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

Jonatas B. C. Chagas, Julian Blank, Markus Wagner, Marcone J. F. Souza and Kalyanmoy Deb

**COIN Report Number 2019008**

**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). In the TTP, a thief has to visit each city exactly once and can pick items throughout its journey. The thief begins its journey with an empty knapsack and travels with a speed inversely proportional to the knapsack weight. 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 method is based on a genetic algorithm with customization addressing problem characteristics. We incorporate domain knowledge through a combination of near-optimal solutions of each sub-problem in the initial population and a custom repair operation to avoid the evaluation of infeasible solutions. Moreover, the independent variables of the TSP and KP components are unified to a real variable representation by using a biased random-key approach. The bi-objective aspect of the problem is addressed through an elite population extracted based on the non-dominated rank and crowding distance of each solution. Furthermore, we provide a comprehensive study which shows the influence of hyperparameters on the performance of our method and investigate the performance of each hyperparameter combination over time. In addition to our experiments, we discuss the results of the BI-TTP competitions at *EMO-2019* and *GECCO-2019* conferences where our method has won first and second place, 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. For instance, in multidisciplinary design optimization various disciplines are linked with each other and have influence on 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 interwoveness modifying a single 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 investigated. 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 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 preciously, 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 nature.

In this paper, 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 algorithm 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

and maps a genotype to phenotype to deal with heterogeneous 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. Afterwards, 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, 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*, however, 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], memetic approaches [Mei et al., 2014], and swarm-intelligence based approaches [Wagner, 2016], to approaches with problem specific search operators [Faulkner et al., 2015]. Wagner et al. [2018] provide a comparison of 21 algorithms for the purpose of portfolio creation.

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. Interestingly, the "TTP2" from Bonyadi et al. [2013] is defined as a multi-objective problem right away. The objectives are the total time and the total value, because the value of items assumes to drop over time. Blank et al. [2017] investigated a variant of it, however, they decided to omit the value drop effect defined in the original TTP2.

When it comes to multi-objective approaches for the TTP1, only few works exist here as well. Yafrani et al. [2017] created an approach that generates diverse sets of TTP/TTP1 solutions, while being competitive with the state-of-the-art single-objective algorithms; the objectives were travel time and total profit of items. Wu et al. [2018] considered a bi-objective version of the TTP1; the objectives were 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.

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].

In this present paper, we consider the original TTP1, with the objectives being time and profit, hence using a setting comparable to that of Blank et al. [2017]. This setting was also used in the BI-TTP competitions at the conference conferences *EMO-2019*[1] and *GECCO-2019*[2]. This definition is equivalent to considering the TTP2 without a value-reduction of items over time. The proposed problem definition aimed to build the bridge from TTP1 to TTP2 by having two objectives but not adding another additional complexity to the problem.

# 3 Bi-objective Traveling Thief Problem

The TTP is a combinatorial optimization problem that consists of two interweaving problems, TSP and KP. In the following, first the two components, TSP and KP, 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 $i, j \in \{0, .., n\}$. The salesman has to visit each city once and the result is a permutation vector $\boldsymbol{\pi} = (\pi_1, \pi_2, ..., \pi_n)$, where $\pi_i$ 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:

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

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 where $d_{i,j} = d_{j,i}$.

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 $j \in \{1, .., m\}$. The binary decision vector $\boldsymbol{z} = (z_1, .., z_m)$ defines, if an item is picked or not. The aim is to maximize the profit $g(\boldsymbol{z})$:

[1] https://www.egr.msu.edu/coinlab/blankjul/emo19-thief/
[2] https://www.egr.msu.edu/coinlab/blankjul/gecco19-thief/

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

The search space of this problem is exponential with respect to $n$ 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.

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 is visiting. The items are stored in a rented knapsack carried by the thief. In more detail, each city $\pi_i$ 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 were visited so far, and summing up the weights of all picked items. The weight at city $i$ given $\boldsymbol{\pi}$ and $\boldsymbol{z}$ is calculated by:

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

The function $a_j(\pi_k)$ is defined for each item $j$ and returns 1 if the item could be stolen at city $\pi_k$ and 0 otherwise. The current weight of the knapsack has an influence on 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 not be negative for a feasible solution. Whenever the knapsack is heavier than the maximum weight $Q$, the capacity constraint is violated.

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

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:

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

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 the 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 $\boldsymbol{z}$ and $\boldsymbol{z}$ is part of the knapsack subproblem, it is challenging to solve the problem to optimality. In fact, such problems are called interwoven systems as the solution of one subproblem highly depends on the solution of the other subproblems.

Regarding the total profit of the items, this remains unchanged in the TTP from the original formulation in the KP problem.

Finally, the TTP problem is defined by

$$
\begin{aligned}
\min \quad & f(\boldsymbol{\pi}, \boldsymbol{z}) \\
\max \quad & g(\boldsymbol{z}) \\
f(\boldsymbol{\pi}, \boldsymbol{z}) &= \sum_{i=1}^{n-1} \frac{d_{\pi_i, \pi_{i+1}}}{v(W(i, \boldsymbol{\pi}, \boldsymbol{z}))} + \frac{d_{\pi_n, \pi_1}}{v(W(n, \boldsymbol{\pi}, \boldsymbol{z}))} \\
g(\boldsymbol{z}) &= \sum_{j=1}^{m} z_j \, b_j \\
\text{s.t.} \quad & \boldsymbol{\pi} = (\pi_1, \pi_2, ..., \pi_n) \in P_n \\
& \pi_1 = 1 \\
& \boldsymbol{z} = (z_1, .., z_m) \in \mathbb{B}^m \\
& \sum_{j=1}^{m} z_j \, w_j \leq Q
\end{aligned}
\tag{6}
$$

