algorithms

MDPI

Article

Hyper-Heuristic Based on ACO and Local Search for Dynamic Optimization Problems †

Felipe Martins Müller [1,*,‡] and Iaê Santos Bonilha [2,‡]

[1] Department of Applied Computing, Federal University of Santa Maria, Santa Maria 97105-900, Brazil
[2] PPGEP, Federal University of Santa Maria, Santa Maria 97105-900, Brazil; iaesb@hotmail.com
* Correspondence: felipe@inf.ufsm.br; Tel.: +55-55-99160-7777

† This paper is an extended version of our paper published in the proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC 2019), Wellington, New Zealand, 10–13 June 2019.

‡ These authors contributed equally to this work.

Abstract: Hyper-heuristics comprise a set of approaches that are motivated (at least in part) by the objective of intelligently combining heuristic methods to solve hard optimization problems. Ant colony optimization (ACO) algorithms have been proven to deal with Dynamic Optimization Problems (DOPs) properly. Despite the good results obtained by the integration of local search operators with ACO, little has been done to tackle DOPs. In this research, one of the most reliable ACO schemes, the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS), has been integrated with advanced and effective local search operators, resulting in an innovative hyper-heuristic. The local search operators are the Lin–Kernighan (LK) and the Unstringing and Stringing (US) heuristics, and they were intelligently chosen to improve the solution obtained by ACO. The proposed method aims to combine the adaptation capabilities of ACO for DOPs and the good performance of the local search operators chosen in an adaptive way and based on their performance, creating in this way a hyper-heuristic. The travelling salesman problem (TSP) was the base problem to generate both symmetric and asymmetric dynamic test cases. Experiments have shown that the $\mathcal{MM}$AS provides good initial solutions to the local search operators and the hyper-heuristic creates a robust and effective method for the vast majority of test cases.

Keywords: hyper-heuristic; ant colony optimization; local search operators; dynamic travelling salesman problem

Citation: Müller, F.M.; Bonilha, I.S. Hyper-Heuristic Based on ACO and Local Search for Dynamic Optimization Problems. *Algorithms* **2022**, *15*, 9. https://doi.org/10.3390/a15010009

Academic Editors: Conor Ryan and Frank Werner

Received: 9 November 2021
Accepted: 22 December 2021
Published: 24 December 2021

1. Introduction

Hyper-heuristics comprise a set of approaches with the common goal of automating design and tuning heuristic methods to solve hard computational search problems. Their main goal is to produce applicable search methodologies more broadly. Hyper-heuristics' distinguishing feature is that they operate on a search space of heuristics (or heuristic components) rather than directly on the search space of solutions to the underlying problem, as happens with most meta-heuristic approaches [1]. For an updated survey about hyper-heuristics for Dynamic Optimization Problems (DOPs), see [2].

Ant colony optimization (ACO) algorithms have proved that they are capable to find the optimal (or near optimal) solution for difficult combinatorial optimization problems (e.g., the static travelling salesman problem (STSP) [3]). In this research, we utilize the travelling salesman problem (TSP) as the base problem to generate both symmetric and asymmetric dynamic test cases. The STSP has been studied extensively for the last few decades [4] and it is one of the most challenging $\mathcal{NP}$-complete combinatorial optimization problems. As a whole, literature publications have dealt only with static problems, without dynamic changes, i.e., the instances do not change during the problem solving. However, several real-world applications experience changes during the optimization process (e.g., traffic restrictions), making the problem even more difficult. DOPs are defiant

because they need to quickly search for the best solution to the problem while dealing with changes in the search space [5]. A dynamic change may involve many factors, such as the objective function, input variables, problem instance, search space, and constraints. The traveling salesman problem with varying parameters are referred to as dynamic traveling salesman problem (DTSP). Many real-life applications can be modelled like a DTSP, e.g., transportation managers need to handle stochasticity and dynamicity in designing and managing a route plan.

