applied sciences

MDPI

Article

A Sequence-Based Hyper-Heuristic for Traveling Thieves

Daniel Rodríguez, Jorge M. Cruz-Duarte, José Carlos Ortiz-Bayliss and Ivan Amaya *

School of Engineering and Sciences, Tecnologico de Monterrey, Monterrey 64849, Mexico
* Correspondence: iamaya2@tec.mx

Abstract: A plethora of combinatorial optimization problems can be linked to real-life decision scenarios. Even nowadays, more diverse and complex problems are popping up. One of these problems is the traveling thief problem (TTP), which combines elements from the knapsack and traveling salesperson problems. Hence, it is paramount to keep improving solvers to tackle combinatorial problems. Among recent proposals, hyper-heuristics have proven useful since they seek to combine the strengths of more straightforward solvers. This paper proposes a sequence-based selection hyper-heuristic and assesses its feasibility when solving the TTP. Our proposal can be represented by an array of operators selecting a city or an item. In the first case, the solution moves to a new city and thus advances the tour. In the second one, the *thief* agent picks an item within the current city and tries to store it in its knapsack. We generate several sets of TTP instances with different parameters to validate our approach and analyze the model's performance. Our data reveal that the proposed approach outperforms randomly generated sequences. Moreover, our approach finds general sequences that surpass sequences specialized for each instance. We believe this is noteworthy and represents a stepping stone towards achieving a more robust solver for complex problems.

Keywords: hyper-heuristic; traveling thief problem; combinatorial optimization

check for updates

Citation: Rodríguez, D.; Cruz-Duarte, J.M.; Ortiz-Bayliss, J.C.; Amaya, I. A Sequence-Based Hyper-Heuristic for Traveling Thieves. *Appl. Sci.* **2023**, *13*, 56. https://doi.org/10.3390/app13010056

Academic Editor: Vincent A. Cicirello

Received: 24 November 2022
Revised: 9 December 2022
Accepted: 9 December 2022
Published: 21 December 2022

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Real-world problems often involve the composition and interdependence of smaller problems. The traveling thief problem (TTP) is a recent example of such a synergy, which combines two well-known optimization problems [1]: the traveling salesperson problem (TSP) and the knapsack problem (KP). Both of them have been addressed extensively in many theoretical and practical studies. For example, Osaba et al. presented an extensive review about the TSP, focusing on their solution through several metaheuristics that include genetic and swarm-based algorithms [2]. Similarly, the KP has been explored by several authors [3,4].

In simple terms, one may describe the TTP as follows. A *thief* travels throughout a set of cities, carrying a knapsack and seeking to loot items from each city. This *thief* aims to maximize their earnings, so they must decide if an item is worth picking up in a city. Since each object has a given value and weight, the *thief* must identify the best picking plan and the best path throughout the cities. The primary constraint in the TTP is that the knapsack has a limited capacity. Furthermore, as the knapsack becomes heavier, the *thief* slows down. Finally, all cities but the starting one must be visited only once. The reason is that the starting city has no objects and represents the initial and final points in the tour.

The literature contains several versions of the TTP proposed by Bonyadi et al. [1]. One of them relates to the description, as mentioned earlier, of the TTP. The second version is given by a bi-objective optimization problem where the goal is maximizing traveling time and profit. It also includes a feature where the value of an item decays over time. Some authors have explored solutions to this variant throughout the years. For example, Blank et al. used multi-objective evolutionary algorithms [5], whilst Kumari et al. used variable neighbourhood search [6]. Similarly, Wu et al. merged a bi-objective evolutionary algorithm with dynamic programming [7].

In this work, we pursue the first version of the TTP, which diverse methods have already tackled. Examples of such methods include the one of Wu et al., where the authors explored the behavior of exact algorithms and dynamic programming [8]. Nonetheless, other authors have explored the use of evolutionary approaches for this version of the TTP. For example, Mei et al. analyzed the complexity of heuristics that carry out a local search and proposed strategies for speeding them up [9]. Additionally, they proposed a memetic algorithm based on a two-stage local search, which exhibited good results. Similarly, Polyakovskiy et al. presented a set of solvers for tackling the TTP, which included simple heuristics and a (1+1) evolutionary approach [10]. They used these approaches to tackle sets of instances they created based on the TSP library and the work of Martello et al. [11]. A couple of years later, Moeini et al. proposed a hybrid approach that combined genetic algorithms, local search, and variable neighborhood search [12].

One way of solving the TTP is first to decide on the tour and then adapt the picking plan [6]. However, Ali et al. recently proposed a strategy where the plan is prioritized, and the tour is solved as a secondary step [13]. The authors developed their model based on simulated annealing (SA) and targeted small and medium sequences. Although they achieved a good performance, their model discards cities where items are not picked up, representing a drawback.

Maity and Das proposed a method that constructs a picking plan through the chained Lin—Kernighan heuristic (CLKH). In their proposal, the items are picked up according to a score formulated by the authors [14]. Additionally, the proposal incorporates bit-flip to improve the picking plan. However, most of the TTP solvers deal with the KP and TSP components of the problems individually, keeping one of these components fixed while modifying the other one. Namazi et al. proposed an expanded form of the 2-Opt operator used for TSP to modify the collection plan simultaneously with the tour [15]. Additionally, Nikfarjam et al. introduced a bi-level evolutionary algorithm to maximize the structural diversity of the set of solutions within the TTP. Their work explored the inter-dependency among the KP and TSP components of the problem according to the structural diversity [16]. Finally, Nikfarjam et al. introduced a MAP-elite-based evolutionary algorithm that used operators for the KP and TSP to explore the KP/TSP behavioral space [17].

Another approach for solving optimization problems is known as hyper-heuristics (HHs) [18]. Although there are different kinds of HHs [19], we restrict our discussion to selection HHs since they represent the kind of solver we develop for this work. Selection HHs seek to improve performance by combining a set of available solvers, where simple heuristics are customary. This way, different solution steps can be carried out with different solvers, enhancing the diversity of solutions.

There are only a few works that incorporate hyper-heuristics for solving the TTP. One of these works was presented by El Yafrani et al. a few years ago [20]. In their work, the authors used genetic programming to develop a model that automatically selects the heuristics. A more recent proposal was presented by Ali et al. [21]. Here, the authors explored the feasibility of four models based on a single-point selection. In both cases, the authors achieved competitive results. Despite this, we are unaware of hyper-heuristic models based on sequences of operators that tackle the TTP. Thus, this work aims to fill such a knowledge gap. Our model evaluates a set of heuristics that simultaneously consider both sub-problems. Throughout the training phase of our approach, we employ simulated annealing (SA) to optimize the sequence of heuristics. Our goal is to achieve sequences that generalize the behavior of different kinds of instances. Hence, our work has the following contributions:

1. The proposal of a sequence-based hyper-heuristic model for the TTP;
2. A study about the generalization capabilities of the proposed model regarding two kinds of instances;
3. The development of an instance generator that allows the creation of TTP instances of different natures.

The rest of this manuscript is organized as follows. Section 2 provides an overview about the TTP and the sub-problems that compose it, as well as about hyper-heuristics. Then, we present our approach in Section 3. In Section 4, we lay out the experimental methodology that validates our approach, and the corresponding data rests within Section 5. Finally, Section 6 presents the significant conclusions and summarizes some paths for future work.

2. Fundamentals

In this section, we present some key concepts for our research. We discuss the optimization problem and the two kinds of solvers we use in this work: metaheuristics and hyper-heuristics. As we will discuss in Section 3, the former is used to train the latter.

2.1. Optimization Problem

For this work, we tackle a relatively recent optimization problem: the traveling thief problem (TTP). This problem is combinatorial and appears as the synergy of the knapsack and traveling salesperson problems (KP and TSP). Therefore, to understand the TTP, we must first talk about the KP and the TSP.

2.1.1. Knapsack Problem

The knapsack problem (KP) consists of packing certain items within a set so that the total profit is maximized while the total weight remains below a given limit. Hence, the KP can be formulated as:

$$\begin{aligned} &\max_{\vec{x}} \{\vec{p}^{\intercal}\vec{x}\} \\ &\text{s.t. } \vec{w}^{\intercal}\vec{x} \leq K \end{aligned} \tag{1}$$

where $\vec{p} \in \mathbb{R}^N$ and $\vec{w} \in \mathbb{R}_+^N$ refer to the profit and weight vectors, respectively, for the N number of available objects to be considered to place inside the knapsack with a weight capacity of K. Thus, the design vector for this problems is $\vec{x} \in \mathbb{Z}_2^N$, which only has binary elements representing whether an object is chosen; i.e., $\mathbb{Z}_2 = \{0, 1\}$.

2.1.2. Traveling Salesperson Problem

A traveling salesperson problem (TSP) is given by a set of cities that must be visited through the shortest path, thus reducing the cost of a tour across the cities. It is essential to highlight that a TSP instance requires each city to be visited only once. The only exception is the departing city, which also represents the final city of the solution. This implies that solutions to a TSP provide a closed loop around the cities (a tour), i.e., a Hamiltonian cycle. Keeping this in mind, the problem can be formulated as follows.

Let us define a weighted complete graph $\mathfrak{G} = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V}$ representing the set of nodes such that $\#\mathcal{V}$ equals the number of cities T, while $\mathcal{E}$ is the set of edges between each pair of nodes. (Recall that # stands for the cardinality of a set). Therefore, each city is related to $T - 1$ edges, and their values represent the cost of moving from the current city to the other ones. Thus, all movement costs related to distances can be associated with a square matrix of order T, $\boldsymbol{D} \in \mathbb{R}_+^{T \times T}$, where any element $d_{ij} \in \boldsymbol{D}$ provides the cost of moving from city i to city j. To fulfil the Hamiltonian cycle property, $\text{diag}(\boldsymbol{D}) = \vec{0}$ ensures that the salesperson returns to the first city in the end since $\text{diag}(\,)$ is the well-known diagonal function and $\vec{0}$ is a vector of t elements equal to zero. In this way, we can define a binary variable $y_{ij} \in \mathbb{Z}_2$, which comprises the information on whether the salesperson travels between these two cities. The symmetric binary matrix $\boldsymbol{Y} \in \mathbb{Z}_2^{T \times T}$ contains all the possible paths y_{ij} from one city to another. Since each travel represents a cost, and taking advantage of basic linear algebra, the traveling salesperson problem can be summarized as:

$$\begin{aligned} &\min_{\boldsymbol{Y}} \{\text{tr}(\boldsymbol{1}(\boldsymbol{D} \odot \boldsymbol{Y}))\}, \\ &\text{s.t. } \vec{1}^{\intercal}\boldsymbol{Y} = \vec{1}^{\intercal}, \\ &\quad \boldsymbol{Y}\vec{1} = \vec{1}, \end{aligned} \tag{2}$$

where $\mathbf{1}$ is a $T \times T$ square matrix with elements equal to 1, and $\vec{1}$ is a vector from this matrix, e.g., $\vec{1} = \text{diag}(\mathbf{1})$. Moreover, tr( ) is the well-known trace matrix function, and $\odot$ is the so-called Hadamard–Schur product. Consider that the constraints restrict the possible movements of the salesperson. In this way, it visits a single city after visiting the ith city and comes from a single city when arriving at the jth city.