In order to illustrate the equations and interdependence, we present an example scenario here (see Figure 1). The thief starts at 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. He begins its tour at city 1 and picks no item there. For an empty knapsack $W(1, \boldsymbol{\pi}, \boldsymbol{z}) = 0$ 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

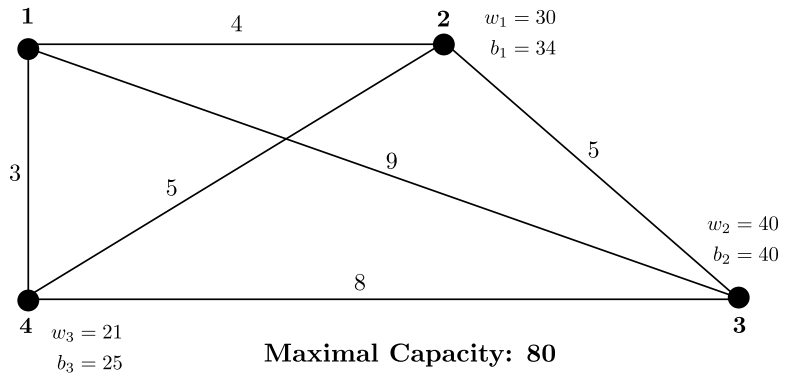



Figure 1: Exemplary traveling thief problem instance.

pick an item and continue to travel to city 2 with $W(2, \boldsymbol{\pi}, \boldsymbol{z}) = 0$ 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 $W(2, \boldsymbol{\pi}, \boldsymbol{z}) = 30$, which means the velocity will be reduced to $v(30) = 1.0 - (\frac{1.0-0.1}{80}) \cdot 30 = 0.6625$. 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 $W(4, \boldsymbol{\pi}, \boldsymbol{z}) = 30 + 21 = 51$. For this reason the velocity decreases to $v(51) = 1.0 - (\frac{1.0-0.1}{80}) \cdot 51 = 0.42625$. 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 | $\pi_i$ | $W(i, \boldsymbol{\pi}, \boldsymbol{z})$ | $v(W(i, \boldsymbol{\pi}, \boldsymbol{z}))$ | $d_{\pi_i, \pi_{i+1}}$ | $t_{\pi_i, \pi_{i+1}}$ | $\sum$ |
|---|---|---|---|---|---|---|
| 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 considering for the hand calculation is highlighted in bold.

Figure 2 shows the objective space colored by tour, where triangles denote non-dominated solutions, and circles denote dominated ones. We can observe that different Pareto-optimal solutions can have different underlying tours. In

Table 2: Pareto front of the example scenario.

| $\pi$ | $z$ | $f(\pi, z)$ | $g(z)$ |
|---|---|---|---|
| [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 |

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.

# 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 itself 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-TPP:

(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.

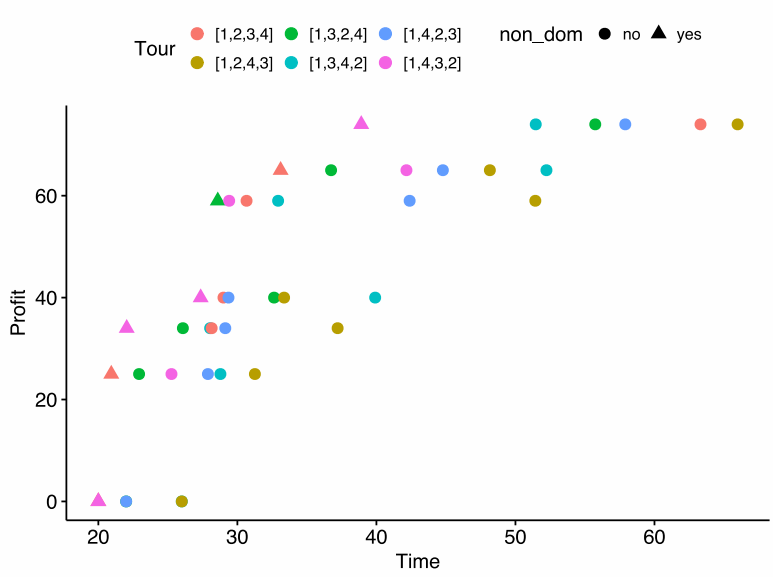



Figure 2: Exemplary traveling thief problem instance

(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 $P_{\bar{e}}^{(t)}$. 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.

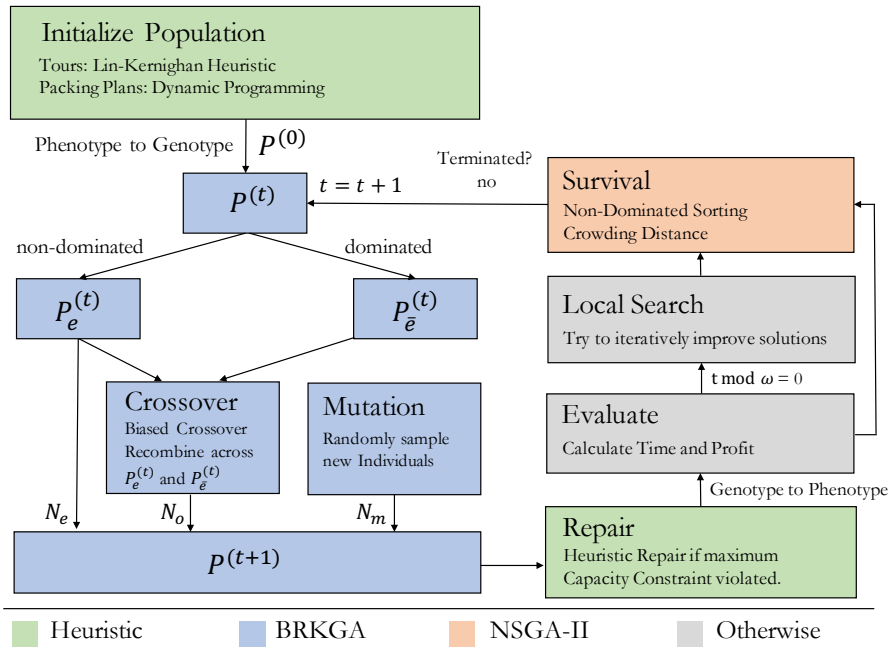



Figure 3: NDS-BRKGA: A customized genetic algorithm.