DTSP appears in some studies, and approaches are proposed to solve it efficiently. The authors of [6] say that DTSPs are more difficult than STSP and also point out that traditional methods to solve STSPs may not be able to find answers for DTSPs. Therefore, the study of metaheuristic optimization algorithms such as swarm and evolutionary algorithms has taken place to solve DTSPs. The studies on this subject matter concentrate primarily on improving metaheuristics — formerly developed to solve conventional static optimization problems. Some examples of these algorithms include but are not limited to Artificial Immune Systems (AISs) [7], ant colony optimization algorithms [8–13], Genetic Algorithms (GAs) [14–16], and Particle Swarm Optimization algorithms (PSOs) [17,18]. Since ACOs can adapt to dynamic changes by transferring knowledge from past environments, ACOs naturally combine with other techniques. Moreover, several ACOs features such as adaptability, instance ease of use, and efficiency make ACO-based approaches popular among researchers to solve DTSPs [11]. The authors of [19] propose a rank-based ACO algorithm to solve DTSP. The authors of [9] propose a memetic algorithm for symmetric DTSPs that was extended to solve asymmetric DTSPs [10]. The authors of [20] propose some pheromone modification strategies and investigate their performance in solving DTSPs. The authors of [21] propose an ACO algorithm for DTSPs with a technique, named Shaking, to smoothen the pheromone trails after a dynamic change. The authors of [22] propose two multi-colony ACO algorithms to solve a DTSP. The authors of [23] propose a memory-based ACO algorithm, namely, P-ACO, with the disadvantage that the algorithm may generate the same ants multiple times, worsening the computational performance. In order to avoid redundancy in the ant population list, different extensions of P-ACO have been proposed, including Fitness Sharing P-ACO (FS-PACO) [24], Simple Crowding P-ACO (SC-PACO) [25], immigrant schemes with ACO [26], and Memetic ACO (M-PACO) [12] algorithms. The authors of [27] propose adaptive large neighborhood search (ALNS)-based immigrant schemes for DTSPs.

Something that stands out is that in the previous articles, DTSPs instances were generated based on a very small number of TSP instances. From our point of view, this can hide important aspects in the results. For this reason, we have used a wider range of TSP base instances to generate our DSTPs. We have also observed that the ACO used as the base for our hyper-heuristic presents one of the best performances in the literature, allowing us to compare between their variants without loss of generality. This paper extends the existing literature by developing an enhanced ACO-based metaheuristic to solve a DTSP. Our efforts lie in developing a hyper-heuristic based on ACO and local search, aiming to solve the DTSP efficiently.

The integration of local search operators with metaheuristics showed significant improvements in the solution quality for static optimization problems [28–31] and, more recently, for DOPs [8,10]. The Unstringing and Stringing (US) [32] and Lin–Kernighan (LK) [33,34] local search operators have demonstrated promising performance in ACO-based memetic algorithms for DOPs [8–10]. This happens because the local search operators take advantage of the global search capabilities of ACO to guide them in neighbourhoods that contain high-quality solutions, while ACO takes advantage of the local search operators to better explore the neighbourhood locally. However, the local search operators' performance often depends on the problem instance and in many cases on the dynamic changes of the DOPs [8]. In this research, a hyper-heuristic [1] is used to enable the selection of the local search operator (either LK or US) that can better contribute to the solution quality of the DTSP.

This paper is an extended version of our paper [8] that presents only a local search operator (LSO), improving the best solution found by an Ant Colony Optimisation framework, and compares the use of two of these local search operators (named Lin–Kernighan and Unstring and Stringing) independently. The hyper-heuristic proposed here (called HULK — Hyper-heuristic with Us and LK as local search operators) is very different from the congress paper due to the intelligence applied in the choice of LSOs and the order in which they are applied that make use of a self-adaptive function. There is another huge difference in the instances and in the frequency of dynamic changes, i.e., the instances used here allow arc blocking, and the frequency of dynamic changes has dropped from 10^4 to 100, making HULK faster without losing quality. This new HULK is able to outperform the methods presented in the congress paper, and a whole set of new results are presented here. The HULK improves its ability to cope with the best features of LK and US, producing much better solutions, mainly in the asymmetric dynamic change environment. For the purpose of keeping a self-sufficient article, the explanations of the LSOs are the same as in the congress paper, and we also present some results to allow for a better understanding of the computational experiments. The paper highlights can be listed as follows:

- We integrate one of the best ACO variations with advanced and effective local search operators, i.e., the Lin–Kernighan and the Unstringing and Stringing heuristics, resulting in a powerful hyper-heuristic (HULK).
- The proposed HULK combines the adaptation capabilities of ACO for DOPs and the superior performance of the local search operators. This is done with a smart and self-adaptive way to choose the LSO that is applied using a weighted roulette wheel, based on the objective function value of the previous solutions.
- We include arc blocking and dropped the frequency of dynamic changes without losing quality.
- The proposed method can provide better solutions, especially in asymmetric dynamic test cases.

