



A Non-Dominated Sorting Based Customized Random-Key Genetic Algorithm for the Bi-Objective Traveling Thief Problem

Jonatas B. C. Chagas^{1,2,*}  .

Julian Blank^{3,*}  .

Markus Wagner⁴  .

Marcone J. F. Souza¹  .

Kalyanmoy Deb³  .

Received: date / Accepted: date

Abstract In this paper, we propose a method to solve a bi-objective variant of the well-studied Traveling Thief Problem (TTP). The TTP is a multi-component problem that combines two classic combinatorial problems: Traveling Salesman Problem (TSP) and Knapsack Problem (KP). We address the BI-TTP, a bi-objective version of the TTP, where the goal is to minimize the overall traveling time and to maximize the profit of the collected items. Our proposed method is based on a biased-random key genetic algorithm with customizations addressing problem-specific characteristics. We incorporate domain knowledge through a combination of near-optimal solutions of each sub-problem in the initial population and use a custom repair operator to avoid the evaluation of infeasible solutions. The bi-objective aspect of the problem is addressed through an elite population extracted based on the non-dominated rank and crowding distances. Furthermore, we provide a comprehensive study that shows the influence of each parameter on the performance. Finally, we dis-

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

Julian Blank
E-mail: blankjul@egr.msu.edu

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

Marcone J. F. Souza
E-mail: marcone@iceb.ufop.br

Kalyanmoy Deb
E-mail: kdeb@egr.msu.edu

¹ Departamento de Computação, Universidade Federal de Ouro Preto, Ouro Preto, Brazil

² Departamento de Informática, Universidade Federal de Viçosa, Viçosa, Brazil

³ Michigan State University, East Lansing, MI, USA

⁴ School of Computer Science, The University of Adelaide, Adelaide, Australia

* These authors have contributed equally to this work

cuss the results of the BI-TTP competitions at *EMO-2019* and *GECCO-2019* conferences where our method has won first and second places, respectively, thus proving its ability to find high-quality solutions consistently.

Keywords Combinatorial Optimization · Multi-objective Optimization · Real-world Optimization Problem · Traveling Thief Problem · NSGA-II

1 Introduction

In optimization research, problems with different characteristics are investigated. To find an appropriate algorithm for a practical problem, often assumptions about characteristics are made, and then a suitable algorithm is chosen or designed. For instance, an optimization problem can have several components interacting with each other. Because of their interaction, they build an interwoven system (Klamroth et al. 2017a) where interdependencies in the design and the objective space exist. An optimal solution for each component independently will, in general, not be a good solution for the interwoven optimization problem. Similarly, in multidisciplinary design optimization, various disciplines are linked with each other and influence the objective value(s). The optimization of an aircraft wing, for example, combines stress analysis, structural vibration, aerodynamics, and controls (Jung 1999). Due to the interwovenness modifying a single decision variable is likely to affect all objective values.

Such complex optimization problems usually require domain knowledge and a sufficient amount of computational resources to be invested. For this reason, many researchers prefer solving academic test problems to show the performance of their algorithms. In order to provide an academic interwoven optimization test problem, the Traveling Thief Problem (TTP) (Bonyadi et al. 2013) was proposed in 2013, where two well-known subproblems, the Traveling Salesman Problem (TSP) and the Knapsack Problem (KP), interact with each other. As in the TSP problem, a so-called thief has to visit each city exactly once. In addition to just traveling, the thief can make a profit during its tour by stealing items and putting them in the rented knapsack. However, the thief's traveling speed decreases depending on the current knapsack weight, which then increases the rent that the thief has to pay for the knapsack. Even though researchers have been investigating both subproblems for many years and a myriad of optimization algorithms has been proposed, the interaction of both problems with each other turned out to be challenging. The TTP problem seeks to optimize the overall traveling time and the profit made through stealing items. Most of the research focused on the single-objective problem, where the objectives are composed by using a weighted sum. To be more precise, the profit is reduced by the costs due to renting the knapsack. The costs are calculated by multiplying the overall traveling time by a renting rate. However, because the traveling time and profit represent solutions with different trade-offs, the problem is bi-objective in its natural formulation.

In this article, we propose a Non-Dominated Sorting Biased Random-Key Genetic Algorithm (NDS-BRKGA) to obtain a non-dominated set of solutions for the BI-TTP, a bi-objective version of the TTP, where the goal is to minimize the overall traveling time and to maximize the profit of the collected items. The algorithm is based on the well-known evolutionary multi-objective optimization solution strategy NSGA-II (Deb et al. 2002), and the biased-random key encoding is used to deal with the mixed-variable nature of the problem. The customization makes use of domain knowledge, which is incorporated by evolutionary operators. Our method uses existing solvers of the subproblems for the initial population. It maps a genotype to phenotype to deal with independent variables. Also, solutions detected as infeasible are repaired before they are evaluated. Moreover, we use a customized survival selection to ensure diversity preservation.

The remainder of this paper is structured as follows. In Section 2, we provide a brief review of the literature about the TTP. Afterward, we present a detailed description of the BI-TTP, as well as a solution example to demonstrate the interwovenness characteristic of the problem in Section 3. In Section 4, we describe our methodology to address the problem and present results evaluated on different test problems in Section 5. Finally, the conclusions of the study are presented in Section 6.

2 Related Work

Thus far, many approaches have been proposed for the TTP. Note that most research so far considered the single-objective TTP formulation (TTP1) from Bonyadi et al. (2013), which is typically the TTP variant that is referred to as *the TTP*. The other variant (TTP2) considers two objectives and additionally, a value drop of items over time. The multi-objective considerations of problems with interconnected components are becoming increasingly popular.

For the single-objective TTP, a wide range of approaches has been considered, ranging from general-purpose iterative search-heuristics (Polyakovskiy et al. 2014), co-evolutionary approaches (Bonyadi et al. 2014; Namazi et al. 2019), memetic approaches (Mei et al. 2014), and swarm-intelligence based approaches (Wagner 2016; Zouari et al. 2019), to approaches with problem specific search operators (Faulkner et al. 2015). On a higher, i.e., algorithmic level, estimation of distribution approaches have been customized (Martins et al. 2017) and hyper-heuristics (El Yafrani et al. 2018) have been explored. Wagner et al. (2018) provide a comparison of 21 algorithms for the purpose of portfolio creation. To better understand the effect of operators on the capability to find good solutions on a more fundamental level, Yafrani et al. (2018) and Wuijts and Thierens (2019) present in fitness-landscape analyses correlations and characteristics that are potentially exploitable.

Optimal approaches are rare but exist. Neumann et al. (2019) showed that the TTP with fixed tours can be solved in pseudo-polynomial time via dynamic programming taking into account the fact that the weights are integer. Wu

et al. (2017) extended this to optimal approaches for the entire TTP, although their approaches are only practical for very small instances.

The number of works on multi-objective TTP formulations is significantly lower. Yafrani et al. (2017) created an approach that generates diverse sets of solutions, while being competitive with the state-of-the-art single-objective algorithms; the objectives have been travel time and total profit of items. Wu et al. (2018) considered a bi-objective version of the TTP; the objectives have been the weight and the TTP objective score. This hybrid approach makes use of the dynamic programming approach for fixed tours and then searches over the space of tours only.

Moreover, researchers have participated in the BI-TTP competitions at the *EMO-2019*¹ and *GECCO-2019*² conference. For these competitions, an intermediate version of TTP1 and TTP2 has been proposed. The problem description can be seen as a TTP1 with two objectives or TTP2 without a value drop of items over time. The same TTP variant was investigated by Blank et al. (2017). The proposed problem definition aimed to build the bridge from TTP1 to TTP2 by having two objectives but not adding another complexity to the problem.

A more general discussion of a multi-objective approach to interconnected problems can be found in Klamroth et al. (2017b), and a more general discussion on the opportunities of multi-component problems can be found in Bonyadi et al. (2019).

3 Bi-objective Traveling Thief Problem

The TTP is a combinatorial optimization problem that consists of two interwoven problems, the TSP and KP. In the following, first, the two components are described independently, and then the interaction of the two subcomponents is shown.

In the TSP (Applegate et al. 2007) a salesman has to visit n cities. The distances are given by a map represented as a distance matrix $A = (d_{ij})$ with $i, j \in \{0, \dots, n\}$. The salesman has to visit each city once and the result is a permutation vector $\boldsymbol{\pi} = (\pi_1, \pi_2, \dots, \pi_n)$, where π_i is the i -th city of the salesman. The distance between two cities divided by a constant velocity v (usually $v = 1$) results in the traveling time for the salesman denoted by $f(\boldsymbol{\pi})$. The goal is to minimize the total traveling time of the tour:

$$\begin{aligned} \min \quad f(\boldsymbol{\pi}) &= \sum_{i=1}^{n-1} \frac{d_{\pi_i, \pi_{i+1}}}{v} + \frac{d_{\pi_n, \pi_1}}{v} & (1) \\ \text{s.t.} \quad \boldsymbol{\pi} &= (\pi_1, \pi_2, \dots, \pi_n) \in P_n \end{aligned}$$

¹<https://www.egr.msu.edu/coinlab/blankjul/emo19-thief/>

²<https://www.egr.msu.edu/coinlab/blankjul/gecco19-thief/>

There are $\frac{(n-1)!}{2}$ different tours to consider, if we assume that the salesman has to start from the first city and travels on a symmetric map, i.e., $d_{i,j} = d_{j,i}$, $\forall (i, j) \in A$.

For KP (Lagoudakis 1996) a knapsack has to be filled with items without violating the maximum weight constraint. Each item j has a value $b_j \geq 0$ and a weight $w_j \geq 0$ where $j \in \{1, \dots, m\}$. The binary decision vector $\mathbf{z} = (z_1, \dots, z_m)$ defines, if an item is picked or not. The aim is to maximize the profit $g(\mathbf{z})$:

$$\begin{aligned} \max \quad & g(\mathbf{z}) = \sum_{j=1}^m z_j b_j & (2) \\ \text{s.t.} \quad & \sum_{j=1}^m z_j w_j \leq Q \\ & \mathbf{z} = (z_1, \dots, z_m) \in \mathbb{B}^m \end{aligned}$$

The search space of this problem is exponential concerning n and contains 2^n possible combinations. However, the optimal solution can be obtained by using dynamic programming with a running time of $\mathcal{O}(mQ)$, which makes the complexity of the problem pseudo-polynomial, when the weights of items are integer values.

The traveling thief problem combines the above-defined subproblems and lets them interact with each other. The traveling thief can collect items from each city he/she is visiting. The items are stored in a rented knapsack carried by the thief. In more detail, each city π_i provides one or multiple items, which could be picked by the thief. There is an interaction between the subproblems: The velocity of the traveling thief depends on the current knapsack weight W . It is calculated by considering all cities, which have been visited so far, and summing up the weights of all picked items. The weight at city i given $\boldsymbol{\pi}$ and \mathbf{z} is calculated by:

$$W(i, \boldsymbol{\pi}, \mathbf{z}) = \sum_{k=1}^i \sum_{j=1}^m a_j(\pi_k) w_j z_j \quad (3)$$

The function $a_j(\pi_k)$ is defined for each item j and returns 1 if the item could be stolen at city π_k and 0 otherwise. The current weight of the knapsack influences the velocity. When the thief picks an item, the weight of the knapsack increases, and therefore the velocity of the thief decreases.

The velocity v is always in a specific range $v = (v_{\min}, v_{\max})$ and cannot be negative for a feasible solution. Whenever the knapsack is heavier than the maximum weight Q , the capacity constraint is violated.

$$v(W) = \begin{cases} v_{\max} - \frac{W}{Q} \cdot (v_{\max} - v_{\min}) & \text{if } W \leq Q \\ v_{\min} & \text{otherwise} \end{cases} \quad (4)$$

If the knapsack is empty, then the velocity is equal to v_{\max} . Contrarily, if the current knapsack weight is equal to Q , the velocity is v_{\min} .