**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 in 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 at 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 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. A 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 this vector to the first city the tour is $\boldsymbol{\pi} = [1, 3, 2, 4]$. 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 the same phenotype can be represented by different genotypes.

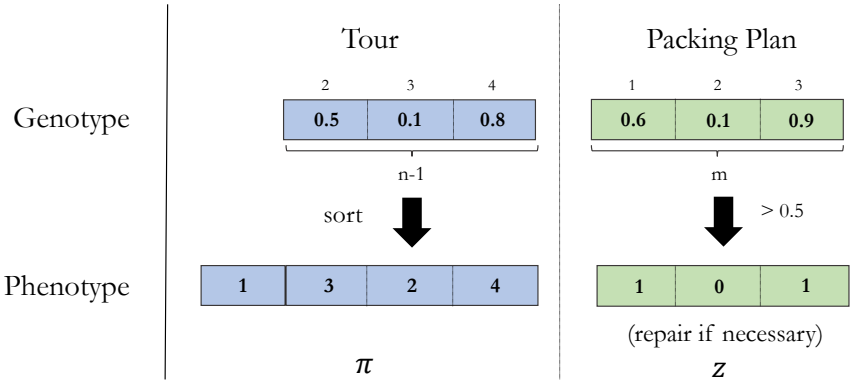



Figure 4: Chromosome structure: genotype to phenotype.

**Repair operator**

According to the decoding procedure previous described, a genotype can generate an infeasible phenotype concerning the packing plan. This 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 assign a real number less than 0.5 to every random-key that references an item that has not been collected.

**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], version 2.0.9[3], for solving the TSP component (Line 1). We consider the symmetrical tour found by LHK (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 hyperparameter of our

[3]Available at http://akira.ruc.dk/~keld/research/LKH/

**Algorithm 1:** Generate initial population

$\boldsymbol{\pi}' \leftarrow$ solve the TSP component `// LKH algorithm`
$\boldsymbol{\pi}'' \leftarrow \{ \pi_1, \pi_n, \pi_{n-1}, ..., \pi_2 \}$ `// symmetric` $\boldsymbol{\pi}'$
$\boldsymbol{z}' \leftarrow$ solve the KP component `// GH + DP algorithm`
$\boldsymbol{z}'' \leftarrow \varnothing$
$\mathcal{S} \leftarrow \{ \zeta(\boldsymbol{\pi}', \boldsymbol{z}''),\ \zeta(\boldsymbol{\pi}'', \boldsymbol{z}'')\}$
**repeat**
  $i \leftarrow$ select item $i \in z'$ with the highest $p_i/w_i$ rate
  $\boldsymbol{z}'' \leftarrow \boldsymbol{z}'' \cup \{ i \}$
  $\boldsymbol{z}' \leftarrow \boldsymbol{z}' \setminus \{ i \}$
  **if** $(\boldsymbol{\pi}', \boldsymbol{z}'') \prec (\boldsymbol{\pi}'', \boldsymbol{z}'')$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{ \zeta(\boldsymbol{\pi}', \boldsymbol{z}'')\}$
  **else**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{ \zeta(\boldsymbol{\pi}'', \boldsymbol{z}'')\}$
  **end**
**until** $z' = \varnothing$
$\mathcal{A} \leftarrow$ select randomly a set of $\alpha \times N$ individuals from $\mathcal{S}$ using a uniform distribution
$\mathcal{B} \leftarrow$ generate a set of $(1 - \alpha) \times N$ random individuals
$\mathcal{P}^{(0)} \leftarrow \mathcal{A} \cup \mathcal{B}$
**return** $\mathcal{P}^{(0)}$

$\zeta(\boldsymbol{\pi}, \boldsymbol{z})$ denotes a method that encodes the BI-TTP solution $(\boldsymbol{\pi}, \boldsymbol{z})$ to a vector of random-keys.

method. Next, it uses the classic dynamic programming algorithm [Toth, 1980] for solving the smaller KP considering the last $m - m'$ 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, we observed that we can generate BI-TTP solutions very close to the Pareto front regarding the profit objective by combining near-optimal KP packing plans with efficient TSP tours for collecting them.

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 $\alpha \times N$ ($\alpha$ 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. 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 $F_1, F_2, \ldots, F_L$, 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 $N_e$ are 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 $P_{\bar{e}}^{(t)}$.

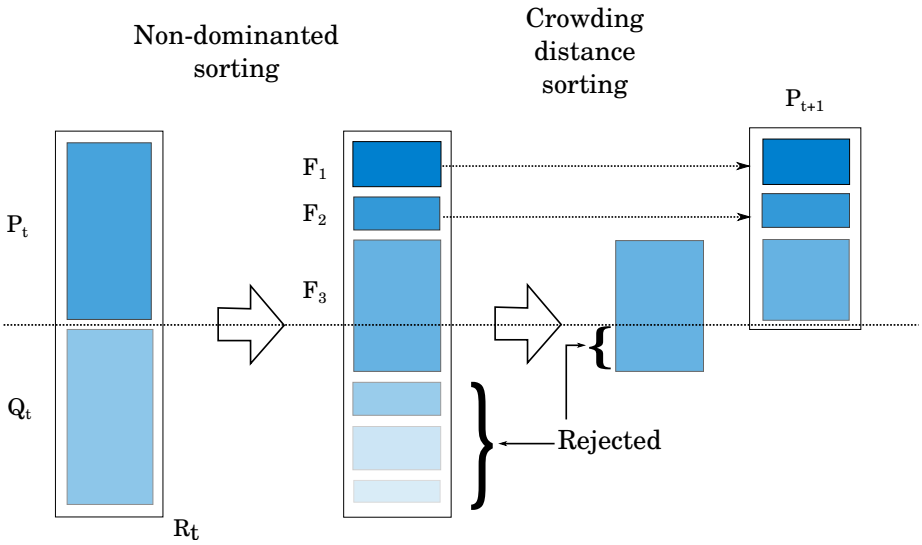



Figure 5: NSGA-II survival selection (Illustration based on Deb et al. [2002]).