We use the TSP as the base problem to systematically generate dynamic test cases [11]. In the literature, most cases of dynamic changes are symmetric [9,11,35,36]. However, in the real world they are often asymmetric [8,10,37]. For example, in pick-up or delivering problems, the travel time to arrive in some points can drastically change depending on the direction travelled due to traffic conditions. For the sake of comparison, both symmetric and asymmetric dynamic changes are considered. The rest of the paper is organized as follows. Section 2 describes the TSP and the proposed dynamic test cases. Section 3 describes the hyper-heuristic method using one of the best variations of ACO as the base. To make the paper self-contained, we also provide the core logic of the two local search operators. Section 4 shows the experimental results and analysis. Finally, the conclusions are presented in Section 5.

2. Proposed Dynamic Changes

2.1. Base Problem Formulation

As mentioned before the TSP is used as the base problem to generate dynamic test cases. Typically, a TSP instance is modelled by a fully connected weighted graph $G = (N, A)$, where $N = \{1, \ldots, n\}$ is a set of n nodes and $A = \{(i, j) \mid i, j \in N, i \neq j\}$ is a set of arcs. Each arc $(i, j) \in A$ is associated with a non-negative value $w_{ij} \in \Re^+$, which for the classic TSP represents the distance between nodes i and j. In the case of such arc, in which it is blocked or does not exist, we set its associated value as a very large number. In the symmetric TSPs, we have $w_{ij} = w_{ji}$ for every pair of nodes. When $w_{ij} \neq w_{ji}$ for at least one pair of nodes, the TSP becomes asymmetric. Every problem instance consists of a weight matrix, $W = \{w_{ij}\}_{n \times n}$, containing all the weights associated with the arcs of the corresponding graph G. A classical mathematical model for TSP was proposed by [38] and later modified and relaxed, so it was easier for the authors of [39] to solve, and it

was recently used by [40] . The formulation of the TSP is an integer linear programming utilizing n^2 binary variables x_{ij}. The variable x_{ij} is defined as:

$$x_{ij} = \begin{cases} 1, \text{ if and only if arc } (i,j)(i = 1,\ldots,n; j = 1,\ldots,n) \text{ is in the optimal tour} \\ 0, \text{ otherwise} \end{cases} \quad (1)$$

The objective function is to minimize the route cost, as shown in Equation (2).

$$min \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} w_{ij} x_{ij} \quad (2)$$

$$subject\ to \sum_{i=1, i\neq j}^{n} x_{ij} = 1, \quad j = 1,\ldots,n \quad (3)$$

$$\sum_{j=1, j\neq i}^{n} x_{ij} = 1, \quad i = 1,\ldots,n \quad (4)$$

$$\sum_{i\in S} \sum_{j\in S, j\neq i} x_{ij} \leq |S| - 1, \quad S \subset N \ : \ S \neq \oslash \quad (5)$$

$$x_{ij} \in \{0,1\} \quad i,j = 1,\ldots,n, \ i \neq j \quad (6)$$

where constraints (3) and (4) impose that the in-degree and out-degree of each vertex, respectively, is equal to one, while constraints (5) are sub-tour elimination constraints and impose that no partial circuit exists. The variables domain are expressed by constraints (6).

2.2. Generating Dynamic Test Cases

To transform the problem instance in a dynamic instance, we modify the weight matrix as follows [11]:

$$W(T) = \{w_{ij}(T)\}_{n\times n}, \quad (7)$$

where $T = \lceil t/f \rceil$ is the period of a dynamic change, t is the counter of algorithmic evaluations, and f is the frequency in which changes occur. Note that, since the dynamic changes are synchronized with the optimization process, the parameter f is expressed in algorithmic steps.