Furthermore, the traveling time of the thief is calculated by:

$$f(\boldsymbol{\pi}, \boldsymbol{z}) = \sum_{i=1}^{n-1} \frac{d_{\pi_i, \pi_{i+1}}}{v(W(i, \boldsymbol{\pi}, \boldsymbol{z}))} + \frac{d_{\pi_n, \pi_1}}{v(W(n, \boldsymbol{\pi}, \boldsymbol{z}))} \quad (5)$$

The calculation is based on TSP, but the velocity is defined by a function instead of a constant value. This function takes the current weight, which depends on the index i of the tour. The current weight, and therefore also the velocity, will change on tour by considering the picked items defined by \boldsymbol{z} . In order to calculate the total tour time, the velocity at each city needs to be known. For calculating the velocity at each city, the current weight of the knapsack must be given. Since both calculations are based on \boldsymbol{z} , i.e., the knapsack subproblem solution, it is challenging to solve the problem to optimality. Such problems are called interwoven systems as the solution of one subproblem highly depends on the solution of the other subproblems.

After this preliminary presentation, we finally formalize the TTP as follows.

$$\min f(\boldsymbol{\pi}, \boldsymbol{z}) = \sum_{i=1}^{n-1} \frac{d_{\pi_i, \pi_{i+1}}}{v(W(i, \boldsymbol{\pi}, \boldsymbol{z}))} + \frac{d_{\pi_n, \pi_1}}{v(W(n, \boldsymbol{\pi}, \boldsymbol{z}))} \quad (6)$$

$$\max g(\boldsymbol{z}) = \sum_{j=1}^m z_j b_j$$

$$\text{s.t.} \quad \boldsymbol{\pi} = (\pi_1, \pi_2, \dots, \pi_n) \in P_n$$

$$\pi_1 = 1$$

$$\boldsymbol{z} = (z_1, \dots, z_m) \in \mathbb{B}^m$$

$$\sum_{j=1}^m z_j w_j \leq Q$$

In order to illustrate the equations and interdependence, we present an example scenario here (see Figure 1). The thief starts from city 1 and has to visit city 2, 3, 4 exactly once and to return to city 1. In this example, each city provides one item, and the thief must decide whether to steal it or not.

A permutation vector, which contains all cities exactly once, and a binary picking vector are needed to calculate the objectives. Even though, this is a very small example with four cities and three items the total solution space consists of $(n-1)! \cdot 2^m = 6 \cdot 8 = 48$ combinations.

In order to understand how the objectives are calculated, an example hand calculation for the tour [1,3,2,4] and the packing plan [1,0,1] is done as follows. The thief starts with the maximum velocity, because the knapsack is empty.

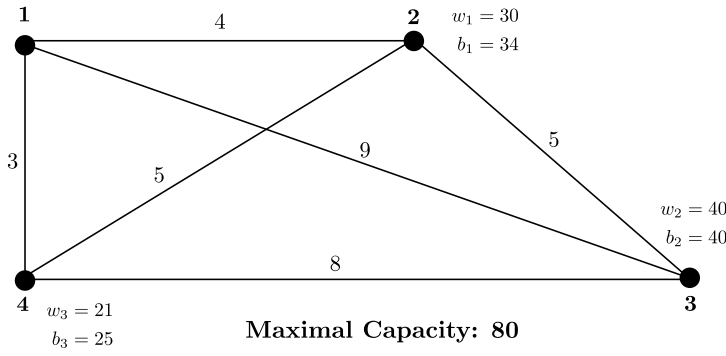


Fig. 1: Exemplary traveling thief problem instance.

He begins its tour at city 1 and picks no item there. For an empty knapsack $W(1, \pi, \mathbf{z}) = 0$ the velocity is $v(0) = v_{\max} = 1.0$. The distance from city 1 to city 3 is 9.0 and the thief needs 9.0 time units. At city 3 the thief will not pick an item and continue to travel to city 2 with $W(2, \pi, \mathbf{z}) = 0$ and therefore with v_{\max} in additional 5.0 time units. Here he picks item 1 with $w_1 = 30$ and the current weight becomes $W(2, \pi, \mathbf{z}) = 30$, which means the velocity will be reduced to $v(30) = 1.0 - \left(\frac{1.0-0.1}{80}\right) \cdot 30 = 0.6625$. For traveling from city 2 to city 4 the thief needs the distance divided by the current velocity $\frac{5.0}{0.6625} \approx 7.5472$. At city 4 he picks item 3 with $w_3 = 21$ and the current knapsack weight increases to $W(4, \pi, \mathbf{z}) = 30 + 21 = 51$. For this reason the velocity decreases to $v(51) = 1.0 - \left(\frac{1.0-0.1}{80}\right) \cdot 51 = 0.42625$. For returning to city 1 the thief needs according to this current speed $\frac{3.0}{0.42625} \approx 7.0381$ time units. Finally, we sum up the time for traveling from each city to the next $\sum_{k=1}^4 t_{\pi_k, \pi_{k+1}} = 9 + 5 + 7.5472 + 7.0381 = 28.5853$ to calculate the whole traveling time.

Table 1: Hand calculations for $[0,2,1,3]$ $[0,1,0,1]$ on the example scenario.

i	π_i	$W(i, \pi, \mathbf{z})$	$v(W(i, \pi, \mathbf{z}))$	$d_{\pi_i, \pi_{i+1}}$	$t_{\pi_i, \pi_{i+1}}$	Σ
1	1	0	1	9	9	-
2	3	0	1	5	5	9
3	2	30	0.6625	5	7.5472	14
4	4	51	0.42625	3	7.0381	21.547
5	1	-	-	-	-	28.585

The final profit is calculated by summing up the values of all items which is $34 + 25 = 59$. Consequently, the TTP solution $[1,3,2,4]$ $[1,0,1]$ is mapped to the point $(28.59, 59.0)$ in the bi-objective space.

Below all Pareto-optimal solutions of this example are listed. The Pareto front contains 8 solutions. The solution for the hand calculation is highlighted in bold.

Table 2: Pareto front of the example scenario.

π	z	$f(\pi, z)$	$g(z)$
[1, 2, 3, 4]	[0, 0, 0]	20.0	0.0
[1, 4, 3, 2]	[0, 0, 0]	20.0	0.0
[1, 2, 3, 4]	[0, 0, 1]	20.93	25.0
[1, 4, 3, 2]	[1, 0, 0]	22.04	34.0
[1, 4, 3, 2]	[0, 1, 0]	27.36	40.0
[1, 3, 2, 4]	[1, 0, 1]	28.59	59.0
[1, 2, 3, 4]	[0, 1, 1]	33.11	65.0
[1, 4, 3, 2]	[1, 1, 0]	38.91	74.0

Figure 2 shows the objective space by highlighting different tours with different markers. The non-dominated solutions are emphasized by black markers. We can observe that different Pareto-optimal solutions can have different underlying tours. In addition, we can see that for each solution s where no item is picked, there is another solution s' with its tour symmetric to the tour of s , and, consequently, both solutions s e s' have the same traveling time, once we consider that the thief travels on a symmetric map.

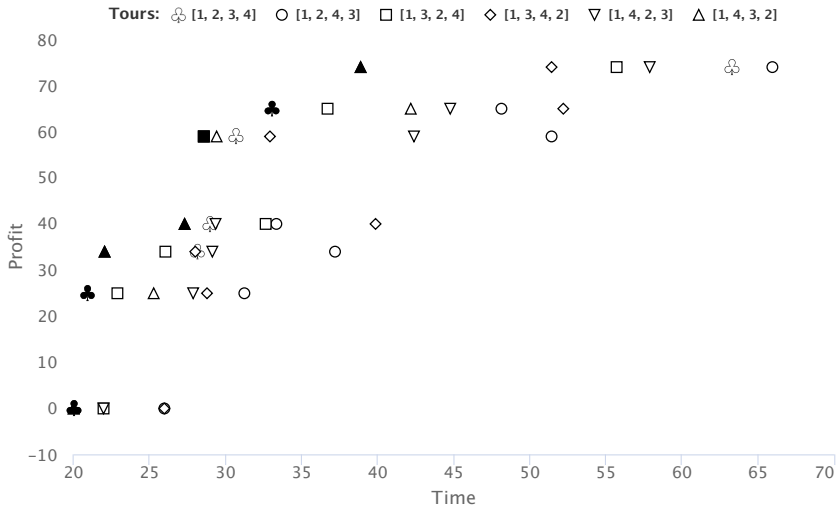


Fig. 2: Exemplary traveling thief problem instance.

4 A Customized Non-dominated Sorting Based Genetic Algorithm with Biased Random-Key Encoding

Genetic algorithms (GAs) provide a good starting point because almost no assumptions about the problem properties are made. GAs are highly customizable, and the performance can be improved through defining/redefining the evolutionary operators. For the BI-TTP, we propose a Non-Dominated Sorting Biased Random-Key Genetic Algorithm (NDS-BRKGA), which combines two classical evolutionary metaheuristics: Biased Random-Key Genetic Algorithm (BRKGA) (Gonçalves and Resende 2011) and Non-Dominated Sorting Genetic Algorithm II (NSGA-II) (Deb et al. 2002). Both concepts come together to address the following characteristics of the BI-TTP:

- (i) **Existing solvers for each subproblem:** Both subproblems, TSP and KP, have been studied for decades, and good solvers for each problem exist. We incorporate this domain knowledge by using a heuristic-based initial population by combining near-optimal solutions of each subproblem. In our initial population, we seek to preserve a high diversity among individuals, in order not to lead our algorithm to premature convergence.
- (ii) **Maximum capacity constraint:** Through a repair operation before any evaluation of an individual, the domain knowledge can be incorporated to avoid the evaluation of infeasible solutions. An effective repair allows the algorithm to search only in the feasible space.
- (iii) **Heterogeneous variable types:** A tour (permutation) and a packing plan (binary decision vector) need to be provided to evaluate a solution. Both variables are linked with each other. Handling different types of variables can be challenging; therefore, we introduce a real-valued genotype by using the biased-random key principle. This allows applying traditional evolutionary recombination operators on continuous variables.
- (iv) **Bi-objective:** The traveling time of the thief is supposed to be minimized, and the profit to be maximized. We consider both conflicting objectives at a time by using the non-dominated sorting and crowding distance in the survival selection. This ensures the final population contains a set of non-dominated solutions with a good diversity in the objective space.

In the remainder of this section, we first explain the overall procedure and then the role of each criterion mentioned above.

Overview

Figure 3 illustrates the overall procedure of NDS-BRKGA. At first, we generate the initial population using efficient solvers for the subproblems independently. Afterward, we combine the optimal or near-optimal solutions for both subproblems and convert them to their genotype representation, which

results in the initial population. For the purpose of mating, the population is split into an elite population $P_e^{(t)}$ and non-elite population $P_g^{(t)}$. The individuals for the next generations $P^{(t+1)}$ are a union of the elite population $P_e^{(t)}$ directly, the offspring of a biased crossover and mutant individuals. In case an individual violates the maximum capacity constraint, we execute a repair operation. Then, we convert each individual to its corresponding phenotype and evaluate it on the problem instance. In order to insert an explicit exploitation phase in our algorithm, we apply at some evolutionary cycles a local search procedure in some elite individuals. Finally, the survival selection is applied, and if the termination criterion is not met, we increase the generation counter t by one and continue with the next generation. In the following, we describe the purpose of each of the design decisions we have made and explain what role it plays during a run of the algorithm.