### Biased crossover

For the purpose of diversity preservation, we apply a biased crossover in order to create new offspring individuals. This 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 all population $P^{(t)}$. Moreover, the biased crossover operator has a parameter $\rho_e$ 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:

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

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 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 $\boldsymbol{\pi}$ is considered separately

and the permutation is modified; Second, it considers only the packing plan $\boldsymbol{z}$ 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 $\boldsymbol{\pi}$ and a packing plan $\boldsymbol{z}$ (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 $\boldsymbol{\pi}$. 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 $LS_{\boldsymbol{\pi}} = \min(100, n^2)$ since the exploitation phase may become computationally expensive when large instances are considered. After all 2-opt moves have been executed, the tour $\boldsymbol{\pi}$ and the packing plan $\boldsymbol{z}$ are added to the elite population $P_e$ if it is not dominated by any solution in $P_e$ (Line 8). Second, we intend to improve the packing plan $\boldsymbol{z}$ by applying bit-flip random moves (Line 9 to 13). 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 $(\pi', z)$ is insert into $P_e$ if it is not dominated by any solution in $P_e$ (Line 12). Finally, if $P_e$ contains more than $N_e$ solutions (Line 15), we select the best $N_e$ according to the their non-dominated rank and crowding distance (Line 16).

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

# 5 Computational Experiments

In this section, we present 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 2.0.9[4]. 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 online[5].

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

[4]Available at http://akira.ruc.dk/~keld/research/LKH/
[5]Available at https://github.com/jonatasbcchagas/nds-brkga_bi-ttp

**Algorithm 2:** Exploitation phase.

$\widehat{P_e} \leftarrow$ select randomly $0.1N_e$ individuals from current elite population $P_e$
**foreach** $p \in \widehat{P_e}$ **do**
  $(\pi, z) \leftarrow$ decode $p$ in a feasible BI-TTP solution
  **for** $i \leftarrow 1$ ***to*** $LS_\pi$ **do**
    $\pi' \leftarrow$ generate a random 2-opt move in $\pi$
    **if** $(\pi', z) \prec (\pi, z)$ **then** $(\pi, z) \leftarrow (\pi', z)$
  **end**
  **if** $(\pi, z) \prec P_e$ **then** $P_e \leftarrow P_e \cup \{\, \zeta(\pi, z)\}$
  **for** $i \leftarrow 1$ ***to*** $LS_z$ **do**
    $item \leftarrow$ select randomly an $item \in \{1, 2, \ldots, m\}$
    **if** $item \in z$ **then** $z' \leftarrow z \setminus \{item\}$ **else** $z' \leftarrow z \cup \{item\}$
    **if** $(\pi, z') \prec P_e$ **then** $P_e \leftarrow P_e \cup \{\, \zeta(\pi, z')\}$
  **end**
**end**
**if** $|P_e| > N_e$ **then**
  $P_e \leftarrow$ select $N_e$ individuals according to the NSGA-II criteria
**end**

$\zeta(\boldsymbol{\pi}, \boldsymbol{z})$ denotes a method that encodes the BI-TTP solution $(\boldsymbol{\pi}, \boldsymbol{z})$ to a vector of random-keys.

design decisions we have made during the algorithm development, we provide a detailed hyperparameter 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.

## 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*[6] and *GECCO-2019*[7] 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 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 at each city. For example, assuming a problem with 280 cities, this results in 2790 items in total (assuming

[6] https://www.egr.msu.edu/coinlab/blankjul/emo19-thief/
[7] https://www.egr.msu.edu/coinlab/blankjul/gecco19-thief/

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 |

the first city never has any items).

## 5.2 Hyperparameter Study

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

Our proposed method has eight hyperparameters 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 $\rho_e$, fraction $\alpha$ of the initial population created from TSP and KP solutions (see Algorithm 1), and the frequency $\omega$, 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 $\delta$ which is the number of different KP capacities that should be considered. To evaluate the influence of each hyperparameter we conduct several experiments.

In this hyperparameter study, we first investigate the effect of $t$ and $\delta$ on the performance. Both variables affect the initial population which 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,

Table 4: Hyperparameter overview.

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

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

| **TSP** | t in seconds | | | | | | |
|---|---|---|---|---|---|---|---|
| comp. | 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% |

we show the relative percentage difference between the solution obtained with limited time and the TSP optimal solution.

Table 5 shows that even for shorter computational times LKH is efficient. On average, LKH has been able to find solutions with a gap 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.[8]

Moreover, the parameter $\delta$ 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 $\delta$ 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 $\delta$'s. In addition to the difference, it provides the computational time required in seconds $t(s)$. It can be observed that for almost all instances GH+DP was able to find the KP optimal solution even with a relatively small $\delta$. Moreover, the larger $\delta$, the larger

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

Table 6: Influence of $\delta$ 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 7: Parameter values considered during the experiment.

| **Parameter** | **Values** |
|---|---|
| $N$ | 100, 200, 500, 1000 |
| $N_e$ | $0.3N$, $0.4N$, $0.5N$ |
| $N_m$ | $0.0N$, $0.1N$, $0.2N$ |
| $\rho_e$ | 0.6, 0.7, 0.8 |
| $\alpha$ | 0.0, 0.1, 0.2 |
| $\omega$ | 10, 50, 100 |

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 on each of the subproblems without considering the interwovenness aspect. Next, the parameter $\alpha$ 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 with predefined parameter settings. The considered parameter values for each parameter are shown in Table 7. We have considered all 972 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 $437,400$ CPU hours which is equivalent to almost 50 CPU years.