A specific dynamic test case can be generated by blocking a set of arcs and/or assigning an increasing or decreasing factor value to the arc connecting nodes i and j as follows:

$$w_{ij}(T+1) = \begin{cases} w_{ij}(0) + MAX, & \text{if arc } (i,j) \in B_S(T), \\ w_{ij}(0) + \mathcal{R}_{ij}, & \text{if arc } (i,j) \in A_S(T), \\ w_{ij}(T), & \text{otherwise,} \end{cases} \quad (8)$$

where $B_S(T) \subset A$ the set of arcs to be blocked, MAX is a constant sufficiently large number, $w_{ij}(0)$ is the original weight of the arc (i,j) (for the static problem instance when $T = 0$), $\mathcal{R}_{ij}$ is a normally distributed random number (with zero mean and standard deviation set to $0.2 \cdot w_{ij}(0)$ [37]) that defines the modified factor value of the arc, $A_S(T) \subset A$ defines the set of arcs randomly selected for the change at that period and T defines the period index (as we can see in Equation (7)). The difference between the instances presented in [8] that does not allow arc blocking is the inclusion of the first line in the right hand side of Equation (8). The number of blocked arcs correspond to 1% of the arcs selected to dynamic change.

The size of the change set A is determined by the number of arcs ($n(n-1)/2$ for symmetric and $n(n-1)$ for asymmetric cases). Consequently, the size of $A_S(T)$ is defined by the size of A and the extent of change (i.e., $m \in [0,1]$). For example, in a period T, precisely $\lceil mn(n-1)/2 \rceil$ and $\lceil mn(n-1) \rceil$ arcs are selected to have their weights changed in symmetric and asymmetric cases, respectively. The higher the m value, the more arcs will change value. Recalling that, if w_{ij} changes in symmetric cases, then w_{ji} changes uniformly

(i.e., $w_{ji} = w_{ij}$). On the contrary, when w_{ij} changes in asymmetric cases, w_{ji} will not change unless arc (j, i) is selected for a change (but not necessarily uniformly with w_{ij}). The same occurs when blocking arcs are introduced in order to maintain the symmetry or asymmetry of the problem.

A particular solution $\pi = [\pi_1, \ldots, \pi_n]$ in the search space is specified by a permutation of the node indices, and for the dynamic TSP (DTSP), it is evaluated as follows see Equation (9) :

$$\phi(\pi, t) = w_{\pi_n \pi_1}(T) + \sum_{i=1}^{n-1} w_{\pi_i \pi_{i+1}}(T). \tag{9}$$

3. HULK: Hyper-Heuristic Based on ACO and Local Search Operators

The hyper-heuristic presented in this paper is integrated with an ACO metaheuristic. ACO consists of a colony of ω ants that construct solutions and share their information among each other via their pheromone trails. One of the best performing ACO variations, named $\mathcal{MAX}$-$\mathcal{MIN}$ AS ($\mathcal{MM}$AS) [41], which used in the hyper-heuristic framework shown in Algorithm 2. Notice that in [8], the step of choosing the LSOs does not exist since the LSO is picked before the algorithm starts. We will use the TSP notation to describe the hyper-heuristic based on ACO framework and LSOs.

3.1. Building Solutions Inside ACO

Ants utilize information of the pheromone trail to build solutions and reinforce information on the pheromone trail to mark their path. The probability in which ant k, currently at node i, will move to node j is calculated as follows:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \text{if } j \in \mathcal{N}_i^k, \tag{10}$$

where τ_{ij} and η_{ij} are the existing pheromone trail and the heuristic information available a priori between nodes i and j, respectively. The pheromone trails are initialized evenly with the same value τ_0. The heuristic information is calculated as $\eta_{ij} = 1/w_{ij}(T)$ where $w_{ij}(T)$ is defined as in Equation (7). $\mathcal{N}_i^k$ is the set of unvisited nodes for ant k adjacent to node i. α and β are the two parameters that determine the relative influence of τ_{ij} and η_{ij}, respectively.

3.2. Choosing and Applying Local Search Operator

Stützle and Hoos [41] applied local search operators to the iteration-best ant of $\mathcal{MM}$AS after every iteration, while in [29], they further applied local search operators to all ants. However, local search operators are computationally expensive methods; thus, this massive usage may not be very efficient for DTSP because of the natural increase of computation effort. As previously discussed, for DTSPs, algorithms must produce high-quality solutions quickly (essentially before the next dynamic change occurs) [37]. In [8,10], the local search operator is applied to the best-so-far ant of $\mathcal{MM}$AS only when a new best solution is found. This is because local search operators are typically executed until no further improvement is possible. However, in this research, as there may be the exchange of LSO there are cases where it is interesting to apply LSO even when there is no improvement in the best solution so far. These exchanges between LSO can change the pheromone trail guiding the search to promising places not yet explored. We investigate two powerful local search heuristics designed for the TSP, LK, and US heuristics that were adapted to cope with both symmetric and asymmetric cases; we describe them in Sections 3.2.1 and 3.2.2, respectively.