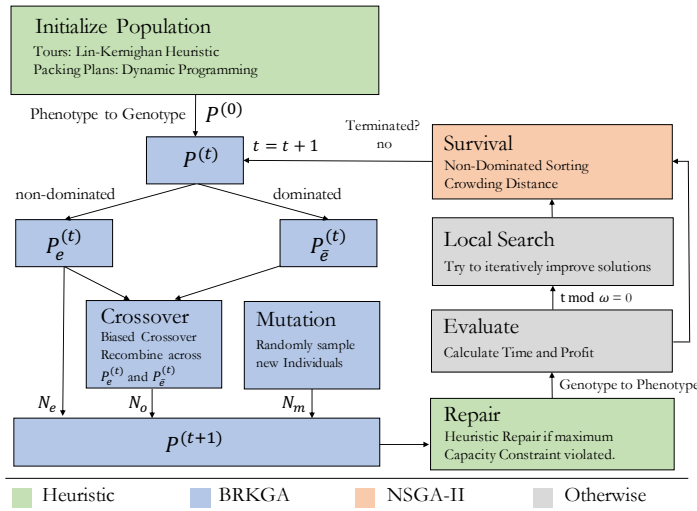


Fig. 3: NDS-BRKGA: A customized genetic algorithm.

Genotype to phenotype decoding

In order to facilitate the exploration of the BI-TTP solution space, we represent the genotype of each individual as a vector of random-keys, which is a vector of real numbers between the interval $[0,1]$. It is an indirect representation that allows us to navigate in the feasible solution space of any optimization problem through simple genetic operators. This representation strategy has been successfully applied to several complex optimization problems (Gonçalves and Resende 2012; Resende 2012; Gonçalves and Resende 2013; Lalla-Ruiz et al. 2014; Gonçalves and Resende 2015; Santos and Chagas 2018).

Because this representation is independent of the problem addressed, a deterministic procedure is necessary to decode each individual to a feasible solution of the problem at hand, i.e., an algorithm that decodes a genotype to its respective phenotype. In Figure 4, we illustrate the genotype and phenotype structure for the BI-TTP. The structure can be divided into two parts, the tour, and the packing plan. The tour needs to be decoded to a permutation vector. It is known that the thief is starting from city 1 and, therefore, the order of the remaining $n - 1$ cities needs to be determined. To achieve this, the sorting of the random key vector with length $n - 1$ forms a permutation from 1 to $n - 1$. Then, each value is increased by 1 to shift the permutation from 2 to n . By appending this permutation to the first city, the tour is decoded to its phenotype. The packing plan needs to be decoded to a binary decision vector of length m . The decision of whether to pick an item or not is made based on the value of the biased random key, which has the same length. If the corresponding value is larger than 0.5, the item is picked up, otherwise not. An exemplary decoding of the biased random key vector $[0.5, 0.1, 0.8, 0.6, 0.1, 0.9]$ (see Figure 4) would be the following: First separate the genotype into two parts $[0.5, 0.1, 0.8]$ and $[0.6, 0.1, 0.9]$. Then, sort the first vector and increase each value by 1 results in the permutation $[3, 2, 4]$. By appending it to the first city, the tour is $\pi = [1, 3, 2, 4]$. For the second part, for each value in $[0.6, 0.1, 0.9]$ we set the bit if it is larger than 0.5 which results in $[1, 0, 1]$. Note that this example decodes to the variable used for our hand-calculation in Section 3. Moreover, the decoding is a many-to-one mapping, which means different genotypes can represent the same phenotype.

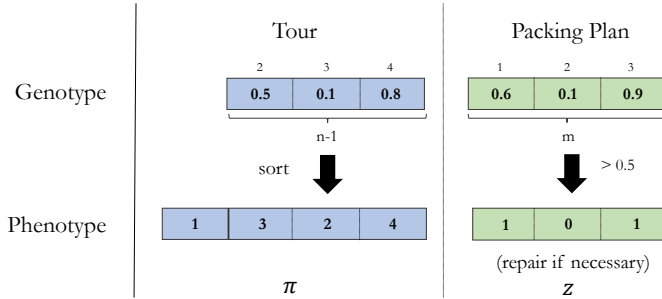


Fig. 4: Chromosome structure: genotype to phenotype.

Repair operator

According to the decoding procedure previously described, a genotype can generate an infeasible phenotype concerning the packing plan. It occurs when the total weight of the picked items is higher than the maximum limit of the knapsack. In order to repair an infeasible genotype, we apply an operator that removes items from the packing plan until it becomes feasible. In this repair

operator, we give preference to keeping items collected last; since this way, the thief can travel faster at the beginning of its journey. Therefore, we first remove all items collected from the first city visited by the thief. If the removal of these items makes the packing plan feasible, the repair operator is finished; otherwise, we repeat the previous step considering all items of the next city visited by the thief. This process repeats until the weight of all remaining items in the packing plan does not exceed the limit of the knapsack. We also repair the genotype to avoid propagating non-feasibility throughout the evolutionary process. For this purpose, we should assign any real number less than 0.5 to every random-key that references an item that has not been collected. In our implementation, we have used the number zero.

Initial population

We use a biased initial population to incorporate domain knowledge into the genetic algorithm. Because both subproblems of the BI-TTP are well-studied, we make use of existing algorithms to generate a good initial population. We maintain a population \mathcal{P} of N individuals throughout the evolutionary process. To create the initial population $\mathcal{P}^{(0)}$, we combine the tour found by TSP and the packing plan by KP solvers. To be not too biased to near-optimal solutions of each subproblem, those combinations represent only a small fraction of the entire population. Because the corresponding solvers provide the phenotype presentation, we convert them to their genotype representation to be able to apply evolutionary operators later on. We complete the population $\mathcal{P}^{(0)}$ by adding randomly created individuals to it, where each random-key is generated independently at random in the real interval $[0, 1]$.

In Algorithm 1, the required steps to create the initial population $\mathcal{P}^{(0)}$ are described in more detail. At first, we use the Lin-Kernighan Heuristic (LKH) (Lin and Kernighan 1973) for solving the TSP component (Line 1). We consider the symmetrical tour found by LKH (Line 2). As we consider that the thief travels on a symmetric map, where both these tours result in the same overall traveling time. Note that achieving near-optimal TSP tours is not a guarantor for near-optimal TTP solutions, and it has been observed that slightly longer tours have the potential to yield overall better TTP solutions (Wagner 2016; Wu et al. 2018). However, we observed that near-optimal TSP tours combined with KP packing with lighter items generate BI-TTP solutions very close to the Pareto front regarding the traveling time objective.

Next, we apply a two-stage heuristic algorithm, which has been developed by us for solving the KP component (Line 3). We named this two-stage heuristic algorithm GH+DP because it combines a Greedy Heuristic (GH) with classical Dynamic Programming (DP) for solving the knapsack problem. The GH+DP algorithm starts by sorting all m items according to profit/weight ratio in non-increasing order. It then proceeds to insert the first m' items such that the total weight $\sum_{i=1}^{m'} w_i$ is not greater than $Q - \delta$ where delta is a parameter of our method. Next, it uses the classic dynamic programming algorithm (Toth 1980) for solving the smaller KP considering the last $m - m'$

Algorithm 1: Generate initial population

```

1  $\pi' \leftarrow$  solve the TSP component // LKH algorithm
2  $\pi'' \leftarrow \{ \pi_1, \pi_n, \pi_{n-1}, \dots, \pi_2 \}$  // symmetric  $\pi'$ 
3  $z' \leftarrow$  solve the KP component // GH + DP algorithm
4  $z'' \leftarrow \emptyset$ 
5  $\mathcal{S} \leftarrow \{ \zeta(\pi', z''), \zeta(\pi'', z'') \}$ 
6 repeat
7    $i \leftarrow$  select item  $i \in z'$  with the highest  $p_i/w_i$  rate
8    $z'' \leftarrow z'' \cup \{ i \}$ 
9    $z' \leftarrow z' \setminus \{ i \}$ 
10  if  $(\pi', z'')$  is not dominated by  $(\pi'', z'')$  then
11     $\mathcal{S} \leftarrow \mathcal{S} \cup \{ \zeta(\pi', z'') \}$ 
12  else
13     $\mathcal{S} \leftarrow \mathcal{S} \cup \{ \zeta(\pi'', z'') \}$ 
14  end
15 until  $z' = \emptyset$ 
16  $\mathcal{A} \leftarrow$  select randomly a set of  $\alpha \times N$  individuals from  $\mathcal{S}$  using a uniform
    distribution
17  $\mathcal{B} \leftarrow$  generate a set of  $(1 - \alpha) \times N$  random individuals
18  $\mathcal{P}^{(0)} \leftarrow \mathcal{A} \cup \mathcal{B}$ 
19 return  $\mathcal{P}^{(0)}$ 

```

$\zeta(\pi, z)$ encodes the BI-TTP solution (π, z) to a vector of random-keys.

items and a knapsack of capacity $Q - \sum_{i=1}^{m'} w_i$. There is no guarantee that near-optimal KP packing plans generate to near-optimal TTP solutions. However, in contrast to single-objective approaches, we have observed that we can generate BI-TTP solutions close to the optimal profit objective by combining near-optimal KP packing plans with efficient TSP tours (see Section 5.4).

Afterward, we combine TSP and KP solutions to create new individuals (Line 4 to 15). Note that we first create two individuals (Line 5) from the tour (and its symmetric tour) found by LKH and from the empty knapsack solution. Next, iteratively, we create new non-dominated individuals so that at each iteration a single individual is created from the TSP solutions previously considered and also from a partial solution of the KP solution found by GH+DP algorithm. After creating all individuals, we select only a subset of them to compose the initial population. We randomly select $\alpha \times N$ (α is a parameter with its value between 0 and 1) individuals uniformly distributed from all individuals generated (Line 16), then we generate $(1 - \alpha) \times N$ random individuals (Line 17) in order to complete the initial population, which is returned at the end of algorithm (Line 19).

Elite and non-elite population

It is a common strategy of multi-objective optimization algorithms to give more importance to non-dominated solutions in the population during the recombination and environmental survival Chand and Wagner (2015). We split the population into two groups: the elites and non-elites. The number of elites is defined beforehand by the parameter N_e . We use the survival selection of

NSGA-II (Deb et al. 2002) as a splitting criterion (see Figure 5). The current population $P^{(t)}$ and the offspring $Q^{(t)}$ are merged together and non-dominated sorting is applied. The outcome is a number of fronts F_1, F_2, \dots, F_L , each of which is a set of individuals. Because the survival selection requires to select only $|P^{(t)}|$ individuals from the merged population, it might be the case that a front needs to be split into surviving and non-surviving individuals. In our example, F_1 and F_2 are surviving individuals because of their non-domination criterion. However, F_3 needs to be split. Therefore, a second criterion, crowding distance is introduced. Based on the distance to neighboring individuals in the objective space, a crowding distance metric is calculated and assigned to each individual.

We use the non-dominated sorting and crowding distance to incorporate elitism. As is usually done, before calculating the crowding distance, we normalize the objectives in order to avoid a possible higher influence of a single objective. The number of elite individuals N_e is determined by executing the NSGA-II survival on our current population $P^{(t)}$ with the goal to let N_e individuals survive. The resulting survivors are added to the group of elites $P_e^{(t)}$ and the remaining to the group of non-elites $P_n^{(t)}$.