2.1.3. Traveling Thief Problem

As in the TSP, the problem formulation considers a set of T cities where the information about the distance between each pair of cities ($\forall\ d_{ij} \in \boldsymbol{D}$) is contained within a known matrix $\boldsymbol{D}$. Moreover, every city i contains a finite set of items $\mathcal{M}_i = \{k_1, k_2, \ldots, k_{M_i}\}$, where $M_i = \#\mathcal{M}_i$ can vary from one city to the next, and each item has a profit $p_{ik} \in \mathbb{R}_+$ and a weight $w_{ik} \in \mathbb{R}_+, \forall\, k \in \mathcal{M}_i$. For the sake of simplicity, we define $N \triangleq \max\{M_1, M_2, \ldots, M_T\}$ as the maximum number of items from at least one city; this quantity is obviously linked with the knapsack problem. Thus, it is straightforward to introduce the $T \times N$ matrices $\boldsymbol{P}$ and $\boldsymbol{W}$ containing the profits and weights, respectively, for each item from any city; $(\boldsymbol{P}, \boldsymbol{W}) \in \mathbb{R}_+^{T \times N}$. Moreover, let us say that we currently have a tour of ordered cities $\mathcal{C} = \{c_1, c_2, \ldots, c_T\}$ where each city in the tour represents one of the cities available in the TTP instance since $T = \#\mathcal{C}$. In this way, $c_i \in \mathcal{C}$ represents the starting city, and $c_j \in \mathcal{C}$ is the next city that will be visited ($j = i + 1 < T$), even if it is not the closest one or does not represent the second city in the problem instance. Hence, $d_{c_i c_j} \neq d_{ij}, \forall\ (c_i, c_j) \in \mathcal{C}$ and $\forall\ d_{ij} \in \boldsymbol{D}$. Now, we can lay out a profit function $Z(\mathcal{C}, \boldsymbol{X})$ by incorporating a $T \times N$ binary matrix $\boldsymbol{X} \in \mathbb{Z}_2^{T \times N}$ representing the packing plan for each city. Recall that each element from this matrix $x_{ik} \in \boldsymbol{X}$ becomes one if an object k is taken from city i and is zero otherwise. Additionally, the *thief* rents a knapsack with a capacity K for a rate of R cost units per time unit. Lastly, the *thief* moves with a velocity v that decreases linearly from v_{max} to v_{min} as the knapsack gains weight, such as

$$v(c_i) = v_{min}\tilde{w}(c_i) + v_{max}(1 - \tilde{w}(c_i)), \tag{3}$$

since $\tilde{w}(c_i) \in [0, 1]$ is the percentage of the total knapsack capacity after leaving the city c_i. This factor corresponds to the rescaled version of the total weight of the items within the knapsack using K, which is determined as follows:

$$\tilde{w}(c_i) = \frac{1}{K}\left(\sum_{j=1}^{i} \hat{e}_{c_j}\right)^{\mathsf{T}} (\boldsymbol{W} \odot \boldsymbol{X})\vec{1}, \tag{4}$$

where $\hat{e}_l$ is the lth standard unit vector in the Euclidean space $\mathbb{R}^T$, and $\vec{1}$ is a vector with N elements equal one. Therefore, the objective function targets the tour $\mathcal{C}$ and the packing plan $\boldsymbol{X}$, seeking to maximize the profit Z given by:

$$Z(\mathcal{C}, \boldsymbol{X}) = \text{tr}(\mathbf{1}(\boldsymbol{P} \odot \boldsymbol{X})) - R\sum_{i=1}^{T} \frac{d_{c_i c_{(i \bmod T)+1}}}{v(c_i)} \tag{5}$$

Note that the first right-hand side term sums the profit of all packed items, while the second term is the cost associated with renting the knapsack.

2.2. Hyper-Heuristics

Hyper-heuristics are a relatively recent approach for solving optimization problems based on a combine-and-conquer approach [22]. Although they were initially described as "heuristics to choose heuristics" [23], they have outgrown such a statement. This can be inferred by looking at the various criteria and categories available for cataloging a subset of the existing models [24]. Such a bloom of ideas reflects the effectiveness of hyper-heuristics, which have successfully been applied to diverse problems [18].

In the context of this work, we refer to a hyper-heuristic as a model that combines different heuristics to solve a problem, i.e., we restrict our discussion to selection hyper-

heuristics on combinatorial problems. Nonetheless, some hyper-heuristics have achieved promising results in continuous optimization problems in recent years [19,25].

Selection hyper-heuristics usually solve problems one step at a time. In each step, they choose a heuristic to use from the available pool. This requires a decision model, which can be based on rules [26], or more complex models, such as those associated with rewards [27–29]. In the first case, one may have a set of rules represented through an array of problem features. Such features are used to identify the current nature of the problem instance, and the closest rule is selected since it should exhibit a similar nature. This rule defines an action, i.e., a heuristic, which pushes the solution of the instance forward, modifying it, thus leading to a new rule selection. In the second case, one may incorporate ideas from machine learning. For example, we can preserve the array of problem features for mapping the current state of the problem. Moreover, we can generate the feature values that training instances will yield when solved under different scenarios. This information can be grouped into clusters, providing a similar structure to the rule-based model. However, each cluster offers an expected performance level with each heuristic instead of having a single action. This information can be used greedily to select the heuristic expected to perform best. Once again, this modifies the problem state and leads to a new selection. Additionally, this approach requires updating the expected performance of the heuristic. This can be done based on whether the solution was better or worse than the expected outcome.

Even if the approaches above seem overly complex, it is important to remind the reader that the ultimate goal is to combine heuristics when solving a problem instance. Thus, a hyper-heuristic scheme can become as simple as a sequence of heuristics. This liberates the model from using features but may hinder performance if problems are too different. Sanchez-Diaz et al. explored this idea for the knapsack problem [30]. The authors laid out several operators that allowed extending and shortening the sequence of heuristics. Moreover, they included operators for perturbing the existing sequence by exchanging heuristics. Their data revealed that this kind of hyper-heuristics can outperform the base heuristics in up to 80% of the testing scenarios. Similarly, Sanchez et al. studied a sequence-based hyper-heuristic but for the balanced partition problem [31].

Despite its drawbacks, we chose to follow this path to define a base model that can be improved upon with further research. In simple terms, our model uses the simulated annealing (SA) algorithm to find an optimum sequence of heuristics for solving the instances. We detail the model in the next section.

3. Our Proposed Approach

This section details our proposed implementation for both the instance generator and the hyper-heuristic model.

3.1. Instance Generator

Remember that we may understand a TTP instance as a synergy between a TSP and a KP instance. Thus, it is possible to start with a TSP instance and modify it by distributing the items (from the KP instance) to each city. Afterward, we only require to define a renting rate and a knapsack capacity, which, by no means, suggests that the optimal solution for the TSP subproblem remains the optimal solution of the whole problem. This idea was demonstrated by Wagner [32], where the author used particular instances in which much larger paths led to better fitness values than those yielded by the optimal solution of the respective TSPs.

Naturally, such a statement also applies to the knapsack component of the TTP. A typical behavior in KP instances is that their difficulty relates to the correlation between the profit and the weight of the items. The instances where profit and weight are independent are usually easier to solve [11]. In contrast, instances where the profit is highly dependant on the weight are harder to solve [33]. However, as stated above, good KP solutions are not necessarily extendable to the TTP. The chief reason befalls to the renting rate. In the KP, it does not really matter whether an item is selected at the beginning or the end of the

solution process. Conversely, in the TTP, such dependence is strongly related to the renting rate, as it affects the time required to complete the tour and, ultimately, hinders the profit that can be achieved. Thus, heavier items cause more of an effect if they are chosen at the beginning of the solution process. This implies that it will likely be disregarded if the item is located in one of the first cities in the tour.

If we combine these basic ideas, then we can build multiple kinds of instances for the TTP. Thus, the parameters associated with the generator include the bounds for the weight and profit of the items and the degree of correlation between the profit and weight of the items. Additionally, they cover the restrictions associated with the renting rate and knapsack capacity. Polyakovski et al. classified KP problems inro three groups [10]: bounded strongly correlated, uncorrelated with similar weights, and uncorrelated.

Scattering items across cities is an arguable modification. At first glance, a non-uniform distribution may promise interesting examples. However, care must be taken as problems grow. Having a high percentage of cities with no or few items reduces their impact on the solution since the effect of the knapsack component has less of an impact. The renting rate (R) is paramount to the problem. Through it, one may affect the balance between both sub-problems and avoid dominating one over the other. For this work, we use $R = 0.5$ based on preliminary experiments.

Algorithm 1 shows one of the generators we used for creating the sets of instances for this work. Although our approach is simple, it is computationally cheap and allows the creation of instances with different correlation values. The other approach we followed is similar, but we used a simple assignment of random values to the profit and weight of each item.