The proposed hyper-heuristic is designed to select how and when to apply the local search operator. As previously discussed, each time the ACO finds an improved solution, a local search heuristic is applied to that solution for further improvement, but the

hyper-heuristic can also apply LSOs to avoid solution stagnation and to perform a sort of diversification in the search space; the whole hyper-heuristic framework is presented in Algorithm 2. To decide which local search will be applied, we use a weighted roulette wheel with choice probabilities of initially equal to 0.5 for both and updated as shown in Equation (11) if US was chosen, and in Equation (12) if LK was chosen. This idea is similar to Genetic Algorithms, where the chromosomes with better fitness value (LSO with better performance) have greater chances to picked. $averUS$ and $averLK$ will be the average improvement of up to three last-solution values produced by US and LK, respectively. We need to make sure that this adjustment was only made if both operators had already been selected at least once and the average of the solutions found by the algorithm chosen was less than the other (minimization problem), i.e., $averUS < averLK$ if US is chosen, and $averLK < averUS$ if LK was chosen. This update attempts to prevent the choice of only one of the operators available prematurely; also, γ, and δ are limited in the range $[0.1, 0.9]$. Observe that HULK starts with the same chance of selecting both LSO, but during the search, the percentages are self-adjusted in favor of the LSO with better performance, and this procedure is easily adapted if there are more than two LSOs or if we consider other DOPs. To choose the LSO, we first generate a random number uniformly distributed in the interval $[0.0, 1.0]$, and if this number is less than γ, US is chosen. Otherwise, we choose LK. The integration of the LSO into the hyper-heuristic is made through the procedure ApplyLSO(π^{ib}, A, $DistanceMatrix$), where π is the current solution and A is the LSO. This procedure is detailed in Algorithm 1.

$$\gamma = \gamma + \left(1 - \frac{averUS}{averLK}\right)\delta \text{ and } \delta = \delta - \left(1 - \frac{averUS}{averLK}\right)\delta \text{ , if US was chosen} \quad (11)$$

$$\delta = \delta + \left(1 - \frac{averLK}{averUS}\right)\gamma \text{ and } \gamma = \gamma - \left(1 - \frac{averLK}{averUS}\right)\gamma \text{ , if LK was chosen} \quad (12)$$

Algorithm 1 Procedure for Applying LSO

```
1: ApplyLSO(π^ib, A, DistanceMatrix)
2: if (A = US) then
3:    ApplyLocalSearchUS(π^ib)
4:    if (Solution was improved?) then
5:       Update γ and δ as show in Equation (11)
6:    end if
7: else
8:    ApplyLocalSearchLK(π^ib)
9:    if (Solution was improved?) then
10:      Update γ and δ as show in Equation (12)
11:   end if
12: end if
13: OUTPUT: π^ib   %best TSP solution after local search
```

3.2.1. Lin–Kernighan Local Search Operator

The LK heuristic performs a series of k-opt moves to transform a TSP tour into a shorter one [33], where k-opt move consists of the exchange of a set of k tour arcs by a set of k new arcs keeping the tour feasibility. The algorithm has an efficient implementation [34] with open code but with so many modifications over time that are very difficult to use and adapt; however, here we successfully made it also include the ability to deal with asymmetry. The LK heuristic starts with two empty arc sets: X (i.e., *out*-arcs) and Y (i.e., *in*-arcs). At each step, one arc that currently belongs to the tour will be added to X and a new arc that does not belong to the tour will be added to Y. After the first step, the LK heuristic will favour arc insertions that result in a shorter complete TSP tour. When a new

complete tour is achieved, the algorithm will begin a new phase of arc exchanges and this process will continue until there is no further improvement.

