parameter *ptries*, which is used for deciding how many attempts our packing algorithm performs to determine the set of stolen items.

Table 1 shows the parameter values we have considered in our analysis. The ranges have been selected following preliminary experiments.

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 order to find a suitable configuration of parameters among all possible ones, we use the Irace package [20], which is an implementation of the method I/F-Race [4]. The Irace package implements an iterated racing framework for the automatic configuration of algorithms. In our experiments, we have used all Irace default settings, except for the parameter *maxExperiments*, which has been set to 5000. This parameter defines the stopping criteria of the tuning process. We refer the readers to [19] for a complete user guide of Irace package.

To analyze the influence of parameter values across the different types of instances, we divide all 432 instances into 48 groups and then execute Irace on each of them. Each group is identified as `XXX_YY_ZZZ`, where `XXX` informs the TSP base group, `YY` the number of items per city and `ZZZ` the type of knapsack. Each group `XXX_YY_ZZZ` contains all nine instances defined with different sizes of knapsacks and maximum travel time.

In Figure 1, we plot for each group all configurations returned by Irace at the end of its run. All logs generated by the Irace executions, as well as their settings can be found on the GitHub link along with our code. Each parallel coordinate plot lists for each of the 48 groups (shown in the left-most column) the configurations returned by Irace (shown in the other columns). As Irace can return more than one configuration, multiple configurations are sometimes shown. Each axis indicates a parameter and its range of values, and each configuration of parameters is described by a line that cuts each parallel axis in its

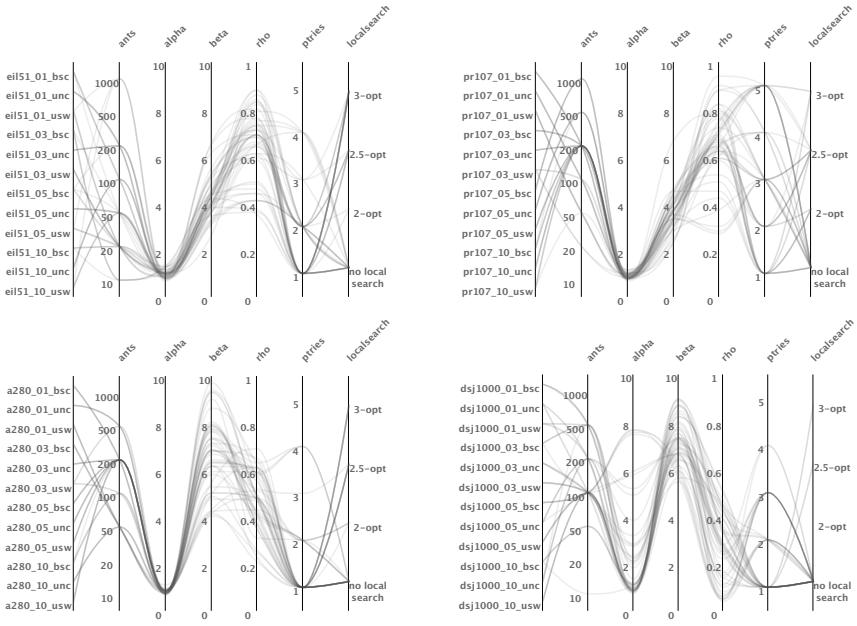


Fig. 1: Best parameter configurations for the 48 groups of instances.

corresponding value. Through the concentration of the lines, we can see which parameter values have been most selected among all tuning experiments.

We can make several observations. For example, the number of ants has a higher concentration between 50 and 200, with a higher frequency between 100 and 200 for the groups of instances that consider the TSP bases *pr107*, *a280*, and *dsj1000*. The importance of the pheromone trail has remained with values close to 1 for all groups of instances. This is generally compensated by the values of *beta*, which varies based on the underlying TSP instance. This is not too surprising, as the underlying TSP instances are different in nature and not normalized, hence requiring different values of beta. We can also observe that only few packing attempts (as exhibited by the low *ptries* values) are needed to reach good results, which is especially true for larger instances. Somewhat similar to this are the settings of the parameter *localsearch*, which attains many values for smaller instances, but which also indicates that TSP-specific local search should be deactivated for larger instances. The indirect reason for this deactivation appears to be that the local search consumes a fair amount of wallclock time (especially for the larger instances) as it calls the packing heuristic every time, which is time that is then not usable by the other components of the overall approach. Furthermore, the fact that high-quality TSP tours do not necessarily result in high-quality ThOP tours might explain why there is no need for an improvement phase based solely on route distances.