Algorithm 1: Generation of strongly correlated instances with values used in this work. K, R, v_{min}, and v_{max} are arbitrarily selected. M_i is the number of items per city

Generate a $2 \times T$ matrix of random coordinates $\boldsymbol{M} \in [0, 3000]^{2 \times T}$, where each column stands for the coordinates of each city;
Fill the distance matrix $\boldsymbol{D}$ with the Euclidean distance between each pair of cities;
Set $K \leftarrow 500$, $R \leftarrow 0.5$, $v_{min} \leftarrow 5$, and $v_{max} \leftarrow 10$;
for $i \leftarrow 2$ **to** T **do**
  Set $M_i \sim \mathcal{U}(1, 10)$;
  for $k \leftarrow 1$ **to** M_i **do**
    $w_{ik} \sim \mathcal{U}(1, 100)$; `Random with a uniform pdf between 1 and 100`
    $p_{ik} \leftarrow 1.5 \times w_{ik} + \mathcal{U}(1, 5)$;
  end
end

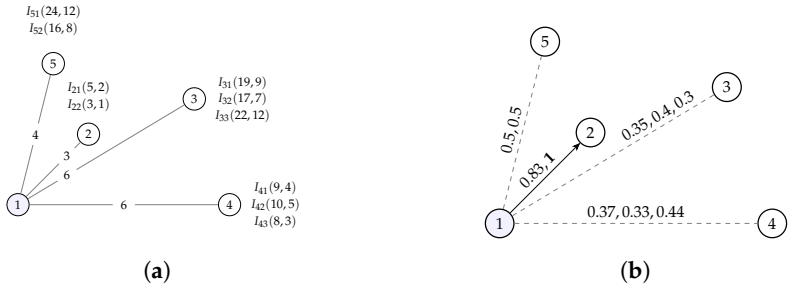
3.2. Solvers Considered for This Work

Since the TTP appears from fusing the KP and the TSP, we need two kinds of solvers. One of them is in charge of defining the next city to visit. The other must determine the items to be picked up at each city.

To keep our approach as simple as possible, we consider a simple greedy heuristic for selecting the next city to visit. Additionally, we consider three commonplace heuristics for picking items up. The heuristic targets the unvisited city with the best potential item in the first case. This is given by the city with the best profit/(distance × weight) value. The heuristic will react merely to the distance criterion if there are no feasible items, e.g., because the knapsack capacity is exceeded.

Figure 1 shows an example of the process followed by the city selection heuristic. In this case, we have an instance with five cities and a total of ten items (Figure 1a). Moreover, we must decide on the first stop on our tour. Since there are four available paths, the heuristic computes the profit over the weight ratio of each candidate item (those that can be picked up) at each unvisited city. Then, these ratios are divided by the length of the

corresponding path, yielding a score metric. The path that holds the best score is finally selected. In our example, this corresponds to city number 2, as indicated in Figure 1b by the value in bold.



(**a**) (**b**)

Figure 1. City selection process carried out by the `MOV` heuristic. (**a**) Possible movements from city #1. Edge: distance. Node: profit, weight; (**b**) Score metrics for each item and selected path.

The heuristics for picking items up are pretty straightforward. They can select the item with the highest profit (`MAXP`), the lowest weight (`MINW`), or the best profit-over-weight ratio (`MAXPW`). Remember that such a selection is made on the items available in the current city and that the items are only stored within the knapsack if they do not violate the maximum capacity constraint, i.e., the knapsack capacity K. Figure 2 shows the effect of using these heuristics in an arbitrarily selected city, where values in bold represent the parameter analyzed by each heuristic and where the blue font indicates the selected item.

3 $I_{31}(19,9)$ $I_{32}(\mathbf{17,7})$ $I_{33}(22,12)$ 3 $I_{31}(19,9)$ $I_{32}(17,\mathbf{7})$ $I_{33}(22,12)$ 3 $I_{31}(19,9)$ $I_{32}(17,7)$ $I_{33}(\mathbf{22},12)$

Figure 2. Item selection process carried out by each heuristic. Values in bold font represent the parameters analyzed by the heuristic.

Do note that the complexity of the optimization problem and the nature of the heuristic makes it impossible for a single heuristic to provide a good solution. For example, using only heuristics for picking items up render solutions where no cities are visited. Conversely, using only the next city heuristic provides a full tour with no selected items. Thus, we require a combination of both kinds of heuristics. To this end, our approach promotes that the number of city movements equals the number of cities within the instance.

3.3. Definition of a Hyper-Heuristic Sequence

In this work, we define a hyper-heuristic (HH) as a sequence of heuristics that are successively applied to the problem. However, only specific sequences render valid solutions. In this regard, the HH is given by a combination of low-level operators of any length, where repetitions are allowed. This promotes that the number of city selection occurrences equals the number of cities within the problem instance. Moreover, the first two movement operators are always side-by-side, and the sequence ends with an implicit movement operation. This is because the starting city does not contain items to pack and because the tour must be a closed loop around the cities. Hence, we aliased this first occurrence of the movement heuristic as `INIT`. Figure 3 provides an example of a feasible HH for a problem instance with five cities. Note that item selection heuristics can appear as many times as desired, although they do not require selecting an item at all times. This is because the number of items per city changes from one instance to another.

INIT	MOV	MAXPW	MOV	MOV	MAXPW	MAXPW	MOV	MAXPW	MAXPW	MAXP	MAXP

Figure 3. An example of a feasible sequence-based hyper-heuristic for an arbitrary TTP instance with five cities.

Let us now analyze how such an HH solves the problem instance from Figure 3. This instance provides a step-by-step solution process. The first movement operator selects city number 2 as the first stop since it contains the item with the best potential (Figure 3). Note that each subfigure shows either the result from movement or selection heuristics. Additionally, all operators are processed even if they select no items because constraints are violated or because no items remain in a city. This behavior is noticeable in Figure 3. The hyper-heuristic offers four selection operators in this city, but the city only contains three items. Hence, the final item selection heuristic has no effect. Consequently, this heuristic is marked as disregarded. Finally, remember that all hyper-heuristics incur an implicit movement operator that closes the tour returning to the initial point, as indicated in Figure 3.

3.3.1. Reduction of a Hyper-Heuristic

Remember that a valid HH may have any arbitrary length as long as it complies with the minimum number of city selection operations. This is possible because our implementation ignores the request of an item selection heuristic when no valid objects are left in the current city. Thus, adding more item selection operators does not improve performance, although it increases the computational burden of the model. Hence, one may arrive at excessively long sequences.

To assess this behavior, we include a capability for preserving short and meaningful sequences. Whenever an item selection operator is disregarded when solving a training instance, it is marked for possible deletion. After all the instances are solved for each iteration of the training phase, the algorithm filters operators. Should an operator be marked in more instances than a given threshold, it is removed from the sequence. Otherwise, it is preserved. We refer to this criterion as the democratic removal criterion.

3.4. Training Phase Based on Simulated Annealing

Additionally, we require an approach that tests different sequences and refines the model based on their performance to improve our proposed model. We achieve this by adapting the simulated annealing (SA) algorithm proposed by Kirkpatrick et al. [34].

We show an overview of the adapted procedure in Algorithm 2. Note that the first step is to define the initial temperature for the algorithm. Hence, we set this value based on a starting threshold, as Algorithm 3 shows. This process continuously generates Markov chains until the minimal acceptance ratio, τ_{min}, is achieved. Temperature is increased at a constant rate β with each iteration t. For this work, we found that $\tau_{min} = 75\%$ and $\beta = 1.5$ provide good results.

Afterward, the algorithm iterates by evaluating candidate solutions and determining whether they are accepted in a probabilistic manner. This requires our main change, which stems from how a neighboring solution is obtained. To preserve a similarity with the current sequence, we alter sequences by modifying the order of its operators based on the following low-level rules and excluding the `INIT` and the first `MOV` operators:

Swap. Exchanges two randomly selected operators.

Flip. Fully inverts the sequence of operators.

Shuffle. Randomizes the order of all operators in the sequence.

Repeat. Duplicates the sequence by appending a copy of the original sequence.

Reflect. Doubles the sequence length by appending a flipped copy of the original sequence.

Random block. Returns a sequence appending a random block of five heuristics.

Random item. Extends the sequence by appending a random heuristic operator.

Delete last. Shortens the sequence by deleting the last element.

Delete random. Shortens the sequence by deleting a random element.

Algorithm 2: Simulated annealing algorithm used for training hyper-heuristics. l_{max} is the length of a Markov chain, representing the number of tries for a single temperature. Stopping criteria considered are the number of rejections (1000), the minimal temperature achieved (10^{-9}), and number of iterations (10^6). A neighbor solution is generated using a randomly chosen sequence perturbator, as indicated in Section 3.4.

$\alpha \leftarrow 0.8$ and $l_{max} \leftarrow 100$;
$u_0 \leftarrow$ [`INIT`, `MOV`];
$\theta \leftarrow$ Algorithm 3;
$u \leftarrow u_0$ and $rejects \leftarrow 0$;
while *stopping criteria not met* **do**
  $s \leftarrow 0$;
  for $l \leftarrow 1$ **to** l_{max} **do**
    $v \leftarrow \text{neighbor}(u)$;
    if $\mathfrak{F}(v) \leq \mathfrak{F}(u)$ **then**
      $u \leftarrow v$, $rejects \leftarrow 0$, and $s \leftarrow s + 1$;
    else
      if $\mathcal{U}(0,1) < \text{Exp}\left(-\frac{\mathfrak{F}(v)-\mathfrak{F}(u)}{\theta}\right)$ **then**
        $u \leftarrow v$ and $s \leftarrow s + 1$;
      else
        $rejects \leftarrow rejects + 1$;
      end
    end
    if $s > 20$ **then**
      break;
    end
  end
  $\theta \leftarrow \alpha\theta$;
end
return u;

Algorithm 3: Temperature initialization considering a warming constant $\beta = 1.5$ and a target initial acceptance rate $\tau_{min} = 75\%$

$\theta \leftarrow 0.1$;
$u \leftarrow$ [`INIT`, `MOV`];
$\tau_{min} \leftarrow 0.75$, $\beta \leftarrow 1.5$, and $\tau \leftarrow 0$;
while $\tau < \tau_{min}$ **do**
  $s \leftarrow 0$;
  for $l \leftarrow 1$ **to** l_{max} **do**
    $v \leftarrow \text{neighbour}(u)$;
    if $\mathfrak{F}(v) \leq \mathfrak{F}(u)$ **then**
      $u \leftarrow v$ and $s \leftarrow s + 1$;
    else
      if $\mathcal{U}(0,1) < \text{Exp}\left(-\frac{\mathfrak{F}(v)-\mathfrak{F}(u)}{\theta}\right)$ **then**
        $u \leftarrow v$ and $s \leftarrow s + 1$;
      end
    end
    $\tau \leftarrow \frac{s}{l_{max}}$;
  end
  $\theta \leftarrow \beta\theta$
end
return θ;

Note that only one of these operators is selected per iteration (in a random fashion) and that there are rules that preserve, extend, and shorten the sequence length. We do this to provide a higher degree of variability to the training process. Additionally, when training with a set of instances, the model's performance is calculated by summing the profits achieved across all the instances in the training set. Remember that the penalty discussed in Section 3.5 is calculated on a per-instance basis. In this sense, if a hyper-heuristic in-training has few `MOV` operators, there may be several instances where the tour is not completed. Therefore, the final value for the fitness function is penalized for each of these instances, thus promoting solutions with a higher number of `MOV` operators.