LK establishes a series of rules to be followed, aimed at enhancing performance. These rules can be summarized as follows:

- Each arc removed must share a node with its added counterpart. After the first arc exchange in each cycle, each arc being removed must also share a node with the previously added arc. Figure 1 illustrates an example of a 2-*opt* move, where in the first step arc (V_1,V_2) is removed and arc (V_2,V_4), which shares the node V_2 with its removed counterpart, is added. In the second step arc (V_3,V_4), which shares node V_4 with the previously added arc, is removed and arc (V_3,V_1) is added, closing the tour.
- No exchanges that result in the tour being broken into multiple closed circuits are allowed. An example of this type of exchange is shown in Figure 2, where arcs (V_1,V_2) and (V_3,V_4) are removed and arcs (V_4,V_1) and (V_3,V_2) are added. In this case, the addition of any arcs would not be accepted because it would result in a segment of tour forming a cycle.
- Each pair of arcs exchanged must be gainful, meaning that each arc being added must be shorter than its removed counterpart. If the problem is asymmetric, both orientations of the resulting tour must be analysed.
- Once an arc is removed, it cannot be reinserted until the tour is closed.

Although LK was originally designed for symmetric problems [33], it can easily be modified to address asymmetric problems by transforming an asymmetric weight matrix into a symmetric, normally by doubling the graph nodes [34,42].

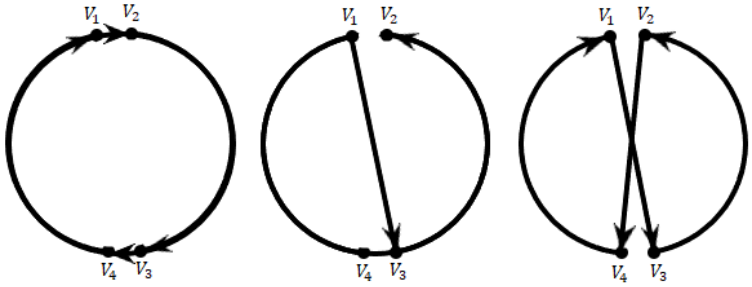

Figure 1. An example of 2–*opt* move.

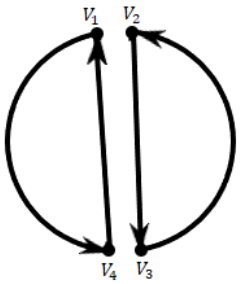

Figure 2. A forbidden movement resulting in closed circuits (cycles).

3.2.2. Unstringing and Stringing Local Search Operator

The US heuristic is the core part of the Generalized Insertion Procedure-Unstringing and Stringing (GENIUS) [32]. The US repeatedly removes (unstringing) and inserts (stringing) nodes searching for solution improvements. The main feature of the algorithm is that the stringing nodes can be made between non-adjacent nodes, resulting in a route where both nodes become adjacent to the node being inserted. All movements are based on local and global neighbourhoods. The asymmetric version of US was proposed and implemented in [43], considering the node V_x to be inserted between any two nodes V_i and V_j . For a given orientation of a tour, consider V_k a node in the sub tour from V_j to V_i,

and V_l a node in the sub tour from V_i to V_j. We also consider for any node V_h on the tour, V_{h+1} its successor and V_{h-1} its predecessor. The re-insertion of V_x between V_i and V_j can be done in several ways using different types of insertions and removals. Since the potential number of choices for V_i, V_j and V_x could be large, we use a candidate set based on the closest neighbours. Once V_x is chosen, its neighbours will be the q nodes that have lower value arcs with V_x. To choose the best place to insert a node in the tour, we look only for the neighbourhood of each node involved with stringing or unstringing movements. In [9,32], only symmetric problem instances were considered and tackled with Type I and Type II removals (Figure 3a,b) and Type I and Type II insertions (Figure 4a,b). In [8,10,43], other two types of removals and other two types of insertions were considered in order to tackle asymmetric problem instances: Type III and Type IV removals (Figure 3c,d), and Type III and Type IV insertions (Figure 4c,d).

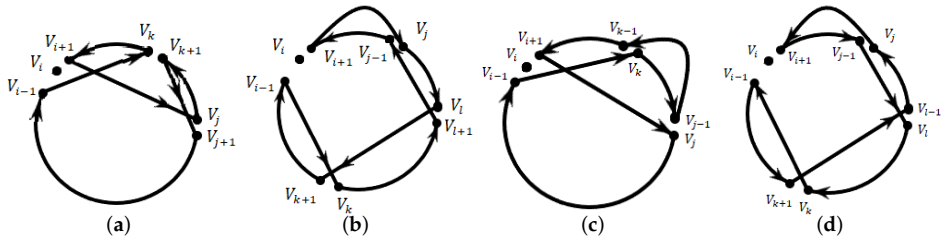

Figure 3. (**a**) Unstringing Type I, (**b**) unstringing II, (**c**) unstringing Type III, and (**d**) unstringing Type IV.