In an attempt to furnish a single configuration of parameters that can generalize all tuning results and also be able to provide a more appropriate configuration for new unknown instances, we average the numerical values and take

the mode of the categorical parameters *ptries* and *localsearch*. This results in the following configuration: $ants = 196$, $alpha = 1.24$, $beta = 5.46$, $rho = 0.51$, $ptries = 1$, and *localsearch* = no local search.

4.3 Results

In order to analyze the efficiency of the proposed algorithm on all ThOP instances, we run our ACO algorithm 10 independent times on each instance, and then use the average value of the objective function and the best one found in these runs in our analysis. Our experiments analyze two versions of our ACO algorithm. In the first one, we consider the algorithm set with the best parameter values found by the Irace package (collectively called ACOThOP*). In contrast, the second version uses the general configuration of parameters derived from the Irace results of all tuning experiments (called ACOThOP).

In Figure 2, we assess the quality of our algorithm, in its two versions, by comparing the solutions found by it with the best results reached by the algorithms proposed by Santos and Chagas [25]. For each instance, we consider the best-known solution to be an upper bound on the achievable objective value. Then, we take the average results produced by each approach and compute the ratio between that average and the best objective value found, which gives us the approximation ratio. Note that the higher this metric, the higher the average efficiency of that particular solution method. In the figure, we show the results for the 48 previously defined groups of instances. We report the average approximation ratio obtained for the instances belonging to each group of instances.

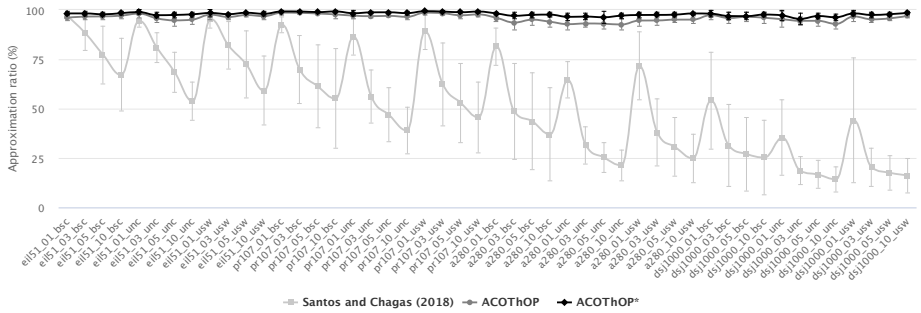


Fig. 2: Approximation ratio of the solution approaches across different groups of instances – whiskers show the standard deviation in the groups.

We can see in Figure 2 that our algorithm has performed significantly better on all groups of instances, especially on those with larger instances. Note that the algorithms proposed by Santos and Chagas [25] are highly affected by the number of items contained in each city, while our framework appears to do better everywhere, and especially well (relatively speaking) on larger instances, where is finds many new best solutions. Our approaches ACOThOP* and ACOThOP outperform the results achieved in [25] on 419 and 410 out of 432 instances,

based on the average solution quality. Regarding the best results found, our approaches have been able to find better solutions for 410 and 402 instances, respectively. On average, considering the best results obtained for all instances, our approaches ACOThOP* and ACOThOP have been, respectively, 320% and 313% better than the best solutions found in [25].