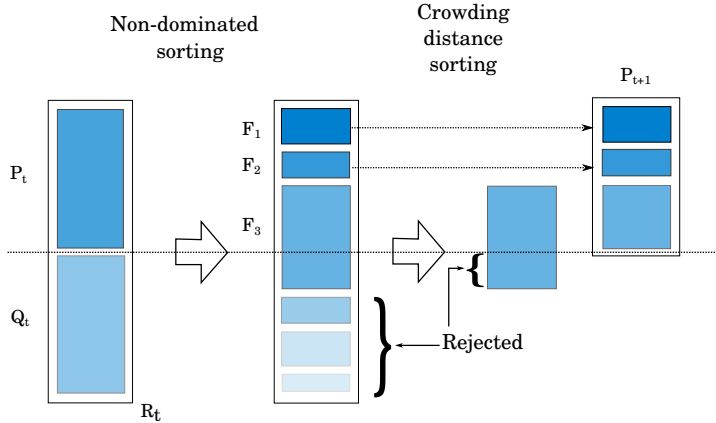


Fig. 5: NSGA-II survival selection (Illustration based on Deb et al. (2002)).

Biased crossover

With the purpose of diversity preservation, we apply a biased crossover in order to create new offspring individuals. It is common practice when random-keys are used as a genotype representation. The biased crossover operator involves two parents. The first is randomly selected from the elite population $P_e^{(t)}$ and the second randomly from the whole population $P^{(t)}$. Moreover, the biased crossover operator has a parameter ρ_e which defines the probability of each random-keys of the first parent (it always belongs to the elite population) to

be inherited by the offspring individual. More precisely, from an elite parent a and another any parent b , we can generate an offspring c according to the biased crossover as follows:

$$c_i \leftarrow \begin{cases} a_i & \text{if } \text{RAND}(0, 1) \leq \rho_e \\ b_i & \text{else} \end{cases} \quad \forall i \in \{1, 2, \dots, n-1+m\}$$

where a_i , b_i and c_i are, respectively, the i -th random-key of individuals a , b and c .

Mutant individuals

As in BRKGAs, we are not using any mutation operator, which are commonly used in most GAs (Mitchell 1998). In order to maintain diversity in the population, we use so called mutant individuals. Mutant individuals are simply randomly created individuals where each random-key is sampled from a uniform distribution in the $[0, 1]$ range.

Survival

The population of the next generation $\mathcal{P}^{(t+1)}$ is formed based on the current population $\mathcal{P}^{(t)}$. The survival is the union of three different groups of individuals:

- (i) **Elite population:** Part of the survival is based on elitism. Before the mating, a sub-population P_e of N_e individuals are selected as elites according to the NSGA-II criteria. Which means non-dominated sorting determines the rank of each solution, and then for each non-dominated front the crowding distance is calculated. In case of a tie, two solutions are ordered randomly. Based on this absolute ordering we pick N_e individuals and directly copy them to $\mathcal{P}^{(t+1)}$.
- (ii) **Mutant individuals:** In order to maintain a high diversity, a population P_m of N_m mutant individuals are added to $\mathcal{P}^{(t+1)}$. The strategy of keeping a separate set with mutants introduces a diversity of offspring during the mating. Otherwise, the recombination would be biased towards elites in the population, and a premature converge through a loss of diversity is likely.
- (iii) **Offsprings from biased crossover:** To complete the number of individuals in the next population $\mathcal{P}^{(t+1)}$, $N - N_e - N_m$ individuals are generated and added on it through mating by the biased crossover operator. The biased crossover chooses one parent from the elites and another one from the non-elites. This mating is even more biased towards the elite population than the traditional binary tournament crossover used in NSGA-II, because it forces for each mating a non-dominated solution to participate in it.

Finally, the surviving individuals are obtained by merging these three sets together $\mathcal{P}^{(t+1)} = P_e \cup P_m \cup P_o$. The survival is partly based on elitism through

letting P_e survive for sure, but also adds two more diverse groups through evolutionary operators.

Local search

At some evolutionary cycles, we apply an exploitation procedure of the search space by modifying the genotypes of some individuals to enhance the fitness of the current population. This methodology is commonly applied to traditional genetic algorithms in order to balance the concepts of exploitation and exploration, which are aspects of effective search procedures (Neri and Cotta 2012). Genetic Algorithms (GAs) with exploitation procedure are known and widely referenced as Memetic Algorithms (MAs). According to Krasnogor and Smith (2005), MAs have been demonstrated to be more effective than traditional GAs for some problem domains, especially for combinatorial optimization problems.

In NDS-BRKGA, the local search is only applied to some percentage of the population and consists of two phases: First the tour π is considered separately and the permutation is modified; Second, it considers only the packing plan z and through bit-flips items are either removed or added. In Algorithm 2 we describe the exploitation phase in more detail. In order to ensure a high diversity in the current population, we execute a local search only for 10% of all elite individuals (Line 1). Initially, we decode each individual to its phenotype consisting of a tour π and a packing plan z (Line 3). Then, the two phases of local optimizations are considered. First, we apply a limited local search procedure (Line 4 to 7) to the tour π . The local search makes use of the well-known 2-opt move the permutation vectors. This principle has been successfully incorporated to solve various combinatorial optimization problems, including the single-objective TTP (El Yafrani and Ahiod 2016, 2018). We limited the number of 2-opt moves to a small value $LS_\pi = \min(100, n^2)$ since the exploitation phase may become computationally expensive when large instances are considered. After all 2-opt moves have been executed, the tour π and the packing plan z are added to the elite population P_e if it is not dominated by any solution in P_e (Line 8). Note that in the first phase, we do not change the packing plan z , which means the second objective of the problem (KP component) remains unchanged. Second, we intend to improve the packing plan z by applying bit-flip random moves (Line 9 to 15). The bit-flip is also a well-known operator widely used in combinatorial optimization problems, including the single-objective TTP as well (Faulkner et al. 2015; Chand and Wagner 2016). Again because of the computational expensiveness, we apply $LS_z = \min(100, m)$ random bit-flip moves (Line 10-11). As before, the new BI-TTP solution (π, z') is insert into P_e if it is not dominated by any solution in P_e (Line 13). Finally, if P_e contains more than N_e solutions (Line 17), we select the best N_e according to the their non-dominated rank and crowding distance (Line 18).

To balance the exploration and exploitation phases in a run of the algorithm, we apply the exploitation phase only at every ω evolutionary cycles, which is another parameter of our proposed method.

Algorithm 2: Exploitation phase.

```

1  $\widehat{P}_e \leftarrow$  select randomly  $0.1N_e$  individuals from current elite population  $P_e$ 
2 foreach  $p \in \widehat{P}_e$  do
3    $(\pi, z) \leftarrow$  decode  $p$  in a feasible BI-TTP solution
4   for  $i \leftarrow 1$  to  $LS_\pi$  do
5      $\pi' \leftarrow$  generate a random 2-opt move in  $\pi$ 
6     if  $(\pi', z)$  is not dominated by  $(\pi, z)$  then  $(\pi, z) \leftarrow (\pi', z)$ 
7   end
8   if  $(\pi, z)$  is not dominated by any solution in  $P_e$  then  $P_e \leftarrow P_e \cup \{ \zeta(\pi, z) \}$ 
9   for  $i \leftarrow 1$  to  $LS_z$  do
10     $item \leftarrow$  select randomly an  $item \in \{1, 2, \dots, m\}$ 
11    if  $item \in z$  then  $z' \leftarrow z \setminus \{item\}$  else  $z' \leftarrow z \cup \{item\}$ 
12    if  $(\pi, z')$  is not dominated by any solution in  $P_e$  then
13       $P_e \leftarrow P_e \cup \{ \zeta(\pi, z') \}$ 
14    end
15  end
16 end
17 if  $|P_e| > N_e$  then
18    $P_e \leftarrow$  select  $N_e$  individuals according to the NSGA-II criteria
19 end

```

$\zeta(\pi, z)$ encodes the BI-TTP solution (π, z) to a vector of random-keys.

5 Computational Experiments

In this section, we present the computational experiments we have employed to study the performance of our proposed method. We have chosen C/C++ as a programming language and have used BRKGA framework developed by [Toso and Resende \(2015\)](#) and the Lin-Kernighan Heuristic (LKH), version 2.0.9³. The experiments have been executed on a high-performance cluster where each node is equipped with Intel(R) Xeon(R) 2.30 GHz processors. Each run of our algorithm has been sequentially (nonparallel) performed on a single processor. Our source code as well as all non-dominated solutions found for each test instance considered our available online⁴.

In the following, we evaluate the performance of our proposed method on a variety of test instances. To be neither biased towards test instances with only a small or large number of cities and items, we have selected test instances with the purpose to cover different characteristics of the problem. Due to the design decisions we have made during the algorithm development, we provide a detailed parameter study to show the effectiveness of each customization of the evolutionary algorithm. Moreover, we present the rankings of competitions where we submitted our implementation to.

³Available at <http://akira.ruc.dk/~keld/research/LKH/>

⁴Available at https://github.com/jonatasbcchagas/nds-brkga_bi-ttp

Table 3: BI-TTP test instances.

Instance	n	m	Q	Knapsack Type	R
a280_n279	280	279	25936	bsc	01
a280_n1395		1395	637010	usw	05
a280_n2790		2790	1262022	unc	10
fnl4461_n4460	4461	4460	387150	bsc	01
fnl4461_n22300		22300	10182055	usw	05
fnl4461_n44600		44600	20244159	unc	10
pla33810_n33809	33810	33809	2915215	bsc	01
pla33810_n169045		169045	77184794	usw	05
pla33810_n338090		338090	153960049	unc	10

5.1 Test instances

In order to analyze the performance, we have considered nine medium/large instances from the comprehensive TTP benchmark developed by Polyakovskiy et al. (2014). These instances, which are described in Table 3, have been used in the BI-TTP competitions at *EMO-2019*⁵ and *GECCO-2019*⁶ conferences.

From Table 3, we can observe the characteristics of the instances, which involve 280 to 33810 cities (column n), 279 to 338090 items (column m), and knapsacks (column Q). Furthermore, the knapsack component of each instance has been built according to the profit/weight ratio of items in three different ways: bounded strongly correlated (bsc), uncorrelated with similar weights (usw), and uncorrelated (unc). To diversify the size of the knapsack component, it has been defined for each instance how many items per city are available (column R). For instance, when $R = 10$, then 10 items are available in each city. For example, assuming a problem with 280 cities, this results in 2790 items in total (as the first city never has any items).

5.2 Parameter Study

Customization often involves adding new parameters to the algorithm. Therefore, it is crucial to ensure the parameters are chosen well concerning the performance of the algorithm on a variety of test problems. For this reason, we investigate the influence of parameters in our proposed method. In the experiment, we run a systemic setup of parameters to finally draw conclusions regarding their performance on the different types of test instances. Finally, we provide suggestions on how to choose parameters for new unknown problems.

Our proposed method has eight parameters in total which are shown in Table 4: Population size N , elite population size N_e , mutant population size N_m , elite allele inheritance probability ρ_e , fraction α of the initial population

⁵<https://www.egr.msu.edu/coinlab/blankjul/emo19-thief/>

⁶<https://www.egr.msu.edu/coinlab/blankjul/gecco19-thief/>

Table 4: Parameter overview.

Parameter	Description
N	Population Size
N_e	Number of Elites
N_m	Number of Mutant Individuals
ρ_e	Probability of Elite during Biased Crossover
α	Fraction of near-optimal Solutions obtained by solving TSP or KP independently
ω	Frequency of Local Search Procedures
TSP - t	Time in Seconds to solve the TSP problem
KP - δ	Gap of Knapsack Capacity Q in between different KP Optimizer Runs

Table 5: Influence of execution time on the LKH heuristic algorithm.