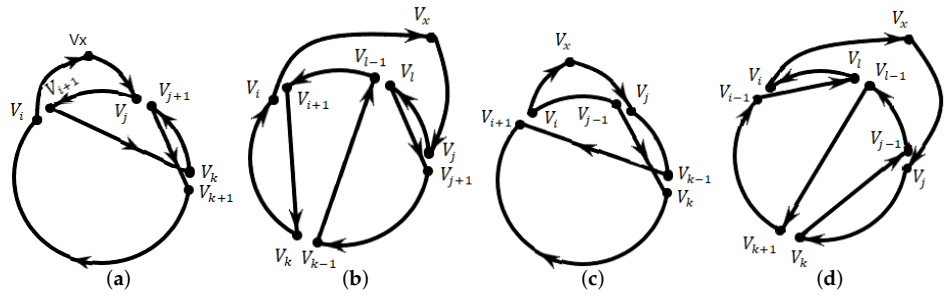

Figure 4. (**a**) Stringing Type I, (**b**) stringing Type II, (**c**) stringing Type III, and (**d**) stringing Type IV.

The unstringing procedure removes a node from the tour and reconnects the tour with the remaining nodes trying to obtain an improved closed tour. The procedure consists of four types of removals defined as follows:

- Unstringing Type I: consider V_j belonging to the neighbourhood of V_{i+1} and V_k belonging to the neighbourhood of V_{i-1}, with V_k being part of the sub tour $(V_{i+1}, \ldots, V_{j-1})$. The removal of node V_i results in the deletion of arcs (V_{i-1}, V_i), (V_i, V_{i+1}), (V_k, V_{k+1}) and (V_j, V_{j+1}) and the insertion of arcs (V_{i-1}, V_k), (V_{i+1}, V_j) and (V_{k+1}, V_{j+1}). Additionally, the sub tours $(V_{i+1}, \ldots, V_k)$ and $(V_{k+1}, \ldots, V_j)$ are reversed.
- Unstringing Type II: consider V_j belonging to the neighbourhood of V_{i+1}, V_k belonging to the neighbourhood of V_{i-1}, with V_k being part of the subtour $(V_{j+1}, \ldots, V_{i-2})$, and V_l belonging to the neighbourhood of V_{k+1}, with V_l being part of the sub tour $(V_j, \ldots, V_{k-1})$. The removal of node V_i results in the deletion of arcs (V_{i-1}, V_i), (V_i, V_{i+1}), (V_{j-1}, V_j), (V_k, V_{k+1}) and (V_l, V_{l+1}) and the insertion of arcs (V_{i-1}, V_k), (V_{l+1}, V_{j-1}), (V_{i+1}, V_j) and (V_l, V_{k+1}). As before, the sub tours $(V_{i+1}, \ldots, V_{j-1})$ and $(V_{l+1}, \ldots, V_k)$ are reversed.
- Unstringing Type III: consider V_j belonging to the neighborhood of V_{i+1} and V_k belonging to the neighborhood of V_{i-1} with V_k being part of the sub tour $(V_{i+1}, \ldots, V_{j-1})$. The removal of node V_i results in the deletion of arcs (V_{i-1}, V_i), (V_i, V_{i+1}), (V_{j-1}, V_j),

and (V_{k-1}, V_k) and the insertion of arcs (V_{i+1}, V_j), (V_{i-1}, V_k), and (V_{j-1}, V_{k-1}). As before, the sub tours $(V_{j-1}, \ldots, V_k)$ and $(V_{i-1}, \ldots, V_j)$ are reversed.