We use the hypervolume indicator (HV) [Zitzler and Thiele, 1998] as a performance indicator to compare and analyze results obtained from 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, thereby the higher the hypervolume indicator the better the solution sets are (in general terms). To make hypervolume suitable for comparison in the objective values need to be normalized beforehand. Therefore, have first normalized the values of the objectives between 0 and 1 according to their boundaries before computing the hypervolume.

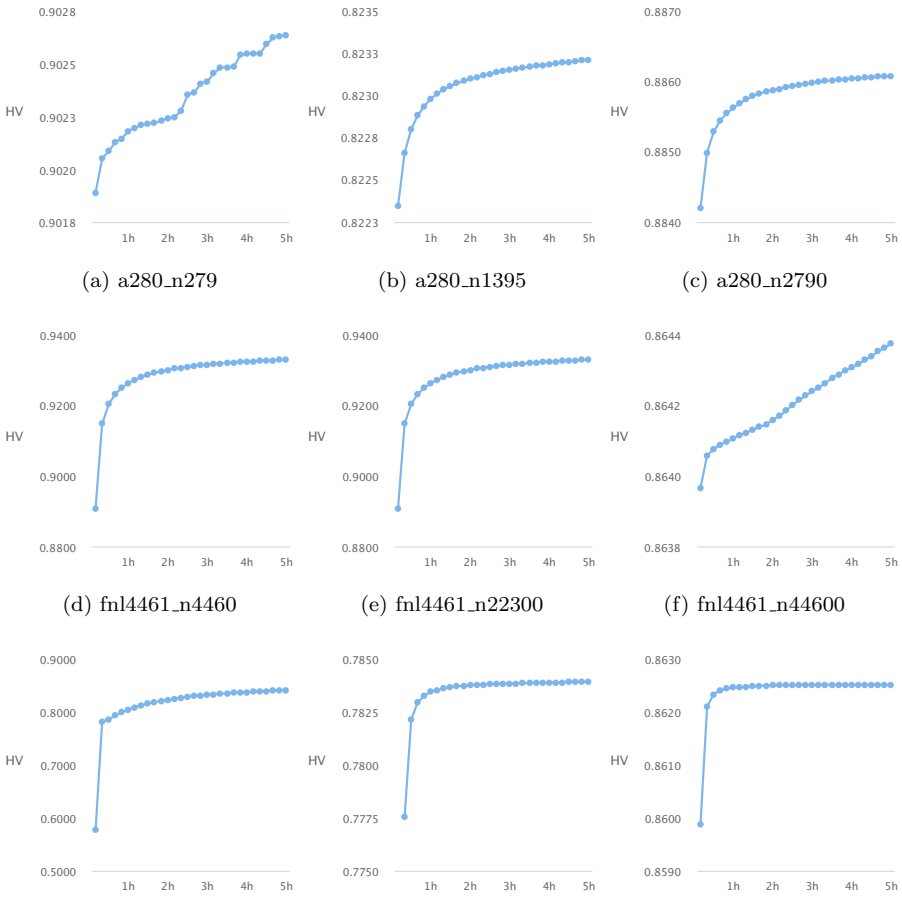
Figure 6 shows the convergence according to the hypervolume indicator for each instance throughout 5 hours, considering 10-minute intervals. For each interval, we have plotted the result of the best parameter configuration at each time interval. Each parameter configuration is described in Table 8. It 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 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 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 importance of values of each parameter among the best parameter configurations is indicated by the intensity of the blue color 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 hyperparameters were not set appropriately, the algorithm is still able to converge.

(ii) **Importance of TSP and KP solvers:** It has influence on the overall performance of the algorithm if TSP and KP solvers are used for ini-




(a) a280_n279 (b) a280_n1395 (c) a280_n2790

(d) fnl4461_n4460 (e) fnl4461_n22300 (f) fnl4461_n44600

(g) pla33810_n33809 (h) pla33810_n169045 (i) pla33810_n338090

Figure 6: Convergence plots according to the hypervolume indicator.