TSP comp.	t in seconds						
	60	180	300	420	600	1800	3600
a280	0.0000%	0.0000%	0.0000%	0.0000%	0.0000%	0.0000%	0.0000%
fnl4461	0.0180%	0.0078%	0.0035%	0.0028%	0.0028%	0.0000%	0.0000%
pla33810	0.6653%	0.5964%	0.4261%	0.3339%	0.2377%	0.1084%	0.0837%
Avg.	0.2278%	0.2014%	0.1432%	0.1122%	0.0802%	0.0361%	0.0279%

created from TSP and KP solutions (see Algorithm 1), and the frequency ω , in terms of evolutionary cycles in which a local search is applied (see Algorithm 2). Furthermore, two subproblem dependent parameters have to be defined: t , which is the upper bound for the time of the TSP solvers to be executed, and δ , which is the number of different KP capacities that should be considered. To evaluate the influence of each parameter, we conduct several experiments.

In this parameter study, we first investigate the effect of t and δ on the performance. Both variables affect the initial population that consists partly of solutions from TSP and KP solvers. For solving the TSP component independently, we have used the Lin-Kernighan Heuristic (LKH). The LKH is one of the most efficient algorithms for generating optimal or near-optimal solutions for the symmetric traveling salesman problem. Naturally, the LKH has higher computational costs as TSP instances increase. To balance the computational cost and the quality of the solution achieved, we limit the LKH execution time to different values and compare the obtained solution with the optimal solution. As LKH has random components, we run it ten independent times and use the average reached by them. In Table 5, for each TSP instance and each runtime, we show the relative percentage difference between the solution obtained with limited time and the TSP optimal solution.

Table 6: Influence of δ on GH+DP algorithm.

KP comp.	$\delta = 10^3$		$\delta = 5 \cdot 10^3$		$\delta = 10^4$		$\delta = 5 \cdot 10^4$		$\delta = 10^5$		$\delta = 5 \cdot 10^5$	
	d_p^*	t(s)	d_p^*	t(s)	d_p^*	t(s)	d_p^*	t(s)	d_p^*	t(s)	d_p^*	t(s)
n279	0	0	0	0	0	0	0	0	0	0	0	2
n1395	0	0	0	0	0	0	0	3	0	33	0	1364
n2790	0	0	0	0	0	0	0	2	0	18	0	1339
n4460	0	0	0	0	0	1	0	54	0	162	0	9007
n22300	27	0	12	1	12	2	0	105	0	265	0	10234
n44600	0	0	0	0	0	1	0	47	0	136	0	5193
n33809	0	1	0	3	0	6	0	261	0	1149	0	22945
n169045	298	3	298	7	295	14	228	437	11	3980	0	35711
n338090	0	1	0	2	0	5	0	226	0	928	0	19594
Avg.	36.1	0.6	34.4	1.4	34.1	3.2	32.0	126.1	1.2	741.2	0.0	11708.8

Table 5 shows that even for shorter computational times LKH is efficient. On average, LKH has been able to find solutions with a gap of less than 0.23% to the optimal solutions, even considering only 60 seconds of processing. Naturally, the quality of solutions increases with longer computational time. As we do not pursue spending a significant amount of time solving the TSP independently, we have limited the LKH in 300 seconds in our implementation. The experiment indicates that this is sufficient to produce near-optimal TSP solutions to be used in the initial population.⁷

Moreover, the parameter δ used in the KP solver has to be studied and the performance of the GH+DP algorithm evaluated. This algorithm has been developed for solving the KP component independently. For each KP instance, we have run the GH+DP for different values of δ and have measured the quality of the solutions obtained. Table 6 shows the difference between the solution obtained and the KP optimal solution d_p^* for different δ values. In addition to the difference, it provides the computational time required in seconds $t(s)$. We can observe that for almost all instances, GH+DP has been able to find the KP optimal solution even with a relatively small δ . Moreover, the larger δ , the larger the fraction of the knapsack solved using the dynamic programming algorithm, hence the higher quality of the solution and the longer the computation time. In order to find the best possible solutions for the KP component within a reasonable time, we have chosen to use $\delta = 5 \cdot 10^4$.

This preliminary study on subproblem solvers has shown that $t = 300$ for the TSP solver and $\delta = 5 \cdot 10^4$ for KP solver seem to be reasonable parameters regarding the trade-off of running time and quality of solutions. So far, we have evaluated the goodness of each of the subproblems without considering the interwovenness aspect. Next, the parameter α defines how many solutions are used from those subproblem solvers during the initialization. To draw conclusions about the remaining six parameters, we have conducted an experiment

⁷Note that we rotate the computed tours to conform with the requirement for all TTP tours to start and finish in city number 1.

Table 7: Parameter values considered during the experiment.

Parameter	Values
N	100, 200, 500, 1000
N_e	$0.3N$, $0.4N$, $0.5N$, $0.6N$
N_m	$0.0N$, $0.1N$, $0.2N$
ρ_e	0.5, 0.6, 0.7, 0.8
α	0.0, 0.1, 0.2, 0.3
ω	1, 10, 50, 100

with predefined parameter settings. The considered parameter values for each parameter are shown in Table 7. We have considered all 3072 possible combinations that can be formed by combining these values. Because the proposed method contains components with underlying randomness, we have run each parameter configuration 10 times for 5 hours. Altogether, the experiments have consumed 1,382,400 CPU hours, which is equivalent to almost 158 CPU years.

We use the hypervolume indicator (HV) (Zitzler and Thiele 1998) as a performance indicator to compare and analyze results obtained from the set of parameter configurations. It is one of the most used indicators for measuring the quality of a set of non-dominated solutions by calculating the volume of the dominated portion of the objective space bounded from a reference point. Considering the BI-TTP, it considers the dominated volume regarding the minimum time and the maximum profit. Note that maximizing the hypervolume indicator is equivalent to finding a good approximation of the Pareto front,⁸ thereby the higher the hypervolume indicator, the better the solution sets are (in general terms). To make the hypervolume suitable for the comparison of objectives with greatly varying ranges, these need to be normalized beforehand. Therefore, we have first normalized the values of the objectives between 0 and 1 according to their minimum and maximum value found during the parameter tuning experiments before computing the hypervolume.

Figure 6 shows the convergence according to the hypervolume indicator for each instance throughout 5 hours, considering 10-minute intervals. For each interval, we have plotted the result of the best parameter configuration found. Each parameter configuration is described in Table 8. It is important to note that the vertical axis (hypervolume values) of the plots are not on the same scale. The figure shows that our proposed method has been able to quickly converge for most of the instances, indicating that our algorithm does not need excessive processing time to achieve good solutions.

Table 8 shows that the best parameter configuration for each instance changes over runtime. However, the number of changes among them decreases as the runtime increases, which means our method keeps stable regarding

⁸However, maximizing the hypervolume is not equivalent to finding the optimal approximation, see, e.g., Bringmann and Friedrich (2013); Wagner et al. (2015)

Table 8: Best parameter configurations as observed in the parameter study.

Instance	Runtime	Best parameter configuration						
		N	N_e	N_m	ρ_e	α	ω	
a280_n279	600	500	0.4	0.1	0.7	0.1	10	
	1200	500	0.4	0.1	0.7	0.2	10	
	1800	500	0.5	0.0	0.5	0.2	10	
	2400 - 18000	500	0.3	0.2	0.8	0.3	50	
a280_n1395	600	1000	0.6	0.0	0.6	0.3	10	
	1200	1000	0.6	0.0	0.5	0.2	10	
	1800 - 2400	1000	0.6	0.1	0.5	0.2	50	
	3000 - 4800	1000	0.6	0.0	0.5	0.2	50	
	5400 - 9600	1000	0.6	0.0	0.6	0.2	50	
	10200 - 15600	1000	0.6	0.0	0.5	0.3	100	
	16200 - 18000	1000	0.6	0.1	0.8	0.3	50	
a280_n2790	600	1000	0.6	0.0	0.6	0.1	100	
	1200	1000	0.6	0.0	0.5	0.1	50	
	1800 - 3600	1000	0.6	0.0	0.7	0.2	50	
	4200	1000	0.6	0.0	0.7	0.1	50	
	4800 - 6000	1000	0.6	0.0	0.5	0.3	50	
	6600 - 12000	1000	0.6	0.0	0.5	0.1	50	
	12600 - 14400	1000	0.6	0.0	0.5	0.3	50	
	15000	1000	0.6	0.0	0.5	0.2	50	
	15600 - 18000	1000	0.6	0.0	0.5	0.3	50	
fnl4461_n4460	600	500	0.6	0.0	0.5	0.1	50	
	1200 - 4200	1000	0.6	0.0	0.7	0.1	100	
	4800 - 9600	1000	0.6	0.0	0.6	0.2	100	
	10200 - 13800	1000	0.6	0.0	0.5	0.2	50	
	14400 - 18000	1000	0.6	0.0	0.5	0.3	50	
fnl4461_n22300	600	1000	0.5	0.0	0.8	0.1	10	
	1200	1000	0.6	0.0	0.8	0.2	50	
	1800 - 6600	1000	0.6	0.0	0.5	0.2	100	
	7200	1000	0.6	0.0	0.5	0.3	100	
	7800	1000	0.6	0.0	0.5	0.2	100	
	8400 - 10200	1000	0.6	0.0	0.5	0.3	100	
	10800 - 18000	1000	0.6	0.0	0.5	0.1	50	
fnl4461_n44600	600	1000	0.6	0.0	0.8	0.3	100	
	1200	1000	0.6	0.0	0.8	0.3	100	
	1800 - 15600	1000	0.6	0.0	0.5	0.3	100	
	16200 - 18000	1000	0.6	0.0	0.5	0.2	100	
pla33810_n33809	600	1000	0.3	0.2	0.5	0.1	1	
	1200	1000	0.6	0.0	0.7	0.2	10	
	1800	1000	0.5	0.0	0.8	0.3	10	
	2400	1000	0.6	0.0	0.7	0.3	50	
	3000	1000	0.6	0.0	0.5	0.2	50	
	3600	1000	0.6	0.0	0.6	0.2	50	
	4200	1000	0.6	0.0	0.5	0.2	50	
	4800	1000	0.6	0.0	0.6	0.1	50	
	5400	1000	0.6	0.0	0.6	0.3	50	
	6000 - 11400	1000	0.6	0.0	0.6	0.1	50	
	12000 - 18000	1000	0.6	0.0	0.6	0.1	10	
	pla33810_n169045	600	500	0.5	0.0	0.8	0.3	10
		1200	100	0.3	0.1	0.8	0.3	1
1800		500	0.5	0.0	0.8	0.2	1	
2400 - 3000		500	0.3	0.0	0.8	0.3	1	
3600 - 13800		1000	0.3	0.0	0.8	0.3	1	
14400 - 18000		1000	0.4	0.2	0.8	0.3	1	
pla33810_n338090		600	1000	0.3	0.1	0.8	0.3	1
	1200 - 4800	1000	0.6	0.0	0.8	0.3	100	
	5400 - 8400	1000	0.6	0.0	0.7	0.3	100	
	9000 - 14400	1000	0.6	0.2	0.5	0.3	100	
	15000	1000	0.4	0.0	0.8	0.3	1	
	15600 - 18000	1000	0.3	0.1	0.8	0.3	1	