3.5. Fitness Function and Promotion of City Movements

As Algorithm 2 indicates, our model begins with a two-element sequence corresponding to `MOV` operators. From there, it begins mutating and improving its performance. Since we start with a couple `MOV` operators, our approach requires a way to promote an adequate number of this kind of operator in the final solution. We tackle this issue from a dual perspective. On the one hand, we deter unfinished tours by penalizing solutions with less `MOV` operators than the number of cities in an instance. On the other hand, the algorithm trims excessive `MOV` operators through the democratic criterion (see Section 3.3.1).

Let us delve deeper into the behavior of the penalty factor. For this work, we add a fixed value whenever the tour ends without visiting all cities. Based on preliminary experiments, we found that a value of 10^6 suffices for deterring this behavior. Moreover, we also apply this penalty if the knapsack capacity is exceeded by the end of the tour as a fail-safe (items are not packed if they do not fit). Nonetheless, we only penalize the function once, even if both constraints (cities and capacity) are violated. Additionally, we disregard the number of cities the solution does not visit.

Let us imagine that we have a problem instance with five cities and the solution given by (`INIT, MOV, MAXP, MOV`). In this case, our *thief* does not visit two cities. Therefore, the fitness given by our objective function is penalized by 10^6. It is also important to remark that the penalty is applied per instance. Thus, assuming we have a training set of instances where the *thief* does not visit all cities in o instances, the fitness value for such a hyper-heuristic will be affected by $o \times 10^6$. In this way, this candidate solution will be hindered and thus hopefully discarded.

Let us now assume that the solution evolves to the point where it has seven `MOV` operators. As for the instance with five cities, the tour would have finished before navigating through the whole sequence. Thus, this instance will mark a chunk of the sequence for eventual deletion. If enough instances agree, those elements will be deleted, thus trimming the hyper-heuristic.

In summary, the objective function that our proposed approach incorporates is given by Equation (6), where $Z_s(\mathcal{C}, \boldsymbol{X})$ is the profit achieved over instance s that Equation (5) presented. Moreover, S is the total number of training instances, and q_s is a binary variable indicating if the solution must be penalized for the given instance.

$$\mathfrak{F} = \sum_{s=1}^{S} Z_s(\mathcal{C}, \boldsymbol{X}) - 10^6 q_s \tag{6}$$

4. Methodology

We followed a four-stage methodology in this work, as Figure 4 shows. We begin by testing the proposed instance generator, using it to create some sets of instances. Then, we use such sets to evaluate our proposed model and its generalization capabilities. Please consider that we conducted all experiments using the same random number generator for repeatability purposes (the Mersenne Twister algorithm with a seed of 0). Besides, all hyper-heuristics have access to all the heuristics from Section 3.2.

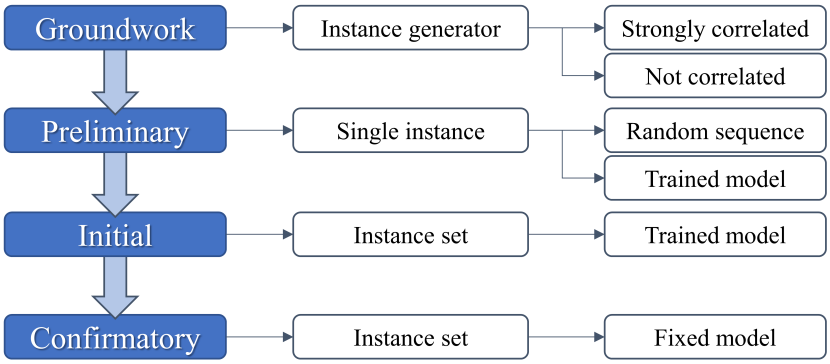


Figure 4. Four-stage methodology followed throughout this work.

4.1. Groundwork Stage

First, we set up the basis for our experiments. Thus, we create several instances distributed along three sets using the parameters shown in Table 1. The first two sets belong to the same kind, containing instances for training (Set 1) and testing (Set 2) of our model. These instances exhibit a strong weight-profit correlation. The final set of instances contains 30 TTP instances with no correlation between the profit and weight of the items. We use this dataset in the final stages mainly to verify if the trained hyper-heuristics generalize properly.

Table 1. Parameters used for generating the datasets considered for this work.

Parameter	Set 1	Set 2	Set 3
Number of instances (S)	20	10	30
Item distribution	Uniform	Uniform	Uniform
Weight-profit correlation (P)	1.5	1.5	N/A
Number of cities (T)	5	5	5
Knapsack capacity (K)	500	500	500
Renting rate (R)	0.5	0.5	0.5
Speed interval (v_{min}, v_{max})	[5, 10]	[5, 10]	[5, 10]
Weight limits (min, max)	[1, 100]	[1, 100]	[1, 100]
Profit limits (min, max)	N/A	N/A	[1, 100]

4.2. Preliminary Stage

We split the preliminary stage into two steps. The first analyzed the performance of randomly generated heuristic sequences over a single TTP instance. Our goal with this test was to assess the feasibility of using a heuristic sequence for improving performance with respect to standalone heuristics. To this end, we produced 1000 random sequences with a fixed length of 64 operators (heuristics) and analyzed their performance over the first instance from dataset 1.

The second step was somewhat more complex. It sought to determine whether trained hyper-heuristics can outperform random ones. Thus, we strove to see if the training cost can be justified by its benefits. Once again, we built sequences of heuristics. However, this time, we used simulated annealing (SA) to maximize the performance of the resulting hyper-heuristic. Because of the stochastic nature of SA, we generated 20 sequences for comparing their distribution with respect to the random approach. We chose SA since it represents a low-cost metaheuristic that can escape local optima with the parameters indicated in Algorithm 2.

4.3. Initial Stage

Seeking to develop a broader analysis, we extended the experiments from the previous stage to the remaining 19 instances from dataset 1. Since this implies generating and evaluating several hyper-heuristics per instance, we focused our analysis on those representing the median performance level across the runs. Moreover, we compared the performance of these specialized HHs against HHs trained over the whole dataset, which we deemed general hyper-heuristics. Our goal in doing so was to achieve a good model at solving sets of instances, even if it does not use them throughout the training stage. Additionally, we considered the following values for analyzing the effect of the democratic cleanup criterion (see Section 3.3.1): 30%, 60%, and 90%. Remember that the higher the threshold, the more instances must be solved without using an operator before it is removed.

4.4. Confirmatory Stage

The goal of our final approach was three-fold. For starters, it sought to determine how the previously trained HHs behaved when solving unseen instances of the same kind. In this stage, we also wanted to analyze how such HHs performed on instances of a different kind. Finally, we strove to compare their performance against that of hyper-heuristics specialized in this kind of instance. Hence, we also split this stage into two steps. The first one summarized the tests with the second dataset, which contains 10 instances. The other one gathered the data for the third dataset, which incorporated 20 instances of a different nature. In all cases, we considered thresholds of 30%, 60%, and 90% for the democratic criterion. Moreover, in some experiments, we also analyzed the effect of a criterion of 100%.

5. Results

We now present our most relevant findings. For the sake of readability, we preserve the same structure discussed in the methodology.

5.1. Groundwork Stage

Let us begin by observing the resulting instances. Figure 5 shows the number of items that each city contains across all instances in the first dataset as an example. Note that city numbering (x-axis) begins at two, since the first city contains no items as per the problem description (i.e., it is the starting city). Moreover, we can see that instances (y-axis) contain a varying number of total items and that no city contains more than 10 items at a single time. Additionally, note that instances are varied. For example, instances 12 and 18 exhibit two cities with the maximum number of items, albeit in a different setup. Similarly, instance 7 contains two cities with a single item and a smaller number of total items (12 instead of the 25 and 31 that the other two instances incorporate).

It is also important to verify that items exhibit the desired correlation between profit and weight. To avoid overextending the manuscript, we limit ourselves to the largest datasets as they exhibit a different nature. Figure 6 shows the distribution of the profit-over-weight ratio for each item across the 20 instances from Set 1. Although there are some outliers near the 4 units mark, most of the items are clustered near the value of 1.5 units. Conversely, Figure 7 shows the distribution for Set 3, where the profit and weight of each item were defined in a random fashion. This is evidenced by the wider spread of the violins. Thus, both datasets have different natures and shall prove to be interesting scenarios for testing our model.

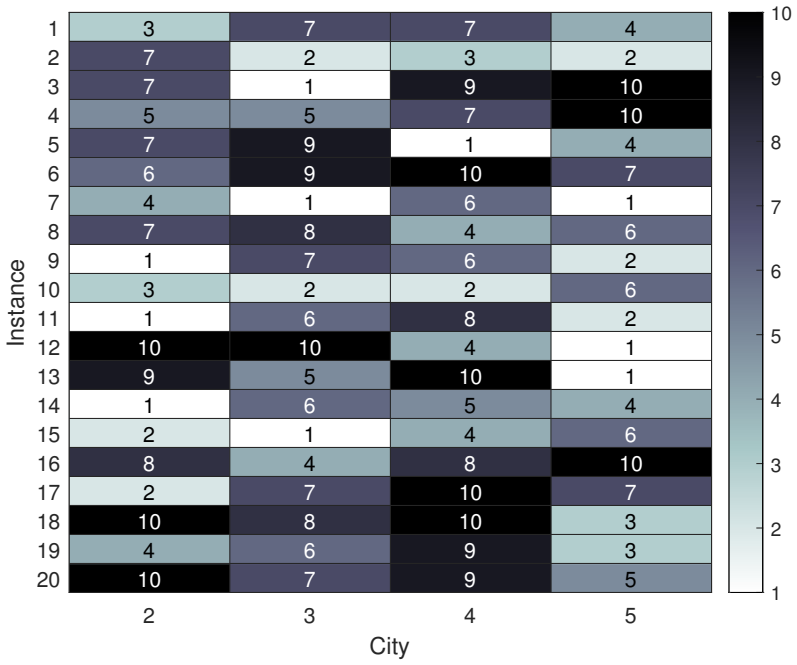


Figure 5. Item distribution across cities for the instances contained in the first dataset.