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

| Instance | Runtime | Best parameter configuration | | | | | |
|---|---|---|---|---|---|---|---|
| | | $N$ | $N_e$ | $N_m$ | $\rho_e$ | $\alpha$ | $\omega$ |
| a280_n279 | 600 | 500 | 0.4 | 0.1 | 0.7 | 0.1 | 10 |
| | 1200 - 1800 | 500 | 0.4 | 0.1 | 0.7 | 0.2 | 10 |
| | 2400 - 3000 | 500 | 0.3 | 0.1 | 0.6 | 0.2 | 10 |
| | 3600 - 6000 | 1000 | 0.3 | 0.1 | 0.8 | 0.2 | 10 |
| | 6600 - 7800 | 1000 | 0.3 | 0.2 | 0.8 | 0.1 | 10 |
| | 8400 | 500 | 0.4 | 0.1 | 0.8 | 0.1 | 100 |
| | 9000 - 9600 | 1000 | 0.3 | 0.2 | 0.8 | 0.1 | 10 |
| | 10200 - 18000 | 500 | 0.3 | 0.1 | 0.8 | 0.2 | 50 |
| a280_n1395 | 600 - 1200 | 1000 | 0.5 | 0.0 | 0.7 | 0.1 | 10 |
| | 1800 | 1000 | 0.5 | 0.0 | 0.6 | 0.1 | 10 |
| | 2400 - 4800 | 1000 | 0.5 | 0.0 | 0.7 | 0.1 | 50 |
| | 5400 - 16200 | 1000 | 0.5 | 0.0 | 0.6 | 0.1 | 50 |
| | 16800 - 18000 | 1000 | 0.5 | 0.0 | 0.6 | 0.1 | 100 |
| a280_n2790 | 600 | 1000 | 0.5 | 0.0 | 0.7 | 0.1 | 100 |
| | 1200 - 1800 | 1000 | 0.5 | 0.0 | 0.6 | 0.1 | 50 |
| | 2400 - 3000 | 1000 | 0.5 | 0.0 | 0.7 | 0.1 | 50 |
| | 3600 - 18000 | 1000 | 0.5 | 0.0 | 0.6 | 0.2 | 50 |
| fnl4461_n4460 | 600 | 500 | 0.5 | 0.0 | 0.7 | 0.2 | 50 |
| | 1200 - 18000 | 1000 | 0.5 | 0.0 | 0.6 | 0.2 | 50 |
| fnl4461_n22300 | 600 | 1000 | 0.5 | 0.0 | 0.8 | 0.1 | 10 |
| | 1200 - 4800 | 1000 | 0.5 | 0.0 | 0.6 | 0.2 | 100 |
| | 5400 - 10800 | 1000 | 0.5 | 0.0 | 0.6 | 0.1 | 100 |
| | 11400 - 18000 | 1000 | 0.5 | 0.0 | 0.6 | 0.2 | 50 |
| fnl4461_n44600 | 600 | 1000 | 0.5 | 0.0 | 0.8 | 0.2 | 100 |
| | 1200 | 1000 | 0.5 | 0.0 | 0.7 | 0.2 | 100 |
| | 1800 - 3000 | 1000 | 0.5 | 0.1 | 0.7 | 0.2 | 100 |
| | 3600 - 7800 | 1000 | 0.5 | 0.0 | 0.6 | 0.2 | 100 |
| | 8400 - 15000 | 1000 | 0.5 | 0.1 | 0.6 | 0.2 | 100 |
| | 15600 - 18000 | 1000 | 0.5 | 0.0 | 0.6 | 0.2 | 50 |
| pla33810_n33809 | 600 | 500 | 0.4 | 0.2 | 0.7 | 0.1 | 10 |
| | 1200 | 500 | 0.5 | 0.1 | 0.8 | 0.2 | 10 |
| | 1800 | 1000 | 0.4 | 0.0 | 0.7 | 0.1 | 10 |
| | 2400 - 18000 | 1000 | 0.5 | 0.0 | 0.6 | 0.2 | 10 |
| pla33810_n169045 | 600 | - | - | - | - | - | - |
| | 1200 | 100 | 0.3 | 0.0 | 0.7 | 0.1 | 10 |
| | 1800 | 500 | 0.3 | 0.0 | 0.7 | 0.1 | 10 |
| | 2400 - 3000 | 500 | 0.5 | 0.2 | 0.8 | 0.2 | 10 |
| | 3600 | 1000 | 0.4 | 0.0 | 0.7 | 0.2 | 10 |
| | 4200 | 1000 | 0.4 | 0.0 | 0.8 | 0.1 | 10 |
| | 4800 - 5400 | 1000 | 0.5 | 0.0 | 0.6 | 0.1 | 10 |
| | 6000 - 10200 | 1000 | 0.5 | 0.0 | 0.8 | 0.1 | 10 |
| | 10800 - 18000 | 1000 | 0.5 | 0.2 | 0.8 | 0.2 | 10 |
| pla33810_n338090 | 600 | 1000 | 0.5 | 0.2 | 0.8 | 0.2 | 100 |
| | 1200 | 1000 | 0.5 | 0.0 | 0.8 | 0.2 | 50 |
| | 1800 - 4800 | 1000 | 0.5 | 0.0 | 0.8 | 0.2 | 100 |
| | 5400 - 18000 | 1000 | 0.5 | 0.2 | 0.8 | 0.2 | 100 |

tialization which is determined by $\alpha$. The best results are obtained if at least 10% or 20% 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, an minor influence of a good initial population, and significant influence of local search procedure.

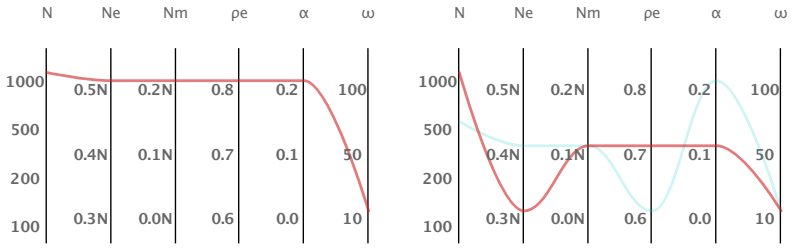
### 5.3 Competition Results

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. Following the criteria of both competitions, we 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 $\mathcal{O}(|A|^3)$.

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 $(\pi, \varnothing)$, $(\pi', \varnothing)$, $(\pi, z)$, and $(\pi', z)$, where $\pi$ is the tour found by LKH algorithm, $\pi'$ is the symmetric tour to $\pi$, and $z$ is the packing plan for the knapsack obtained by GH+DP algorithm.