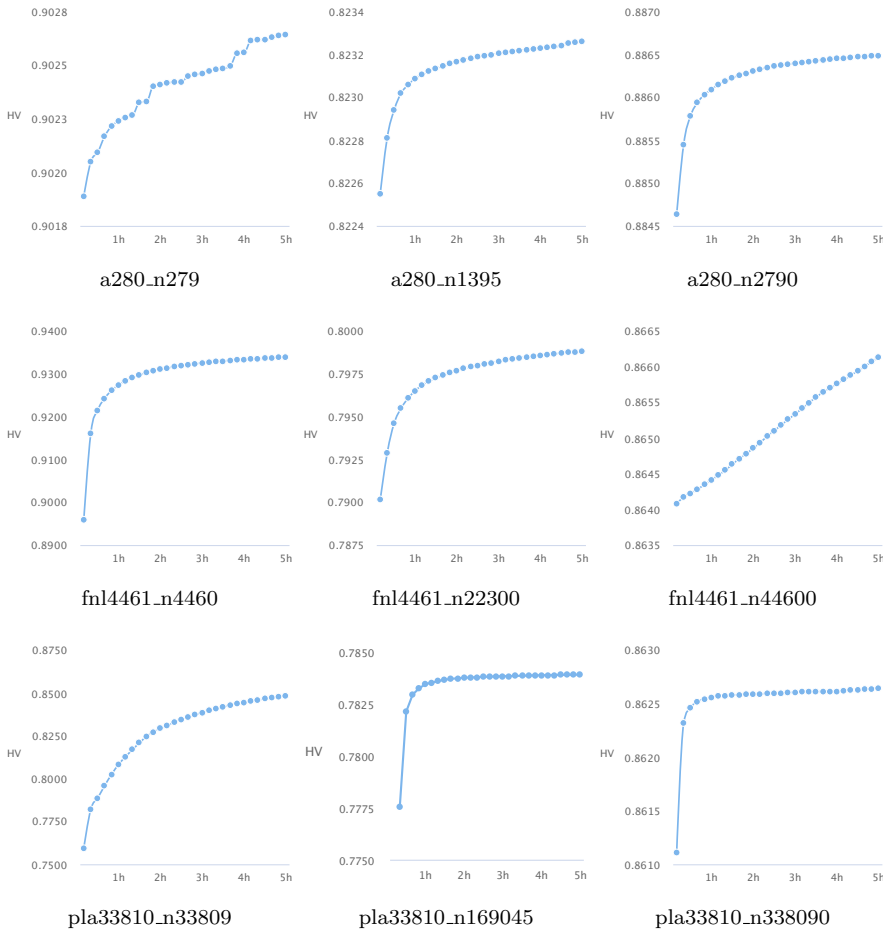


Fig. 6: Convergence plots according to the hypervolume indicator.

its best parameter configuration as the runtime increases. Although a single parameter configuration cannot extract the best performance by considering all instances, we can observe some patterns and trends among the different parameter configurations. A good parameter configuration is a population with a larger number of individuals, higher survival rate for the best individuals, insignificant contribution from mutant individuals, and high contribution of the TSP and KP solvers for creating part of the initial population.

In the following, we analyze the behavior of the parameters considering all instances together. In Figure 7, we can visualize the best parameter configurations at six different execution times. In each plot, the best obtained parameter configuration regarding hypervolume is highlighted in red and parameter configurations up to 0.1% worse than the best are highlighted in blue.

Note that the intensity of the blue color indicates the importance of values of each parameter among the best parameter configurations once some parameter configurations share some parameter values. The following can be observed:

- (i) **More execution time, better results:** The number of parameter configurations that are capable of generating large hypervolume values increases as the execution time of our algorithm increases. This means that in some runs, even though the parameters have not been set appropriately, the algorithm is still able to converge.
- (ii) **Importance of TSP and KP solvers:** It influences the overall performance of the algorithm if TSP and KP solvers are used for initialization, which is determined by α . The best results are obtained if at least 10% percent of the initial solutions are biased towards those solutions found a TSP and KP solvers.
- (iii) **Trends when execution time increases:** We can see a trend as the execution time increases. Our method performs better with a large population, a large survival rate, a small or no explicit diversification through mutant individuals, a small influence of single-parent inheritance, a minor influence of a good initial population, and significant influence of local search procedure.

5.3 Competition Results

In order to analyze the efficiency of NDS-BRKGA compared to other methods, we present the results of the BI-TTP competitions held at *EMO-2019* and *GECCO-2019* conferences, where our method has been used and its solutions have been submitted. Both competitions did not have any regulations regarding running time, number of processors, or any other constraint and the ranking was solely based on the solution set submitted by each participant. Following the criteria of both competitions, we have compared the efficiency of the solutions of each submission for each test instance according to the hypervolume indicator. For each instance, the reference point used to calculate the hypervolume has been defined as the maximum time and the minimum profit obtained from the non-dominated solutions, which have been built from all submitted solutions.

In both competitions, the number of accepted solutions for each instance has been limited: For test instances based on *a280* to 100, *fnl4461* to 50, and for *pla33810* to 20. Because NDS-BRKGA returns a non-dominated set of solutions (named here as A), where its size can be larger than the maximum number of solutions q accepted, we apply the dynamic programming algorithm developed by Auger et al. (2009), which is able to find a subset $A^* \subseteq A$ with $|A^*| = q$, such that the weighted hypervolume indicator of A^* is maximal. As stated by Auger et al. (2009), this dynamic programming can be solved in time $\mathcal{O}(|A|^3)$.

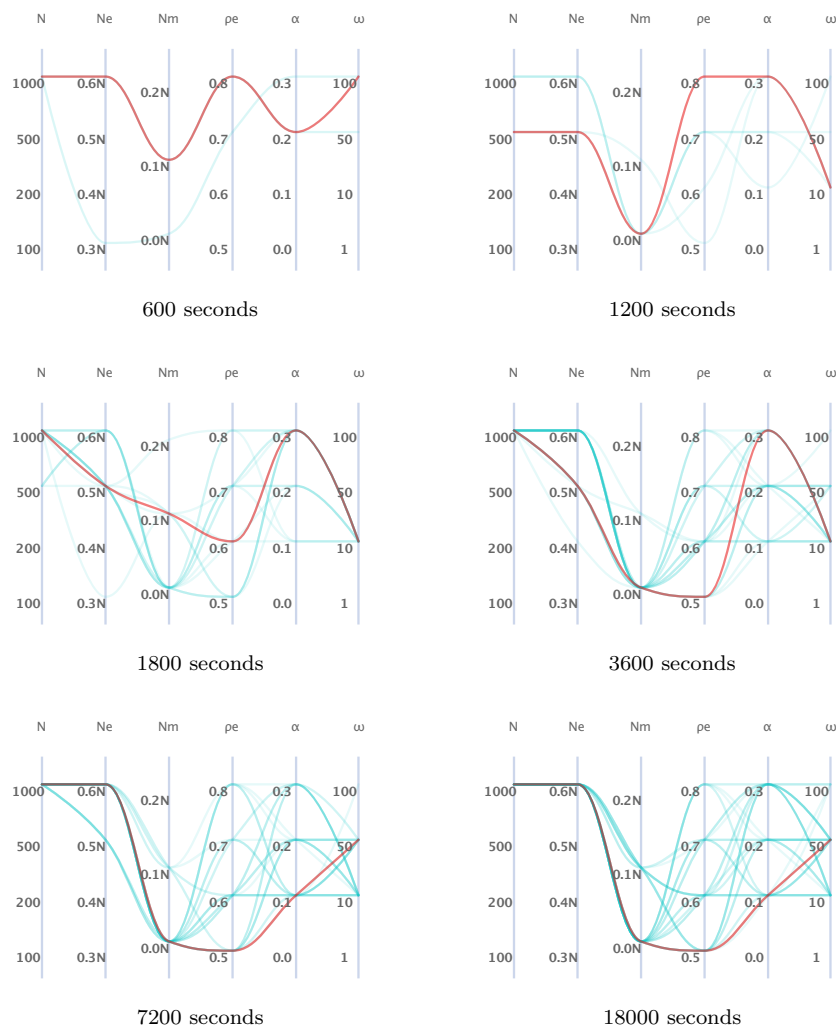


Fig. 7: Best parameter configurations over all instances with varying execution times.

For the *EMO-2019* competition, we have used a preliminary version of the NDS-BRKGA described in Section 4. At that time, our method did not use the exploitation phase, which is summarized in the Algorithm 2. In addition, the initial population of the algorithm used in that version has been created essentially at random. Only four individuals have been created from the TSP and KP solutions, which have been obtained by the same solvers previously described in this work. More precisely, those four individuals have been built from BI-TTP solutions (π, \emptyset) , (π', \emptyset) , (π, z) , and (π', z) , where π is the tour

found by LKH algorithm, π' is the symmetric tour to π , and z is the packing plan for the knapsack obtained by GH+DP algorithm.

In addition to our method, five other teams also submitted their solutions to the *EMO-2019* competition. Among all submissions, NDS-BRKGA has had the best performance in seven of the nine test instances, resulting in the first-place award. All competition details, classification criteria, and results can be found at <https://www.egr.msu.edu/coinlab/blankjul/emo19-thief/>, where our submission is identified as “*jomar*” (a reference to the two authors (**J**onatas and **M**arcone) who first worked on our algorithm). We herewith also present later a brief summary of these results.

After the *EMO-2019* competition, we have realized that the inclusion of more individuals created from TSP and KP solvers helped the evolutionary process of our algorithm by combining more individuals with higher fitness from the first evolutionary cycles. Therefore, we initialize the population as shown in Algorithm 1. This new initial population initialization has been used in the *GECCO-2019* competition, another BI-TTP competition that has considered the same criteria and classification rules of the *EMO-2019* competition.

In the *GECCO-2019* competition, 13 teams have submitted their solutions. In this competition, NDS-BRKGA has won the second place in the final ranking. All detailed competition results can be found at <https://www.egr.msu.edu/coinlab/blankjul/gecco19-thief/>, where our submission is identified as “*jomar*” again.

Table 9 shows a summary of the final results of both BI-TTP competitions. For each instance, we list the hypervolume achieved by the five best approaches that have been submitted to each competition. Our results from back then are highlighted in bold.

The results of the *EMO-2019* competition show the difference between the hypervolume achieved by NDS-BRKGA and by the others on smaller instances has been less significant than on larger instances. Also it is worth mentioning that for the instances *pla33810_n169045* and *pla33810_n338090* the difference between the NDS-BRKGA and the second-best approach has been higher than 0.65 (65% of the total hypervolume). Regarding the results of NDS-BRKGA and other submissions, we can clearly see the improvement achieved, especially for larger instances, by considering the current form of generating the initial population. The results of the instances *fnl4461_n22300* and *fnl4461_n44600* have been significantly improved compared to results obtained with the preliminary version of the algorithm submitted to the *EMO-2019* competition.

The results of the competition *GECCO-2019* show that NDS-BRKGA was able to win the test instance *a280_n2790* and has reached the second place five times. For test instances based on *pla33810*, NDS-BRKGA was able to achieve one of the top five ranks (11 participants in total). After the *GECCO-2019* competition, we incorporated the exploitation phase as the most recent enhancement to our method, thus completing the NDS-BRKGA described in Section 4. In order to compare the performance of all versions, we plot the hypervolume reached by each version in each instance according to the criteria of the competitions (see Figure 8). It can be observed that including more

Table 9: Results of the BI-TTP competitions. The results obtained by NDS-BRKGGA were submitted by the team *jomar*.