To statistically compare the quality of the solutions, we use the Wilcoxon signed-rank test on the results achieved in the 10 independent runs of each solution method. With a significance level of 5% (p-value < 0.05), the performance compared to [25] is as follows on the 432 instances:

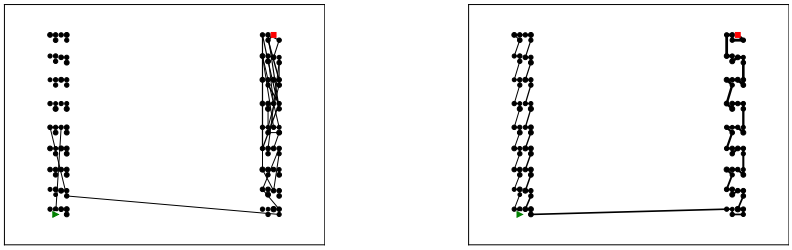
- ACOThOP* is worse in only 2 cases, there is no difference in 21 cases, and it is better in 409 cases (95%).
- ACOThOP is worse in only 18 cases, there is no difference in 12 cases, and it is better in 401 cases (93%).

Table 2 summarizes a closer analysis of the solutions found. For each TSP base instance, which resulted in 108 instances each, we show averaged information concerning all the best solutions achieved by each approach. 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 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. Furthermore, by comparing the values in column $\mathcal{D}$ from the same TSP base instance, we can see which approach has found the most spread-out routes and/or with more edge crossings. As an example, we show this in Figure 3 for the instance *pr107_10_usw_10_03.thop*; this is an instance with a high performance difference between the two shown approaches. The graphical representation of the solutions plots the cities in their respective coordinates. The initial and final cities are represented by a green triangle and a red square, respectively, while black points represent the other cities. The diameter of the point representing a city shows the proportion of profit available in that city. The continuous lines connecting pairs of cities represents the route performed by the thief. The line thickness increases according to the total weight picked by the thief. We can see that our solution has a significantly more efficient route, it travels a shorter distance and, from it, a higher profit has been achieved.

Table 2: Information on the structure of the best solutions found.

TSP base (`XXX`)	Santos and Chagas (2018)			ACOThOP*			ACOThOP		
	$\mathcal{D}$	%T	%W	$\mathcal{D}$	%T	%W	$\mathcal{D}$	%T	%W
eil51	12.6	92.9	73.5	9.7	93.1	82.6	9.8	93.0	81.9
pr107	925.2	96.0	64.7	537.3	99.7	81.8	543.8	99.7	81.0
a280	35.3	97.9	50.5	13.2	98.9	81.4	13.7	98.9	80.3
dsj1000	178520.2	98.6	33.7	30290.7	93.5	82.1	31204.3	93.6	81.1

We can observe in Table 2 that the routes found by our ACO algorithm, in its two versions, are more efficient than those found by Santos and Chagas [25]. Also, we note that the ratio between the total distance traveled and the number of



Santos and Chagas (2018)
Profit = 133925
Distance traveled = 54183

ACOThOP*
Profit = 474464
Distance traveled = 40427

Fig. 3: Graphical representation of the best solution found in [25] (left) and the best solution found by our approach ACOThOP* (right) for the instance *pr107_10_usw_10_03.thop*.

cities visited is higher for the best solutions found in [25], especially for instances with more cities. This behavior directly impacts solutions because they can be quickly limited by the travel time limit, which can be seen when analyzing the columns referring to the solutions found in [25]. As our ACO algorithm – together with our packing routine that fills the knapsack more – has been able to find more efficient routes, a better balance between the limiting factors has been obtained, which resulted in significantly better solutions (see again Figure 2).

5 Concluding remarks

In this work, we have approached the Thief Orienteering Problem (ThOP), a recent academic multi-component problem that combines two classic combinatorial optimization problems: the Orienteering Problem and the Knapsack Problem. We have proposed a two-phase heuristic algorithm based on Ant Colony Optimization, and we have studied the effect of the components using automated algorithm configuration. Our experiments have shown that the best configurations as well as the average configuration are better on over 90% of the 432 instances with an average fitness improvement higher of over 300%; the largest improvements are on the largest instances, when compared to the best solutions in the literature. Based on our analysis, this is due to the efficiency of the ant colony optimization used to determine the thief's route together with our novel, randomized packing routine.

As future work, we will investigate exact algorithms to solve small and mid-sized ThOP instances to establish global optima. Another interesting study will be to address a version of the problem that considers multiple thieves in order to provide a more generic problem, for example, to take a more fundamental approach to the above-mentioned scenarios of the politicians campaigning or the rescue-teams checking safety places.