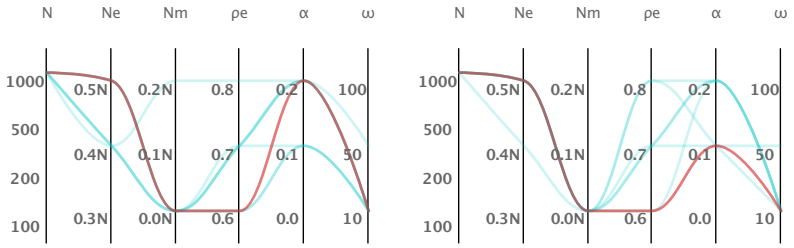
In addition to our method, five other teams also submitted their solutions to the *EMO-2019* competition. Among all submissions, NDS-BRKGA has had the best performance in seven of the nine test instances, resulting in the first-place award. All competition details, classification criteria, and results can be found at `https://www.egr.msu.edu/coinlab/blankjul/emo19-thief/`, where our




(a) 600 seconds



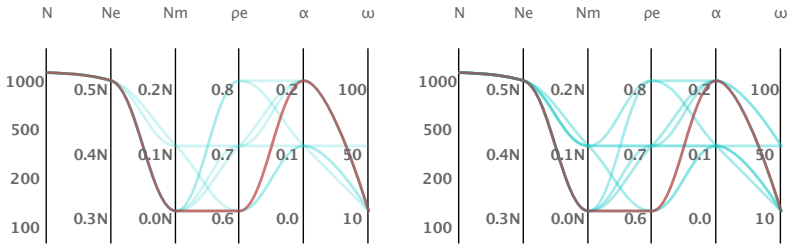

(b) 1200 seconds




(c) 1800 seconds




(d) 3600 seconds




(e) 7200 seconds




(f) 18000 seconds

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

submission is identified as *"jomar"* (a reference to the two authors (**Jo**natas and **Mar**cone) 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 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 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.

Because we have improved our method after the *GECCO-2019* competition

Table 9: Results of the BI-TTP competitions. The results obtained by NDS-BRKGA 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 |

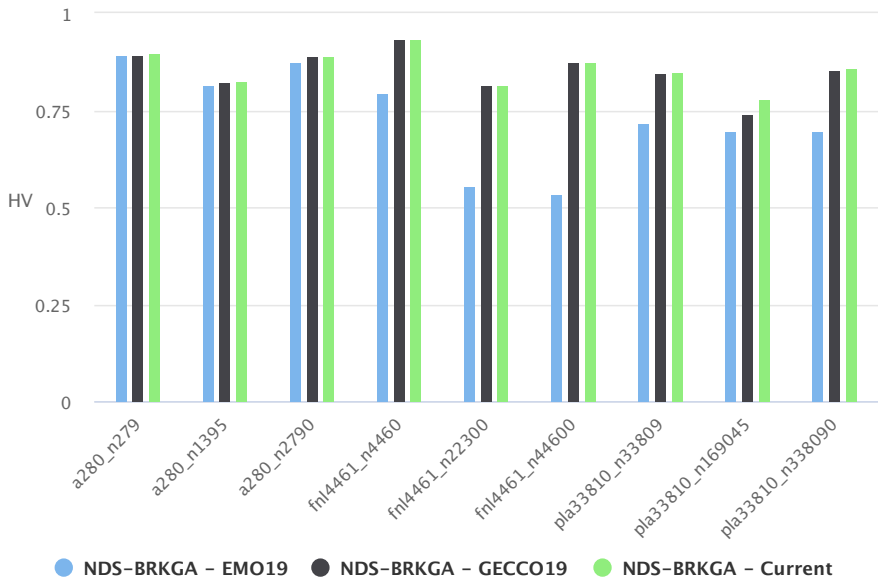



Figure 8: Hypervolumes obtained by the different NDS-BRKGA versions.

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 two instances (*a280_n1395* and *a280_n2790*). For the instances *a280_n279*, *fnl4461_n4460*, *fnl4461_n22300*, and *fnl4461_n44600*, NDS-BRKGA won the second place with a small difference to the first (average difference between hypervolumes equal to 0.002970). For the three largest instances NDS-BRKGA won the second place.

## 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 best known single-objective TTP objective scores, which come from a comprehensive comparisons of a variety of algorithms. The computational 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. NDS-BRKGA has found found better scores for the three smallest instances with 280 cities with up to 2790 items and

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** | **0.826877** | **0.000000** |
| | HPI | 0.825913 | 0.000964 |
| | shisunzhang | 0.820893 | 0.005984 |
| | NTGA | 0.811490 | 0.015387 |
| | ALLAOUI | 0.808998 | 0.017879 |
| 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 | HPI | 0.933901 | 0.000000 |
| | **NDS-BRKGA** | **0.933639** | **0.000262** |
| | NTGA | 0.914043 | 0.019858 |
| | ALLAOUI | 0.889219 | 0.044682 |
| | SSteam | 0.854150 | 0.079751 |
| 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** | **0.849008** | **0.078206** |
| | SSteam | 0.832557 | 0.094657 |
| pla33810_n169045 | HPI | 0.818259 | 0.000000 |
| | **NDS-BRKGA** | **0.780728** | **0.037531** |
| | SSteam | 0.776638 | 0.041621 |
| | NTGA | 0.773589 | 0.044670 |
| | ALLAOUI | 0.769078 | 0.049181 |
| pla33810_n338090 | HPI | 0.876129 | 0.000000 |
| | **NDS-BRKGA** | **0.857054** | **0.019075** |
| | SSteam | 0.853805 | 0.022324 |
| | ALLAOUI | 0.836965 | 0.039164 |
| | NTGA | 0.781286 | 0.094843 |

Table 11: Single-objective comparison of the TTP objectives scores: best known solution (BKS) *vs.* NDS-BRKGA. BKS data is available at https://cs.adelaide.edu.au/~optlog/research/combinatorial.php. In the BI-TTP formulation, the TTP objective score is not an explicitly stated optimization goal.

| **Instance** | **BKS** | **NDS-BRKGA** |
|---|---|---|
| a280_n279 | 18470.000 | 18603.120 |
| a280_n1395 | 110147.219 | 115445.521 |
| a280_n2790 | 429081.783 | 429085.353 |
| fnl4461_n4460 | 263040.254 | 256402.882 |
| fnl4461_n22300 | 1705326.000 | 1567933.421 |
| fnl4461_n44600 | 6744903.000 | 6272240.702 |
| pla33810_n33809 | 1863667.592 | 1230174.003 |
| pla33810_n169045 | 15634853.130 | 12913836.427 |
| pla33810_n338090 | 58236645.120 | 55678603.771 |

provides competitive results for the other test instances.