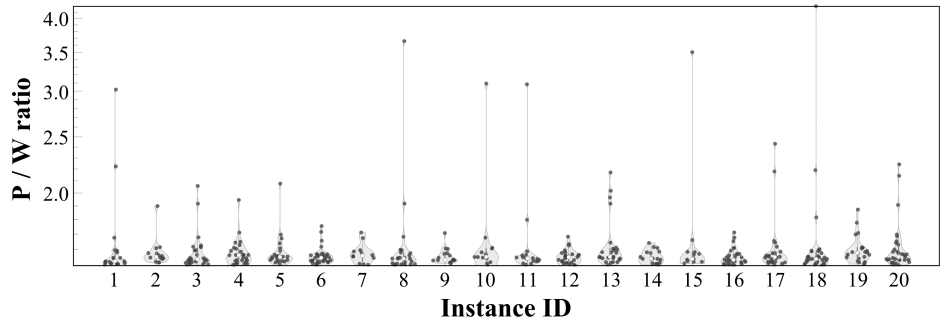


Figure 6. Distribution of the profit/weight ratio of items within each instance from Set 1 (bounded strong correlated instances).

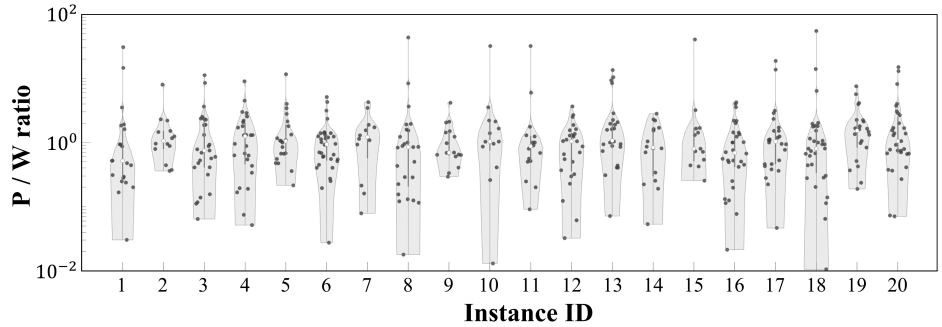


Figure 7. Distribution of the profit/weight ratio of items within each instance from Set 3 (uncorrelated instances).

5.2. Preliminary Stage

We continue our study by observing the feasibility of using a sequence-based hyper-heuristic for solving traveling thief problems (TTP) with simple heuristics. Moreover, we want to determine whether simulated annealing (SA) is useful for training the proposed model. To do so, we considered one instance at a time, seeking to develop the model for the most simple scenario while comparing its performance against randomly generated sequences.

5.2.1. Performance of Random Sequences

Our data reveal that not all 64 operations are actually used when solving the first instance from Set 1. The number of operators that are actually used ranges from 5 to 58, with average and median lengths of 17 and 16, respectively. Figure 8 provides an overview of the performance of these sequences (left violin). Note that the majority of them yield positive profit values, meaning that our *thief* will actually earn something after it sells the items collected during the tour. However, there is also an important chunk of entries with negative values. Therefore, this approach does not guarantee that a single sequence, i.e., a hyper-heuristic, performs properly. Even so, the median profit sits near the mark of 100 units.

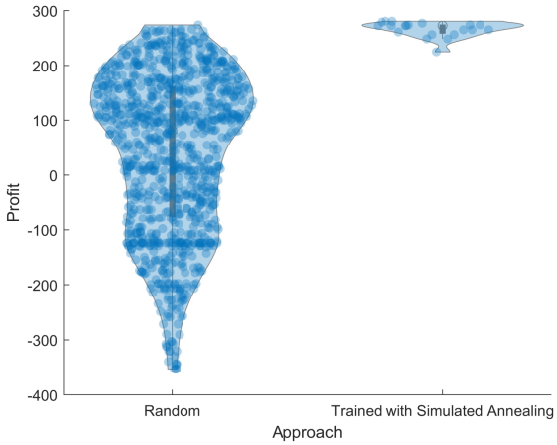


Figure 8. Performance distribution achieved by 1000 random and 20 trained hyper-heuristics.

5.2.2. Performance of a Trained Model

Let us now see what happens when we train our proposed approach. Figure 8 shows the distribution of the performance achieved by 20 hyper-heuristics trained for this instance (right violin) using simulated annealing (SA). It is evident that our proposal outperforms the simple approach of generating random sequences. Not only do we reach a slightly higher maximum profit, but we also do it so more consistently. In fact, the median performance achieved by these hyper-heuristics (near the 300 units mark) is quite similar to the maximum performance achieved by the random sequences and almost thrice as good as the median performance of the random approach. Therefore, SA stands as a valid choice for training our model, at least under the current constraints.

5.3. Initial Stage

Since our proposed model performs well for a simple scenario, let us now analyze whether it works for more complex ones. Remember that we incorporate a democratic elimination criterion, and so we must also analyze its effect. Additionally, keep in mind that we seek a sequence-based hyper-heuristic that properly solves the whole instance set.

The resulting data (Figure 9) are promising. Note that, once again, we have included the performance of the random approach (first violin) for comparison purposes and for validating if training is worthwhile. The second violin shows the performance of independently trained hyper-heuristics, i.e., hyper-heuristics specialized for the given instance. It is noteworthy that, as with the first instance (i.e., the one we analyzed in the previous stage), the performance of trained hyper-heuristics is stable for most of the set and is also higher than that of randomly generated hyper-heuristics. There are only a couple of scenarios where some runs yield negative profits, which relate to instances 3, 13, and 17.

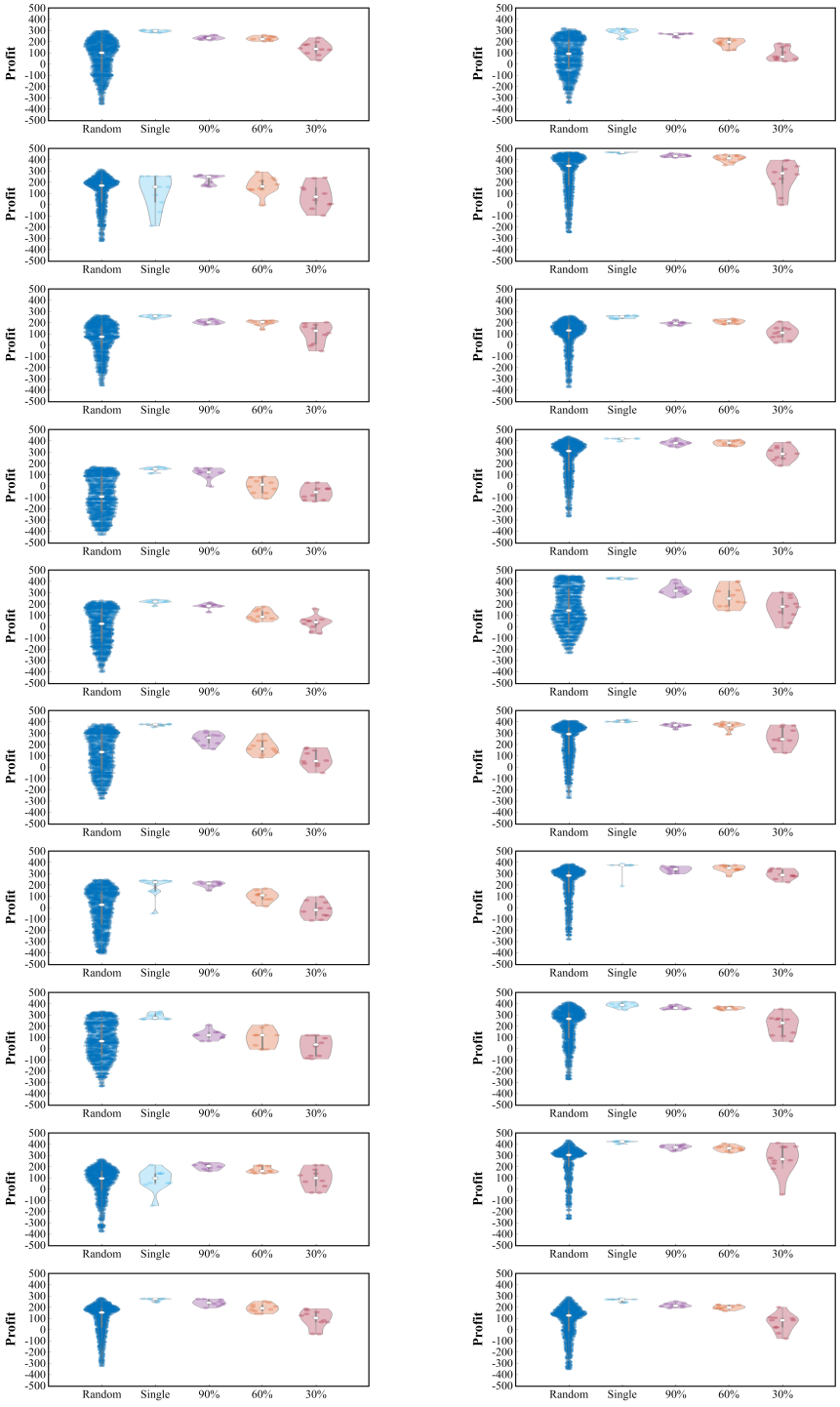


Figure 9. Profit achieved by different kinds of sequence-based hyper-heuristics (HHs) over each instance from the first dataset. Each violin (left to right) corresponds to 1000 randomly generated sequences, 10 HHs trained for a single instance, and 10 HHs trained over the whole set with different thresholds for the democratic elimination criterion (90%, 60%, and 30%, respectively).