Instance	EMO-2019		GECCO-2019	
	Approach	HV	Approach	HV
a280_n279	jomar	0.893290	HPI	0.898433
	ALLAOUI	0.835566	jomar	0.895567
	shisunzhang	0.823563	shisunzhang	0.886576
	rrg	0.754498	NTGA	0.883706
	CIRG_UP_KUNLE	0.000000	ALLAOUI	0.873484
a280_n1395	jomar	0.816607	HPI	0.825913
	shisunzhang	0.756445	jomar	0.821656
	rrg	0.684549	shisunzhang	0.820893
	ALLAOUI	0.581371	NTGA	0.811490
	CIRG_UP_KUNLE	0.000000	ALLAOUI	0.808998
a280_n2790	jomar	0.872649	jomar	0.887945
	shisunzhang	0.861102	HPI	0.887571
	rrg	0.704428	ALLAOUI	0.885144
	ALLAOUI	0.621785	NTGA	0.882562
	CIRG_UP_KUNLE	0.000000	shisunzhang	0.874371
fnl4461_n4460	jomar	0.794519	HPI	0.933901
	shisunzhang	0.719242	jomar	0.932661
	ALLAOUI	0.553804	NTGA	0.914043
	CIRG_UP_KUNLE	0.000000	ALLAOUI	0.889219
	OMEGA	0.000000	SSteam	0.854150
fnl4461_n22300	shisunzhang	0.670849	HPI	0.818938
	jomar	0.554188	jomar	0.814634
	ALLAOUI	0.139420	NTGA	0.803470
	CIRG_UP_KUNLE	0.000000	SSteam	0.781462
	OMEGA	0.000000	ALLAOUI	0.760480
fnl4461_n44600	shisunzhang	0.540072	HPI	0.882894
	jomar	0.534185	jomar	0.874688
	ALLAOUI	0.009693	SSteam	0.856863
	CIRG_UP_KUNLE	0.000000	shisunzhang	0.850339
	OMEGA	0.000000	NTGA	0.824830
pla33810_n33809	jomar	0.718148	HPI	0.927214
	shisunzhang	0.496913	NTGA	0.888680
	ALLAOUI	0.090569	ALLAOUI	0.873717
	CIRG_UP_KUNLE	0.000000	jomar	0.845149
	OMEGA	0.000000	SSteam	0.832557
pla33810_n169045	jomar	0.697086	HPI	0.818259
	shisunzhang	0.022390	SSteam	0.776638
	ALLAOUI	0.007377	NTGA	0.773589
	CIRG_UP_KUNLE	0.000000	ALLAOUI	0.769078
	OMEGA	0.000000	jomar	0.738509
pla33810_n338090	jomar	0.696987	HPI	0.876129
	shisunzhang	0.049182	SSteam	0.853805
	ALLAOUI	0.001853	jomar	0.853683
	CIRG_UP_KUNLE	0.000000	ALLAOUI	0.836965
	OMEGA	0.000000	NTGA	0.781286

individuals from good solutions of the individual BI-TTP components brought a significant improvement. However, we did not observe major improvement after incorporating the exploitation phase after 5 hours running time, except for the test instance *pla33810_n169045* in which the hypervolume increases around 4.2%. Nevertheless, we have noticed a significant faster convergence with the incorporation.

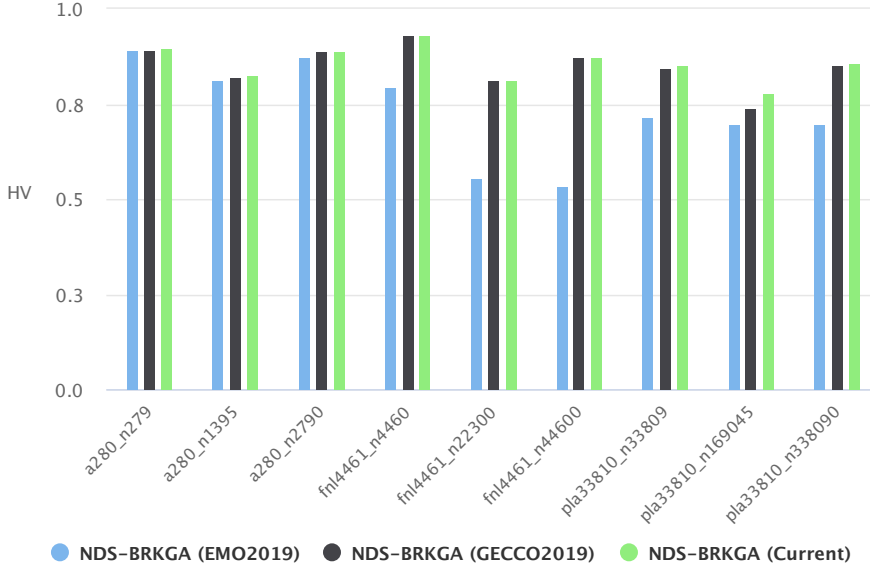


Fig. 8: Hypervolumes obtained by the different NDS-BRKGA versions.

Because we have improved our method after the *GECCO-2019* competition has passed, we have reevaluated the results based on the final version of NDS-BRKGA proposed in this paper. The results are shown in Table 10. In addition to the results, we present the hypervolume for each instance achieved by the four best approaches that have been submitted to both BI-TTP competitions. In the last column of the table, we list the difference between the hypervolume reached by each approach and that reached by the best one for each instance.

One can notice that NDS-BRKGA has outperformed all other approaches in three instances (*a280_n1395*, *a280_n2790*, and *fnl4461_n4460*). For the instances *a280_n279*, *fnl4461_n22300*, and *fnl4461_n44600*, NDS-BRKGA won the second place with a small difference to the first. For the three largest instances, NDS-BRKGA won the second place in two cases.

Table 10: Best result of the BI-TTP competitions *vs.* our final NDS-BRKGA.

Instance	Approach	HV	Diff.
a280_n279	HPI	0.898433	0.000000
	NDS-BRKGA	0.895708	0.002725
	shisunzhang	0.886576	0.011857
	NTGA	0.883706	0.014727
	ALLAOUI	0.873484	0.024949
a280_n1395	NDS-BRKGA	0.826879	0.000000
	HPI	0.825913	0.000966
	shisunzhang	0.820893	0.005986
	NTGA	0.811490	0.015389
	ALLAOUI	0.808998	0.017881
a280_n2790	NDS-BRKGA	0.887945	0.000000
	HPI	0.887571	0.000374
	ALLAOUI	0.885144	0.002801
	NTGA	0.882562	0.005383
	shisunzhang	0.874371	0.013574
fnl4461_n4460	NDS-BRKGA	0.933942	0.000000
	HPI	0.933901	0.000041
	NTGA	0.914043	0.019899
	ALLAOUI	0.889219	0.044723
	SSteam	0.854150	0.079792
fnl4461_n22300	HPI	0.818938	0.000000
	NDS-BRKGA	0.814492	0.004446
	NTGA	0.803470	0.015468
	SSteam	0.781462	0.037476
	ALLAOUI	0.760480	0.058458
fnl4461_n44600	HPI	0.882894	0.000000
	NDS-BRKGA	0.874688	0.008206
	SSteam	0.856863	0.026031
	shisunzhang	0.850339	0.032555
	NTGA	0.824830	0.058064
pla33810_n33809	HPI	0.927214	0.000000
	NTGA	0.888680	0.038534
	ALLAOUI	0.873717	0.053497
	NDS-BRKGA	0.852836	0.074378
	SSteam	0.832557	0.094657
pla33810_n169045	HPI	0.818259	0.000000
	NDS-BRKGA	0.781009	0.037250
	SSteam	0.776638	0.041621
	NTGA	0.773589	0.044670
	ALLAOUI	0.769078	0.049181
pla33810_n338090	HPI	0.876129	0.000000
	NDS-BRKGA	0.857105	0.019024
	SSteam	0.853805	0.022324
	ALLAOUI	0.836965	0.039164
	NTGA	0.781286	0.094843

5.4 Comparison with single-objective TTP solutions

Lastly, we build the bridge to the single-objective TTP which has been mostly investigated so far. Therefore, we compare our results with the single-objective TTP objective scores, which come from a comprehensive comparisons of a variety of algorithms reported in [Wagner et al. \(2018\)](#). The computational

Table 11: Single-objective comparison of the TTP objectives scores.

Instance	TTP score*	NDS-BRKG A
a280_n279	18470.000 ^a	18603.120
a280_n1395	110147.219 ^b	115445.521
a280_n2790	429081.783 ^c	429085.353
fnl4461_n4460	263040.254 ^d	257394.821
fnl4461_n22300	1705326.000 ^a	1567933.421
fnl4461_n44600	6744903.000 ^a	6272240.702
pla33810_n33809	1863667.592 ^d	1230174.003
pla33810_n169045	15634853.130 ^e	12935090.876
pla33810_n338090	58236645.120 ^f	55688288.508

* Available at <https://cs.adelaide.edu.au/~optlog/research/combinatorial.php>

^a Obtained from CS2SA approach proposed by El Yafrani and Ahiod (2016); ^b obtained from C3 approach; ^c obtained from C4 approach; ^d obtained from S5 approach; ^e obtained from S1 approach; ^f obtained from C6 approach. All these last approaches have been proposed by Faulkner et al. (2015).

budgets of the approaches which have obtained the best known solutions might vary.

Table 11 compares for each instance the best known score of the TTP with the best score found by our algorithm when it optimized the BI-TTP. Note that despite the strong connection of the BI-TTP to the single-objective TTP, maximizing the single-objective TTP objective score is not an explicit goal of the BI-TTP. Nevertheless, NDS-BRKG A has found better scores for the three smallest instances with 280 cities with up to 2790 items.

Figure 9 shows the 100% attainment surface for each instance. For each problem all non-dominated solutions found by our algorithm and the single-objective TTP solution obtained by the best performing algorithm (out of 10 runs) in Wagner et al. (2018) are plotted. Moreover, the dotted lines represent the dominated region of the solution obtain by a single-objective optimizer.

In Table 11, we compare the results in the single-objective TTP space. Two outstanding cases here are the instances *a280_n279* and *a280_n1395*: best single-objective solutions are strictly dominated by our bi-objective solutions in both the bi-objective space (Figure 9) and in the TTP score space (Table 11).

Figure 9 clearly shows that for almost all instances, none or only a few solutions are dominated (see the values in brackets in the figure). Not only this, but also the fact that not a single but multiple non-dominated solutions have been obtained shows the efficiency of our proposed approach. Only for the problem instance *pla33810_n33809* the single-objective optimizer was able to find significantly better results, where the single-objective solutions dominate 18.93% of the bi-objective solutions.

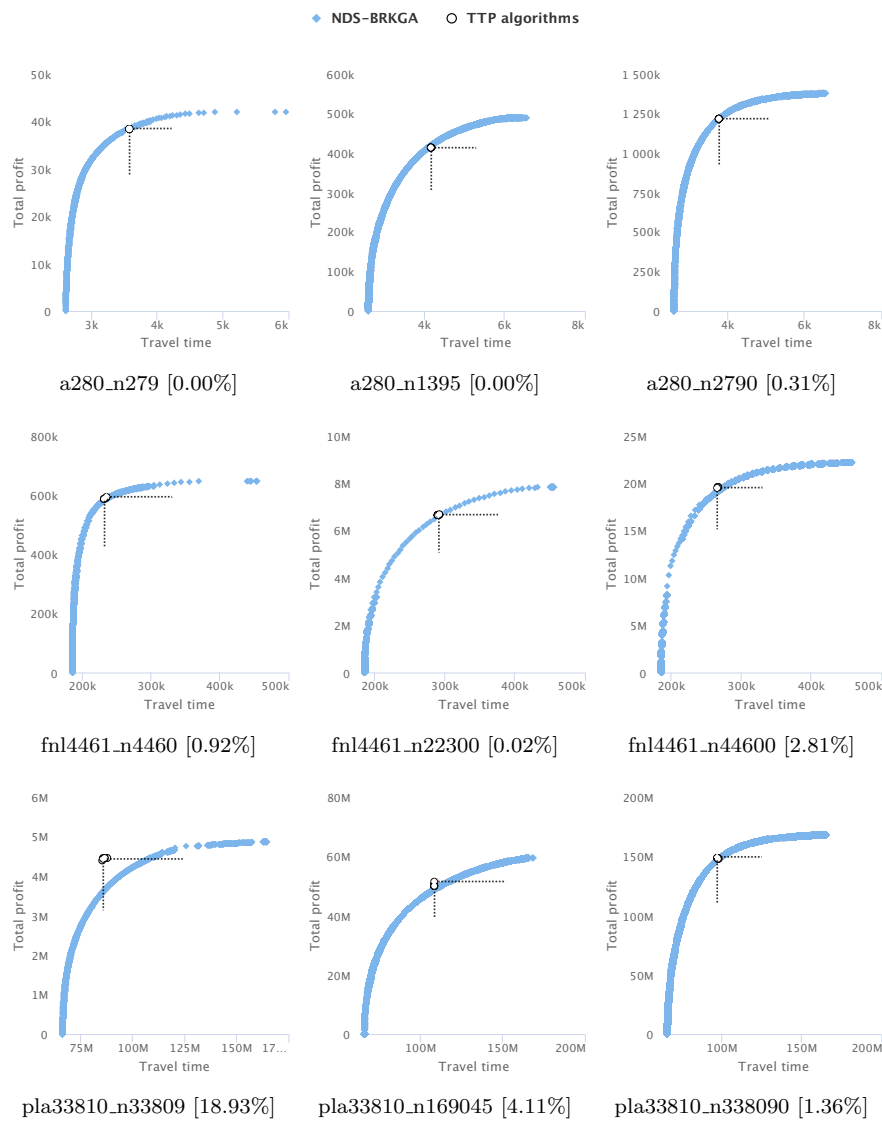


Fig. 9: NDS-BRKGA solutions for the present BI-TTP, and solutions generated by single-objective TTP algorithms (based on the best ones in Wagner et al. (2018)). The numbers in brackets show the percentage of the bi-objective solutions that are dominated by the single-objective solutions.