Acknowledgments. This study has been financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance code 001.

References

1. Aarts, E., Aarts, E.H., Lenstra, J.K.: Local search in combinatorial optimization. Princeton University Press (2003)
2. Aksen, D., Shahmanzari, M.: A periodic traveling politician problem with time-dependent rewards. In: Fink, A., Fügenschuh, A., Geiger, M.J. (eds.) OR. pp. 277–283. Operations Research Proceedings, Springer (2016)
3. Baffo, I., Carotenuto, P., Rondine, S.: An orienteering-based approach to manage emergency situation. Transportation Research Procedia **22**, 297 – 304 (2017), 19th EURO Working Group on Transportation Meeting, EWGT2016, 5-7 September 2016, Istanbul, Turkey
4. 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)
5. Bonyadi, M.R., Michalewicz, Z., Barone, L.: The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In: 2013 IEEE Congress on Evolutionary Computation. pp. 1037–1044. IEEE (2013)
6. 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)
7. Chand, S., Wagner, M.: Fast heuristics for the multiple traveling thieves problem. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 293–300. ACM (2016)
8. Dorigo, M., Birattari, M.: Ant colony optimization. Springer (2010)
9. Dorigo, M., Blum, C.: Ant colony optimization theory: A survey. Theoretical computer science **344**(2-3), 243–278 (2005)
10. Dorigo, M., Di Caro, G.: Ant colony optimization: a new meta-heuristic. In: Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406). vol. 2, pp. 1470–1477. IEEE (1999)
11. Fang, S.H., Lu, E.H.C., Tseng, V.S.: Trip recommendation with multiple user constraints by integrating point-of-interests and travel packages. In: 2014 IEEE 15th International Conference on Mobile Data Management. vol. 1, pp. 33–42 (July 2014)
12. Faulkner, H., Polyakovskiy, S., Schultz, T., Wagner, M.: Approximate approaches to the traveling thief problem. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 385–392. ACM (2015)
13. Freeman, N.K., Keskin, B.B., Çapar, I.: Attractive orienteering problem with proximity and timing interactions. European Journal of Operational Research **266**(1), 354–370 (2018)
14. Gendreau, M., Ghiani, G., Guerriero, E.: Time-dependent routing problems: A review. Computers & Operations Research **64**, 189–197 (2015)
15. Golden, B.L., Levy, L., Vohra, R.: The orienteering problem. Naval Research Logistics **34**, 307–318 (1987)
16. Gonçalves, J.F., Resende, M.G.: Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics **17**(5), 487–525 (2011)
17. 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)
18. Iori, M., Martello, S.: Routing problems with loading constraints. Top **18**(1), 4–27 (2010)

19. López-Ibáñez, M., Cáceres, L.P., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package: User guide. IRIDIA, Université Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2016-004 (2016)
20. 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)
21. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search. In: Handbook of metaheuristics, pp. 320–353. Springer (2003)
22. Neumann, F., Polyakovskiy, S., Skutella, M., Stougie, L., Wu, J.: A fully polynomial time approximation scheme for packing while traveling. In: Disser, Y., Verykios, V.S. (eds.) Algorithmic Aspects of Cloud Computing. pp. 59–72. Springer International Publishing, Cham (2019)
23. Polyakovskiy, S., Bonyadi, M.R., Wagner, M., Michalewicz, Z., Neumann, F.: 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. ACM (2014)
24. 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. pp. 332–346. Springer (2015)
25. Santos, A.G., Chagas, J.B.: The thief orienteering problem: Formulation and heuristic approaches. In: 2018 IEEE Congress on Evolutionary Computation (CEC). pp. 1191–1199. IEEE (2018)
26. Stützle, T., Hoos, H.H.: Max–min ant system. Future generation computer systems **16**(8), 889–914 (2000)
27. Vansteenwegen, P., Souffriau, W., Van Oudheusden, D.: The orienteering problem: A survey. European Journal of Operational Research **209**(1), 1–10 (2011)
28. 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)
29. 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)
30. 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)