Figure 9 also includes the performance of hyper-heuristics trained for the whole dataset and with different democratic elimination criteria (remaining violins). As one may note, using a threshold of 90% leads to a performance level akin to that achieved by specialized hyper-heuristics over their corresponding instances. Moreover, performance is no longer negative in the troublesome instances, and it actually improves upon the median performance of instance 3. It is also noteworthy that the most stringent elimination criteria

lead to the best and most stable performance levels. However, they also lead to longer sequences. Even so, increasing the criterion by 30% only seems to extend the sequences in one or two operators. Therefore, the performance gain may justify the extra steps that hyper-heuristics require. In any case, the sequences remained somewhat short (up to 17 operators, on average, for the 90% criterion), and so they can be implemented in a straightforward fashion. Moreover, this hints at the idea of exploring the effect of voting thresholds near the 100% mark, though this goes beyond our scope. Additionally, note that as the voting threshold diminishes, the performance of the resulting hyper-heuristics becomes more erratic, and so their violins widen.

We now strive to analyze the performance from a different perspective. Thus, we focus on the median performance of the hyper-heuristics. Figure 10 displays the normalized profit achieved by each representative hyper-heuristic across all instances. Keep in mind that, since most hyper-heuristics specialize in a single instance, we expect them to perform poorly in the others. Conversely, we expect that the hyper-heuristic trained in the whole set performs properly across most instances. As one may detect from the figure, the majority of specialized solvers perform best in their own instances, as indicated by the yellow blocks. The only exceptions are for instances 3 and 17. Furthermore, some of these solvers perform well for more than one instance. Hyper-heuristics trained with the fourth instance are an example of this scenario, as they reached values near unity for almost 10 instances. Others, however, are only useful for a few instances, as with hyper-heuristics trained with instances 14 and 15. This indicates that such instances significantly differ from the others despite having been created with the same approach. Figure 10 also shows that the hyper-heuristic trained in the whole set with a 90% democratic elimination criteria performs properly for the majority of instances, including instances 3 and 17.

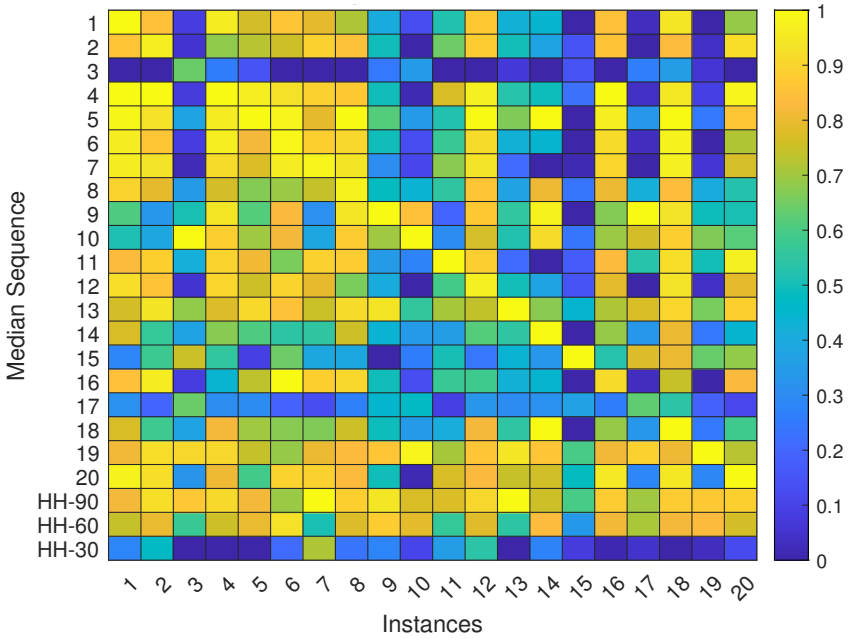


Figure 10. Median performance (normalized profit) achieved by different kinds of hyper-heuristics (HHs) over the first dataset (10 runs). Numbered rows represent HHs trained over a single instance (given by their number). The last 3 rows display the performance of HHs trained over the whole dataset (with democratic elimination criteria of 90%, 60%, and 30% respectively).

It is also clear that blueish blocks predominate along the columns that represent some of the instances, such as those for instance 10. This means that most of the hyper-heuristics specialized for the other instances are unable to properly solve this one. An exception to this pattern is the hyper-heuristic specialized for instance 9, which achieves a similar performance level. This behavior may be due to a similarity between these instances while remaining somewhat different from the others. Moreover, these specialized hyper-

heuristics tend to perform poorly over the other instances, reinforcing this suspicion. In any case, the hyper-heuristic trained over the whole set performs well on instances 9 and 10, yielding similar performance values. Furthermore, it performs well for the remaining instances, thus overcoming the aforementioned drawback.

Let us detail one of the difficult instances. Figure 11 shows that in instance 15, cities closest to the starting point have few items. Moreover, these items contain a similar configuration. Distant cities, on the contrary, contain the best objects. Since we are only considering a single city selection heuristic, this makes it harder to detect such items. Additionally, the city selection heuristic is greedy and combines information between profit and weight of the items, along with the distance to the corresponding city. Because of this, the heuristic may perform poorly when good items are in faraway cities. Thus, it becomes difficult to choose a proper city to visit first.

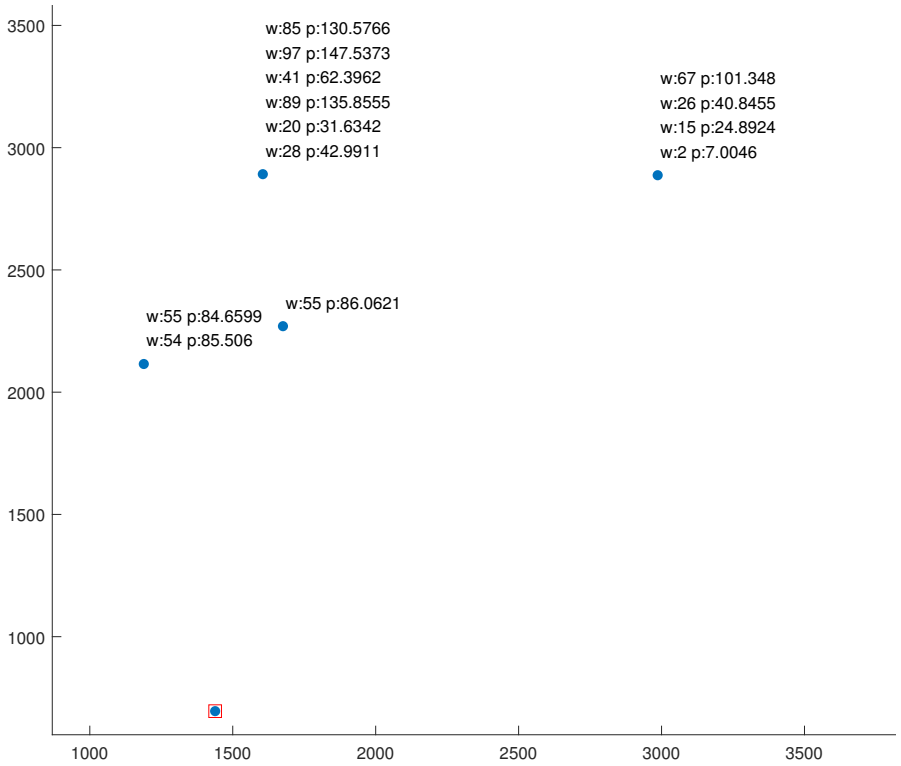


Figure 11. Layout of instance 15. Each circle represents a city, and texts display the weight (w) and profit (p) for every item within each city. The red box indicates the starting city.

5.4. Confirmatory Stage

Up to this point, we have seen that hyper-heuristics trained over the whole set of instances perform well. Thus, we must now analyze how they fare against unseen instances. Bear in mind that such instances can be of the same or different kinds.

5.4.1. Generalization to the Same Kind of Instances

Let us begin with instances of the same kind, as they should exhibit the least discrepancy. Figure 12 shows the median performance of general HHs over the testing set of 10 instances for the different thresholds of democratic criterion. As with Figure 10, the heatmap also shows the median performance for the training instances. Moreover, it shows the median performance of specialized HHs for both datasets (training and testing). Once again, general HHs behave well, especially with more stringent criteria, reaching an overall normalized score beyond 0.7 for the highest threshold (90%). We believe this is noteworthy since these instances are unknown to the model. Moreover, these HHs outperform some of the specialized solvers created for this dataset, which is also remarkable.

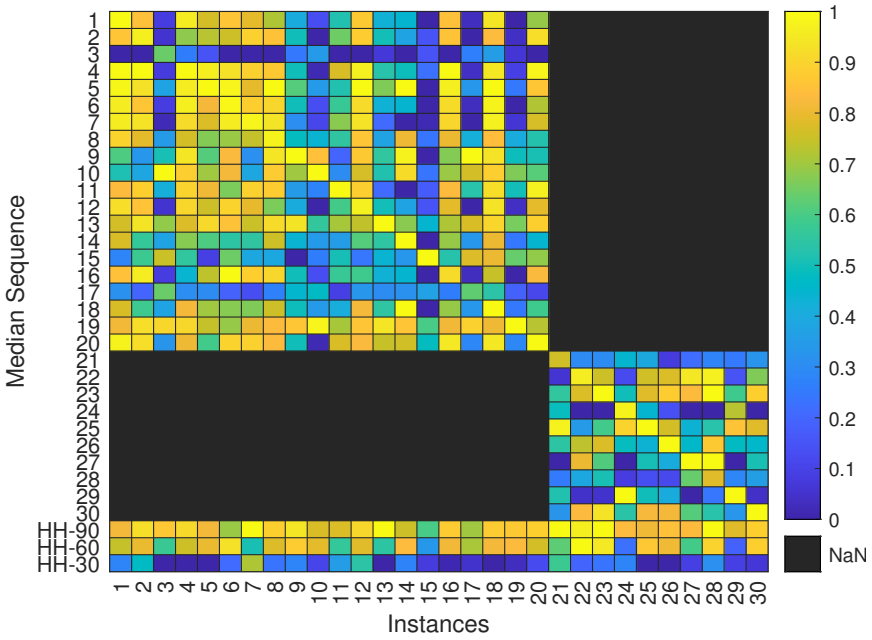


Figure 12. Median performance (normalized profit) achieved by different kinds of hyper-heuristics (HHs) over the first and second datasets (10 runs). Numbered rows represent HHs trained over a single instance (given by their number). The last 3 rows display the performance of HHs trained over the first dataset. Performance related to the first dataset is included for comparison purposes.