# 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 hyperparameters, 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 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 solutions. Moreover, 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.

## Acknowledgement

The authors thank Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), 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.

## References

D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007. ISBN 0691129932, 9780691129938.

A. Auger, J. Bader, D. Brockhoff, and E. Zitzler. Investigating and exploiting the bias of the weighted hypervolume to articulate user preferences. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 563–570. ACM, 2009.

J. Blank, K. Deb, and S. Mostaghim. *Solving the Bi-objective Traveling Thief Problem with Multi-objective Evolutionary Algorithms*, pages 46–60. Springer, 2017.

M. R. Bonyadi, Z. Michalewicz, and L. Barone. The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In *2013 IEEE Congress on Evolutionary Computation*, pages 1037–1044. IEEE, 2013.

M. R. Bonyadi, Z. Michalewicz, M. R. Przybylek, and A. Wierzbicki. Socially inspired algorithms for the TTP. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 421–428. ACM, 2014.

M. R. Bonyadi, Z. Michalewicz, M. Wagner, and F. Neumann. *Evolutionary Computation for Multicomponent Problems: Opportunities and Future Directions*, pages 13–30. Springer, 2019.

S. Chand and M. Wagner. Fast heuristics for the multiple traveling thieves problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 293–300. ACM, 2016.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

M. El Yafrani and B. Ahiod. Population-based vs. single-solution heuristics for the travelling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 317–324. ACM, 2016.

M. El Yafrani and B. Ahiod. Efficiently solving the traveling thief problem using hill climbing and simulated annealing. *Information Sciences*, 432:231–244, 2018.

H. Faulkner, S. Polyakovskiy, T. Schultz, and M. Wagner. Approximate approaches to the traveling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 385–392. ACM, 2015.

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

J. F. Gonçalves and M. G. Resende. A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research*, 39(2):179–190, 2012.

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

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

S.-y. Jung. *Multidisciplinary design optimization of aircraft wing structures with aeroelastic and aeroservoelastic constraints*. PhD thesis, School of Aerospace and Mechanical Engineering, 1999.

K. Klamroth, S. Mostaghim, B. Naujoks, S. Poles, R. Purshouse, G. Rudolph, S. Ruzika, S. Sayın, M. M. Wiecek, and X. Yao. Multiobjective optimization for interwoven systems. *Journal of Multi-Criteria Decision Analysis*, 24(1-2): 71–81, 2017a.

K. Klamroth, S. Mostaghim, B. Naujoks, S. Poles, R. Purshouse, G. Rudolph, S. Ruzika, S. Sayın, M. M. Wiecek, and X. Yao. Multiobjective optimization for interwoven systems. *Journal of Multi-Criteria Decision Analysis*, 24(1-2): 71–81, 2017b.

N. Krasnogor and J. Smith. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.

M. G. Lagoudakis. The 0-1 knapsack problem – an introductory survey, 1996.

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

S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

Y. Mei, X. Li, and X. Yao. On investigation of interdependence between sub-problems of the TTP. *Soft Computing*, 20(1):157–172, 2014.

M. Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

F. Neri and C. Cotta. Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, 2:1–14, 2012.

F. Neumann, S. Polyakovskiy, M. Skutella, L. Stougie, and J. Wu. A fully polynomial time approximation scheme for packing while traveling. In Y. Disser and V. S. Verykios, editors, *Algorithmic Aspects of Cloud Computing*, pages 59–72. Springer International Publishing, 2019. ISBN 978-3-030-19759-9.

S. Polyakovskiy, M. R. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann. A comprehensive benchmark set and heuristics for the traveling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '14, pages 477–484, New York, NY, USA, 2014. ACM.

M. G. Resende. Biased random-key genetic algorithms with applications in telecommunications. *Top*, 20(1):130–153, 2012.

A. G. Santos and J. B. C. Chagas. The thief orienteering problem: Formulation and heuristic approaches. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1191–1199, Rio de Janeiro, Brasil, 2018. IEEE.

R. F. Toso and M. G. Resende. A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, 30(1):81–93, 2015.

P. Toth. Dynamic programming algorithms for the zero-one knapsack problem. *Computing*, 25(1):29–45, 1980.

M. Wagner. Stealing items more efficiently with ants: A swarm intelligence approach to the travelling thief problem. In M. Dorigo, M. Birattari, X. Li, M. López-Ibáñez, K. Ohkura, C. Pinciroli, and T. Stützle, editors, *Swarm Intelligence*, pages 273–281. Springer, 2016.

M. Wagner, M. Lindauer, M. Mısır, S. Nallaperuma, and F. Hutter. A case study of algorithm selection for the traveling thief problem. *Journal of Heuristics*, 24(3):295–320, Jun 2018. ISSN 1572-9397.

J. Wu, M. Wagner, S. Polyakovskiy, and F. Neumann. Exact approaches for the travelling thief problem. In Y. Shi, K. C. Tan, M. Zhang, K. Tang, X. Li, Q. Zhang, Y. Tan, M. Middendorf, and Y. Jin, editors, *Simulated Evolution and Learning*, pages 110–121. Springer, 2017. ISBN 978-3-319-68759-9.

J. Wu, S. Polyakovskiy, M. Wagner, and F. Neumann. Evolutionary computation plus dynamic programming for the bi-objective travelling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 777–784. ACM, 2018.

M. E. Yafrani, S. Chand, A. Neumann, B. Ahiod, and M. Wagner. Multi-objectiveness in the single-objective traveling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, pages 107–108. ACM, 2017. ISBN 978-1-4503-4939-0.

E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms—a comparative case study. In *International conference on parallel problem solving from nature*, pages 292–301. Springer, 1998.