6 Concluding Remarks

In this paper, we have investigated the bi-objective traveling thief problem where the Traveling Salesman and Knapsack Problem interact with each other. We have proposed an evolutionary optimization algorithm that uses the principles of customization to effectively solve this challenging combinatorial optimization problem. Each customization addresses one specific problem characteristic that needs either to be considered during the optimization or can be used to further improve the convergence of the algorithm.

Our proposed method incorporates problem knowledge by creating a biased initial population that contains individuals generated by existing efficient solvers for each subproblem independently. Moreover, the constraint is handled through a customized repair operator during the evolution, and the heterogeneous variables are unified through a genotype to phenotype mapping. To address the existence of two objectives, we have used non-dominated sorting and crowding distance during the environment survival. To further improve the convergence, a local search is applied selectively during evolution.

Since those customizations come with various parameters, we have conducted an extensive experiment to show the effect of each parameter on the overall performance of the algorithm. Our results indicate that the best-performing configurations are those with larger population size, a higher survival rate for the best individuals, and a high contribution of the TSP and KP solvers for creating part of the initial population. The contribution of mutant individuals seems to be insignificant.

As a future study, a new way of initialization of the population is worth investigating. So far, we have used solutions obtained by subproblem solvers, but did not consider seeding it with good already known TTP solution approaches. Moreover, we are planning to efficiently incorporate the algorithmic insights gained from the single-objective approaches into the multi-objective setting; a naive bi-level approach is known to be computationally impractical. Lastly, it is worth investigating how the proposed concepts can be used for other optimization problems where two problems interact with each other. Since the customization of genetic algorithms is a powerful tool to design algorithms specifically for a given problem, the proposed concepts can be embedded into other algorithms in order to solve similar problems.

Acknowledgements The authors thank Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (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) for supporting this research. The authors would also like to thank the HPI Future SOC Lab and its team for enable this research by providing access to their compute infrastructure, <https://www.hpi.de/future-soc-lab>.

References

- Applegate DL, Bixby RE, Chvatal V, Cook WJ (2007) *The Traveling Salesman Problem: A Computational Study* (Princeton Series in Applied Mathematics). Princeton University Press, Princeton, NJ, USA
- Auger A, Bader J, Brockhoff D, Zitzler E (2009) Investigating and exploiting the bias of the weighted hypervolume to articulate user preferences. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, ACM, pp 563–570
- Blank J, Deb K, Mostaghim S (2017) *Solving the Bi-objective Traveling Thief Problem with Multi-objective Evolutionary Algorithms*, Springer, pp 46–60
- Bonyadi MR, Michalewicz Z, Barone L (2013) The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In: *2013 IEEE Congress on Evolutionary Computation*, IEEE, pp 1037–1044
- Bonyadi MR, Michalewicz Z, Przybyłek MR, Wierzbicki A (2014) Socially inspired algorithms for the TTP. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, ACM, pp 421–428
- Bonyadi MR, Michalewicz Z, Wagner M, Neumann F (2019) *Evolutionary Computation for Multicomponent Problems: Opportunities and Future Directions*, Springer, pp 13–30
- Bringmann K, Friedrich T (2013) Approximation quality of the hypervolume indicator. *Artificial Intelligence* 195:265 – 290
- Chand S, Wagner M (2015) Evolutionary many-objective optimization: A quick-start guide. *Surveys in Operations Research and Management Science* 20(2):35 – 42
- Chand S, Wagner M (2016) Fast heuristics for the multiple traveling thieves problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, ACM, pp 293–300
- Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation* 6(2):182–197
- El Yafrani M, Ahiod B (2016) Population-based vs. single-solution heuristics for the travelling thief problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, ACM, pp 317–324
- El Yafrani M, Ahiod B (2018) Efficiently solving the traveling thief problem using hill climbing and simulated annealing. *Information Sciences* 432:231–244
- El Yafrani M, Martins M, Wagner M, Ahiod B, Delgado M, Lüders R (2018) A hyperheuristic approach based on low-level heuristics for the travelling thief problem. *Genetic Programming and Evolvable Machines* 19(1):121–150
- Faulkner H, Polyakovskiy S, Schultz T, Wagner M (2015) Approximate approaches to the traveling thief problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, ACM, pp 385–392
- Gonçalves JF, Resende MG (2011) Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics* 17(5):487–525
- Gonçalves JF, Resende MG (2012) A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research* 39(2):179–190
- Gonçalves JF, Resende MG (2013) A biased random key genetic algorithm for 2d and 3d bin packing problems. *International Journal of Production Economics* 145(2):500–510
- Gonçalves JF, Resende MG (2015) A biased random-key genetic algorithm for the unequal area facility layout problem. *European Journal of Operational Research* 246(1):86–107
- Jung Sy (1999) *Multidisciplinary design optimization of aircraft wing structures with aeroelastic and aeroservoelastic constraints*. PhD thesis, School of Aerospace and Mechanical Engineering
- Klamroth K, Mostaghim S, Naujoks B, Poles S, Purshouse R, Rudolph G, Ruzika S, Sayın S, Wiecek MM, Yao X (2017a) Multiobjective optimization for interwoven systems. *Journal of Multi-Criteria Decision Analysis* 24(1-2):71–81
- Klamroth K, Mostaghim S, Naujoks B, Poles S, Purshouse R, Rudolph G, Ruzika S, Sayın S, Wiecek MM, Yao X (2017b) Multiobjective optimization for interwoven systems. *Journal of Multi-Criteria Decision Analysis* 24(1-2):71–81

- Krasnogor N, Smith J (2005) A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation* 9(5):474–488
- Lagoudakis MG (1996) The 0-1 knapsack problem – an introductory survey
- Lalla-Ruiz E, González-Velarde JL, Melián-Batista B, Moreno-Vega JM (2014) Biased random key genetic algorithm for the tactical berth allocation problem. *Applied Soft Computing* 22:60–76
- Lin S, Kernighan BW (1973) An effective heuristic algorithm for the traveling-salesman problem. *Operations research* 21(2):498–516
- Martins MSR, El Yafrani M, Delgado MRBS, Wagner M, Ahiod B, Lüders R (2017) Hseda: A heuristic selection approach based on estimation of distribution algorithm for the travelling thief problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference, Association for Computing Machinery, New York, NY, USA, GECCO '17*, p 361–368
- Mei Y, Li X, Yao X (2014) On investigation of interdependence between sub-problems of the TTP. *Soft Computing* 20(1):157–172
- Mitchell M (1998) *An introduction to genetic algorithms*. MIT press
- Namazi M, Sanderson C, Newton MAH, Sattar A (2019) A cooperative coordination solver for travelling thief problems. [1911.03124](https://doi.org/10.1145/33124)
- Neri F, Cotta C (2012) Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation* 2:1–14
- Neumann F, Polyakovskiy S, Skutella M, Stougie L, Wu J (2019) A fully polynomial time approximation scheme for packing while traveling. In: *Disser Y, Verykios VS (eds) Algorithmic Aspects of Cloud Computing*, Springer International Publishing, pp 59–72
- Polyakovskiy S, Bonyadi MR, Wagner M, Michalewicz Z, Neumann F (2014) A comprehensive benchmark set and heuristics for the traveling thief problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference, ACM, New York, NY, USA, GECCO '14*, pp 477–484
- Resende MG (2012) Biased random-key genetic algorithms with applications in telecommunications. *Top* 20(1):130–153
- Santos AG, Chagas JBC (2018) The thief orienteering problem: Formulation and heuristic approaches. In: *2018 IEEE Congress on Evolutionary Computation (CEC), IEEE, Rio de Janeiro, Brasil*, pp 1191–1199
- Toso RF, Resende MG (2015) A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software* 30(1):81–93
- Toth P (1980) Dynamic programming algorithms for the zero-one knapsack problem. *Computing* 25(1):29–45
- Wagner M (2016) Stealing items more efficiently with ants: A swarm intelligence approach to the travelling thief problem. In: *Dorigo M, Birattari M, Li X, López-Ibáñez M, Ohkura K, Pinciroli C, Stützle T (eds) Swarm Intelligence*, Springer, pp 273–281
- Wagner M, Bringmann K, Friedrich T, Neumann F (2015) Efficient optimization of many objectives by approximation-guided evolution. *European Journal of Operational Research* 243(2):465 – 479
- Wagner M, Lindauer M, Mısırlı M, Nallaperuma S, Hutter F (2018) A case study of algorithm selection for the traveling thief problem. *Journal of Heuristics* 24(3):295–320
- Wu J, Wagner M, Polyakovskiy S, Neumann F (2017) Exact approaches for the travelling thief problem. In: *Shi Y, Tan KC, Zhang M, Tang K, Li X, Zhang Q, Tan Y, Middendorf M, Jin Y (eds) Simulated Evolution and Learning*, Springer, pp 110–121
- Wu J, Polyakovskiy S, Wagner M, Neumann F (2018) Evolutionary computation plus dynamic programming for the bi-objective travelling thief problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference, ACM*, pp 777–784
- Wuijts RH, Thierens D (2019) Investigation of the traveling thief problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference, Association for Computing Machinery, New York, NY, USA, GECCO '19*, p 329–337
- Yafrani ME, Chand S, Neumann A, Ahiod B, Wagner M (2017) Multi-objectiveness in the single-objective traveling thief problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion, ACM, GECCO '17*, pp 107–108
- Yafrani ME, Martins MSR, Krari ME, Wagner M, Delgado MRBS, Ahiod B, Lüders R (2018) A fitness landscape analysis of the travelling thief problem. In: *Proceedings of*

-
- the Genetic and Evolutionary Computation Conference, Association for Computing Machinery, New York, NY, USA, GECCO '18, p 277–284
- Zitzler E, Thiele L (1998) Multiobjective optimization using evolutionary algorithms—a comparative case study. In: International conference on parallel problem solving from nature, Springer, pp 292–301
- Zouari W, Alaya I, Tagina M (2019) A new hybrid ant colony algorithms for the traveling thief problem. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, Association for Computing Machinery, New York, NY, USA, GECCO '19, p 95–96