5.4.2. Generalization to a Different Kinds of Instances

We shall now analyze if the trend of general HHs holds for a different set of instances. Remember that here, we use 20 unseen instances of the non-correlated type (Section 4.1). There is an interesting distinction with respect to previous tests. Before, a specialized hyper-heuristic performed relatively well for some of the remaining instances. However, as Figure 13 shows, this is no longer the case. This is evidenced by the plethora of greenish and blueish blocks that indicate normalized performance levels of 0.6 and below. Hence, the uncorrelated features of the instances mark a clear challenge to overcome when finding sequences that perform well over this class of instances.

Let us observe how general hyper-heuristics face this challenge. As the bottom part of Figure 13 displays, we plot the performance of two kinds of hyper-heuristics. First, we have the performance of HHs trained with the set of correlated instances (labeled as HH). Although more stringent thresholds lead to better performance, the overall performance is far from perfect. In the best-case scenario (HH-90), the average normalized profit across these 20 instances was 0.61. Moreover, for a different set of 10 unseen instances, this value diminished to 0.59. This is worse than the performance achieved by some of the specialized HHs, such as the one created for instance 20, which reached a value of 0.77 over the 20 instances.

Nonetheless, there is another kind of HH that performs better. The final four rows from Figure 13 summarize the performance of hyper-heuristics trained for this set of uncorrelated instances with different democratic thresholds. It is interesting to see that even medium-sized values lead to good behavior. For example, a threshold of 60% yields a median normalized profit of 0.72 over the 20 instances. Although this is still lower than for the specialized HH, the latter becomes outperformed when the threshold rises to 90%, since the general HH achieves a mark of 0.80. Although more restrictive thresholds improve overall performance, it is noteworthy that there is no single value that solves all instances in the best way. For example, a threshold of 60% leads to a normalized value beyond 0.80 for instance 15. However, should one use a threshold of 100%, this value rises near unity. Therefore, it could prove worthwhile to develop an approach for combining

hyper-heuristics trained with different thresholds or to modify the model so that it exhibits an adaptive threshold.

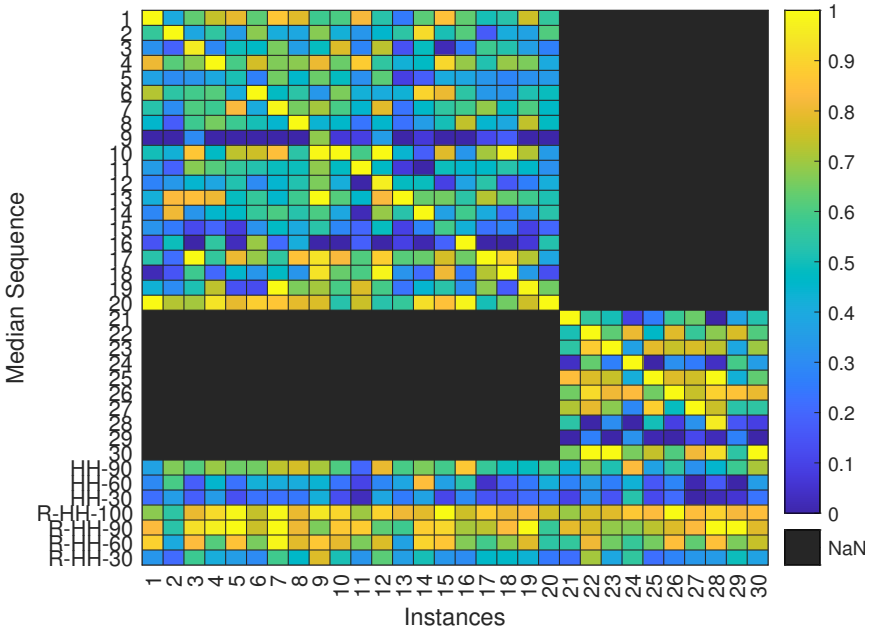


Figure 13. Median performance (normalized profit) achieved by different kinds of hyper-heuristics (HHs) over the third dataset (10 runs). Numbered rows represent HHs trained over a single instance (given by their number). Rows 31 to 34 (labeled as HH) display the performance of HHs trained over the first dataset. The remaining rows (labeled as R-HH) show the performance of HHs trained over the third dataset with different democratic elimination thresholds.

It is also important to mention that these HHs perform well, even for unseen instances. This subset is represented in rows 21 to 30 in Figure 13. As we can see, specialized HHs perform well for their own instance, but tend to perform poorly for the others, as with the previous subset. However, the trend for general HHs trained with a high democratic threshold is to deliver good behavior. For example, a threshold of 100% leads to normalized profits beyond 0.69 for all the testing instances. This is something impossible to achieve with the specialized hyper-heuristics, and so it represents an opportunity for this kind of solvers.

6. Conclusions

In this work, we explored the traveling thief problem (TTP) and how we can solve it through hyper-heuristics. To this end, we laid out a four-stage methodology covering 60 problem instances and two classes of instances. The first testing stage dealt with the development of a basic instance generator. This allowed us to create three sets of instances, which we used throughout the remaining stages. Then, we analyzed the feasibility of training specialized hyper-heuristics, focused on improving the performance for a single instance. We also focused on comparing how performance changes for a general hyper-heuristic, i.e., one trained over a set of instances when solving different sets. In this work, we used simulated annealing (SA), although other metaheuristics or approaches can be used with few changes.

Our data reveal that spending computational resources on training hyper-heuristics for the TTP is worthwhile. When comparing the performance of the resulting hyper-heuristics against that of random operator sequences, we found that the former exhibits higher stability and better performance, located above the 85th percentile of the available range. Conversely, the latter yielded poor solutions, resulting in negative profit values. Moreover, our proposed approach seems to generalize well for the TTP. For the strongly correlated instances, the HHs trained over the whole training dataset achieved scores beyond 80%

of those found by the specialized sequence for 16 out of the 20 instances. Conversely, specialized HHs tended to perform poorly in some cases.

This work is only a stepping stone toward developing more robust solvers for the TTP. Thus, it is only natural that several paths lie ahead. One of them is to evaluate the operation of the model across more complex sets of instances. This is relevant because, in this work, we restricted ourselves to several instances with a few parameters to avoid overextending the manuscript and as a proof-of-concept for our approach. A richer variety of instances with more cities and items would allow simulating problems closer to reality. Furthermore, it would permit analyzing the effect of training with a set of instances of varying complexity on the generalization capabilities of the model.

Another feasible path is to study the complexity of building and training hyper-heuristics. The main purpose would be to evaluate how it grows in terms of sequence length and instance size and the way in which memory and time consumption escalate. This, of course, also implies analyzing how performance changes under these conditions and comparing the resulting data against other state-of-the-art approaches.

The last research path befalls the development of more complex hyper-heuristic models for tackling the TTP. In this work, we used a simple model based on a sequence of heuristics. However, this is not the only approach to developing hyper-heuristics. Therefore, future work may focus on developing a set of rules for mapping the current state of an instance in such a way that a rule-based approach can be implemented. Moreover, one could analyze the effect of transforming such features to improve the performance of the model or use feature information to complement sequence-based hyper-heuristics.

Finally, one could direct their efforts towards a recursive hyper-heuristic model. This would generate a hyper-heuristic of hyper-heuristics, which could help further improve the generalization capabilities of the approach. Bear in mind that such a stratified solver allows for variety, as one may have a rule-based hyper-heuristic that selects among the sequence-based hyper-heuristics trained for different kinds of instances. Alternatively, one may pursue more intricate endeavors, such as hyper-heuristics that select among other kinds of solvers, such as neural networks or decision trees. In any case, hyper-heuristics stand as a good alternative for tackling traveling thief problems.

Author Contributions: Conceptualization, J.M.C.-D., J.C.O.-B. and I.A.; Data curation, D.R.; Formal analysis, D.R., J.M.C.-D., J.C.O.-B. and I.A.; Funding acquisition, I.A.; Investigation, D.R., J.M.C.-D., J.C.O.-B. and I.A.; Methodology, J.M.C.-D., J.C.O.-B. and I.A.; Project administration, I.A.; Resources, D.R. and I.A.; Software, D.R. and I.A.; Supervision, I.A.; Validation, J.M.C.-D., J.C.O.-B. and I.A.; Visualization, D.R.; Writing—original draft, D.R. and I.A.; Writing—review & editing, J.M.C.-D., J.C.O.-B. and I.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Tecnologico de Monterrey, strategic research group in intelligent systems, and by the Mexican National Council of Science and Technology CONACyT grant number 287479.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data available upon request.

Conflicts of Interest: The authors declare no conflict of interest.

Nomenclature

Symbols

$\mathcal{C}$	Set of ordered cities
c	City
$\boldsymbol{D}$	Distance matrix
$\hat{e}$	Standard unit vector
$\mathcal{E}$	Set of edges
$\mathfrak{F}$	Objective Function
$\mathfrak{G}$	Graph representing a traveling salesperson problem
K	Knapsack capacity
k	Item
l	Attempt counter for simulated annealing
$\mathcal{M}$	Set of items available within a city
M	Number of items available within a city
N	Number of items within an instance
p	Profit
P	Weight-profit correlation
$\boldsymbol{P}$	Profit matrix
q	Binary penalization variable
R	Renting rate
S	Number of training instances
θ	Temperature for simulated annealing
#	Cardinality of a finite set
T	Number of cities within an instance
u	Candidate solution
$\mathcal{V}$	Set of cities
v	Velocity
$\tilde{w}$	Percentage of the total knapsack capacity after leaving a city
w	Weight
$\boldsymbol{W}$	Weight matrix
x	Binary variable
$\boldsymbol{X}$	Binary matrix
$\vec{x}$	Binary vector
Z	Profit achieved after solving a problem instance
α	Cooling constant for simulated annealing
β	Heating constant for simulated annealing
τ	Acceptance threshold for simulated annealing

Subscripts

0	Initial value
i	Origin city index
j	Destination city index
k	Item index
max	Maximum value
min	Minimum value
s	Instance index

Acronyms

COP	Combinatorial optimization problem
HH	Hyper-heuristic
`INIT`	Initial movement heuristic
KP	Knapsack problem
`MAXP`	Maximum profit heuristic
`MAXPW`	Maximum profit per weight unit heuristic
`MINW`	Minimum weight heuristic
`MOV`	Movement heuristic
SA	Simulated annealing
TSP	Traveling salesperson problem
TTP	Traveling thief problem

References

1. Bonyadi, M.R.; Michalewicz, Z.; Barone, L. The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In Proceedings of the 2013 IEEE Congress on Evolutionary Computation, Cancun, Mexico, 20–23 June 2013; pp. 1037–1044.
2. Osaba, E.; Yang, X.S.; Del Ser, J. Traveling Salesman Problem: A Prospective Review of Recent Research and New Results with Bio-Inspired Metaheuristics and Novelty Search. Available online: https://www.sciencedirect.com/science/article/pii/B9780128197141000208 (accessed on 20 November 2022).
3. Daravigkas, E.; Draziotis, K.A.; Papadopoulou, A. Solutions of Hard Knapsack Problems Using Extreme Pruning. In *Modern Discrete Mathematics and Analysis: With Applications in Cryptography, Information Systems and Modeling*; Daras, N.J., Rassias, T.M., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 81–95. [CrossRef]
4. Dell'Amico, M.; Delorme, M.; Iori, M.; Martello, S. Mathematical models and decomposition methods for the multiple knapsack problem. *Eur. J. Oper. Res.* **2019**, *274*, 886–899. [CrossRef]
5. Blank, J.; Deb, K.; Mostaghim, S. Solving the bi-objective traveling thief problem with multi-objective evolutionary algorithms. In *Proceedings of the International Conference on Evolutionary Multi-Criterion Optimization*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 46–60.
6. Kumari, R.; Srivastava, K. Variable Neighbourhood Search for Bi-Objective Travelling Thief Problem. In Proceedings of the 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 4–5 June 2020; pp. 47–51.
7. Wu, J.; Polyakovskiy, S.; Wagner, M.; Neumann, F. Evolutionary computation plus dynamic programming for the bi-objective travelling thief problem. In Proceedings of the Genetic and Evolutionary Computation Conference, Kyoto, Japan, 15–19 July 2018; pp. 777–784.
8. Wu, J.; Wagner, M.; Polyakovskiy, S.; Neumann, F. Exact approaches for the travelling thief problem. In *Proceedings of the Asia-Pacific Conference on Simulated Evolution and Learning*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 110–121.
9. Mei, Y.; Li, X.; Yao, X. Improving efficiency of heuristics for the large scale traveling thief problem. In *Proceedings of the Asia-Pacific Conference on Simulated Evolution and Learning*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 631–643.
10. Polyakovskiy, S.; Bonyadi, M.R.; Wagner, M.; Michalewicz, Z.; Neumann, F. A comprehensive benchmark set and heuristics for the traveling thief problem. In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, Vancouver, BC, Canada, 12–16 July 2014; pp. 477–484.
11. Martello, S.; Pisinger, D.; Toth, P. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Manag. Sci.* **1999**, *45*, 414–424. [CrossRef]
12. Moeini, M.; Schermer, D.; Wendt, O. A hybrid evolutionary approach for solving the traveling thief problem. In *Proceedings of the International Conference on Computational Science and Its Applications*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 652–668.
13. Ali, H.; Rafique, M.Z.; Sarfraz, M.S.; Malik, M.S.A.; Alqahtani, M.A.; Alqurni, J.S. A novel approach for solving travelling thief problem using enhanced simulated annealing. *PeerJ Comput. Sci.* **2021**, *7*, e377. [CrossRef] [PubMed]
14. Maity, A.; Das, S. Efficient hybrid local search heuristics for solving the travelling thief problem. *Appl. Soft Comput.* **2020**, *93*, 106284. [CrossRef]
15. Namazi, M.; Sanderson, C.; Newton, M.A.H.; Sattar, A. Surrogate Assisted Optimisation for Travelling Thief Problems. In Proceedings of the Thirteenth International Symposium on Combinatorial Search, SOCS 2020, Vienna, Austria, 26–28 May 2020; Harabor, D., Vallati, M., Eds.; AAAI Press: Palo Alto, CA, USA, 2020; pp. 111–115.
16. Nikfarjam, A.; Neumann, A.; Neumann, F. Evolutionary Diversity Optimisation for the Traveling Thief Problem. In Proceedings of the Genetic and Evolutionary Computation Conference GECCO '22, Boston, MA, USA, 9–13 July 2022; Association for Computing Machinery: New York, NY, USA, 2022; pp. 749–756. [CrossRef]
17. Nikfarjam, A.; Neumann, A.; Neumann, F. On the Use of Quality Diversity Algorithms for the Traveling Thief Problem. In Proceedings of the Genetic and Evolutionary Computation Conference GECCO '22, Boston, MA, USA, 9–13 July 2022; Association for Computing Machinery: New York, NY, USA, 2022; pp. 260–268. [CrossRef]
18. Sanchez, M.; Cruz-Duarte, J.M.; Ortiz-Bayliss, J.C.; Ceballos, H.; Terashima-Marin, H.; Amaya, I. A Systematic Review of Hyper-Heuristics on Combinatorial Optimization Problems. *IEEE Access* **2020**, *8*, 128068–128095. [CrossRef]
19. Pillay, N.; Qu, R. *Hyper-Heuristics: Theory and Applications*; Natural Computing Series; Springer International Publishing: Cham, Switzerland, 2018. [CrossRef]
20. El Yafrani, M.; Martins, M.; Wagner, M.; Ahiod, B.; Delgado, M.; Lüders, R. A hyperheuristic approach based on low-level heuristics for the travelling thief problem. *Genet. Program. Evolvable Mach.* **2018**, *19*, 121–150. [CrossRef]
21. Ali, F.; Mohamedkhair, M. Hyper-Heuristic Approaches for the Travelling Thief Problem. In Proceedings of the 2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE), Khartoum, Sudan, 26 February–1 March 2021; pp. 1–6. [CrossRef]
22. Ortiz-Bayliss, J.C.; Terashima-Marín, H.; Conant-Pablos, S.E. Combine and conquer: An evolutionary hyper-heuristic approach for solving constraint satisfaction problems. *Artif. Intell. Rev.* **2016**, *46*, 327–349. [CrossRef]
23. Cowling, P.; Kendall, G.; Soubeiga, E. A Hyperheuristic Approach to Scheduling a Sales Summit. In *Practice and Theory of Automated Timetabling III*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 176–190. [CrossRef]

24. Drake, J.H.; Kheiri, A.; Özcan, E.; Burke, E.K. Recent advances in selection hyper-heuristics. *Eur. J. Oper. Res.* **2020**, *285*, 405–428. [CrossRef]
25. Cruz-Duarte, J.M.; Amaya, I.; Ortiz-Bayliss, J.C.; Conant-Pablos, S.E.; Terashima-Marín, H.; Shi, Y. Hyper-Heuristics to Customise Metaheuristics for Continuous Optimisation. *Swarm Evol. Comput.* **2021**, *66*, 100935. [CrossRef]
26. Amaya, I.; Ortiz-Bayliss, J.C.; Rosales-Pérez, A.; Gutiérrez-Rodríguez, A.E.; Conant-Pablos, S.E.; Terashima-Marín, H.; Coello Coello, C.A. Enhancing Selection Hyper-Heuristics via Feature Transformations. *IEEE Comput. Intell. Mag.* **2018**, *13*, 30–41. [CrossRef]
27. Lara-Cardenas, E.; Silva-Galvez, A.; Ortiz-Bayliss, J.C.; Amaya, I.; Cruz-Duarte, J.M.; Terashima-Marin, H. Exploring Reward-based Hyper-heuristics for the Job-shop Scheduling Problem. In Proceedings of the 2020 IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, ACT, Australia, 1–4 December 2020; pp. 3133–3140. [CrossRef]
28. Silva-Gálvez, A.; Lara-Cárdenas, E.; Amaya, I.; Cruz-Duarte, J.M.; Ortiz-Bayliss, J.C. A Preliminary Study on Score-Based Hyper-heuristics for Solving the Bin Packing Problem. In *Pattern Recognition: 12th Mexican Conference, MCPR 2020, Morelia, Mexico, 24–27 June 2020, Proceedings*; Springer: Cham, Switzerland, 2020; pp. 318–327. [CrossRef]
29. Silva-Galvez, A.; Orozco-Sanchez, J.; Lara-Cardenas, E.; Ortiz-Bayliss, J.C.; Amaya, I.; Cruz-Duarte, J.M.; Terashima-Marin, H. Discovering Action Regions for Solving the Bin Packing Problem through Hyper-heuristics. In Proceedings of the 2020 IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, ACT, Australia, 1–4 December 2020; pp. 822–828. [CrossRef]
30. Sanchez-Diaz, X.F.C.; Ortiz-Bayliss, J.C.; Amaya, I.; Cruz-Duarte, J.M.; Conant-Pablos, S.E.; Terashima-Marin, H. A Preliminary Study on Feature-independent Hyper-heuristics for the 0/1 Knapsack Problem. In Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC), Glasgow, UK, 19–24 July 2020; pp. 1–8. [CrossRef]
31. Sanchez, M.; Cruz-Duarte, J.M.; Ortiz-Bayliss, J.C.; Amaya, I. Sequence-Based Selection Hyper-Heuristic Model via MAP-Elites. *IEEE Access* **2021**, *9*, 116500–116527. [CrossRef]
32. Wagner, M. Stealing items more efficiently with ants: A swarm intelligence approach to the travelling thief problem. In *Proceedings of the International Conference on Swarm Intelligence*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 273–281.
33. Pisinger, D. Where are the hard knapsack problems? *Comput. Oper. Res.* **2005**, *32*, 2271–2284. [CrossRef]
34. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by simulated annealing. *Science* **1983**, *220*, 671–680. [CrossRef] [PubMed]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.