- Unstringing Type IV: consider V_j belonging to the neighborhood of V_{i+1}, V_k belonging to the neighborhood of V_{i-1} with V_k being part of the sub tour $(V_{l+1}, \ldots, V_{i-2})$, and V_l belonging to the neighborhood of V_{j-1} with V_l being part of the sub tour $(V_{j+1}, \ldots, V_{k-1})$. The removal of node V_i results in the deletion of arcs (V_{i-1}, V_i), (V_i, V_{i+1}), (V_{j-1}, V_j), (V_{l-1}, V_l), and (V_k, V_{k+1}) and the insertion of arcs (V_k, V_{i-1}), (V_{j-1}, V_l), (V_{k+1}, V_{l-1}), and (V_j, V_{i+1}). As before, the sub tours $(V_{k+1}, \ldots, V_{i-1})$ and $(V_j, \ldots, V_{l-1})$ are reversed.

The stringing procedure can be seen as the reverse of the unstringing procedure and consists of four types of insertions as follows:

- Stringing Type I: Assuming $V_k \neq V_i$ and $V_k \neq V_j$. The insertion of V_x results in the deletion of arcs (V_i, V_{i+1}), (V_j, V_{j-1}) and (V_k, V_{k+1}) and the insertion of arcs (V_i, V_x), (V_x, V_j), (V_{i+1}, V_k) and (V_{j+1}, V_{k+1}). Additionally, the sub tours $(V_{i+1}, \ldots, V_j)$ and $(V_{j+1}, \ldots, V_k)$ are reversed.
- Stringing Type II: assuming $V_k \neq V_j$, $V_k \neq V_{j+1}$, $V_l \neq V_i$, and $V_l \neq V_{i+1}$. The insertion of V_x results in the deletion of arcs (V_i, V_{i+1}), (V_{l-1}, V_l), (V_j, V_{j+1}) and (V_{k-1}, V_k) and the insertion of arcs (V_i, V_x), (V_x, V_j), (V_l, V_{j+1}), (V_{k-1}, V_{l-1}) and (V_{i+1}, V_k). As before, the sub tours $(V_{i+1}, \ldots, V_{l-1})$ and $(V_l, \ldots, V_j)$ are reversed.
- Stringing Type III: this stringing type can be seen as the inverse of Stringing Type I. Notice that when node V_x is inserted between V_i and V_j, the sub tour of nodes is rearranged in such a way that almost the entire sequence is reversed. The objective is to explore other promising regions of the search space. As in Stringing Type I, assume that $V_k \neq V_i$ and $V_k \neq V_j$. The insertion of V_x results in the deletion of arcs (V_{i-1}, V_i), (V_{j-1}, V_j) and (V_{k-1}, V_k) and the insertion of arcs (V_i, V_x), (V_x, V_j), (V_k, V_{j-1}) and (V_{k-1}, V_{i-1}). As before, the sub tours $(V_i, \ldots, V_{j-1})$ and $(V_k, \ldots, V_{i-1})$ are reversed.
- Stringing Type IV: similarly, this type of insertion can be seen as the reverse of Stringing Type II. As in Stringing Type II, assume that $V_k \neq V_j$, $V_k \neq V_{j+1}$, $V_l \neq V_i$ and $V_l \neq V_{i+1}$. The insertion of V_x results in the deletion of arcs (V_{i-1}, V_i), (V_l, V_{l+1}), (V_{j-1}, V_j) and (V_k, V_{k+1}) and the insertion of arcs (V_i, V_x), (V_x, V_j), (V_{i-1}, V_l), (V_{l+1}, V_{k+1}) and (V_k, V_{j-1}). As above, the sub tours $(V_i, \ldots, V_l)$ and $(V_{l+1}, \ldots, V_{j-1})$ are reversed.

3.3. Pheromone Trail Update

One of the most important steps on ACO algorithms is how to adjust the pheromone trail to guide the search to promising solutions in the search space as well to forget misleading paths. The pheromone trail plays an important role for HULK since it maintains relevant information about previous dynamic environments, accelerating the solution search in the new dynamic environment or when applying a different LSO. The pheromone adjustments in $\mathcal{MM}$AS are made in the same way as in [8], i.e., the evaporation is commanded by Equation (13).

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij}, \forall (i, j) \in A, \tag{13}$$

where ρ is the evaporation rate, satisfying $0 < \rho \leq 1$, and τ_{ij} is the current pheromone value. After evaporation, the best ant updates the pheromone trail by adding pheromone as described in Equation (14).

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best}, \forall (i, j) \in \pi^{best}, \tag{14}$$

where $\Delta\tau_{ij}^{best} = 1/\phi(\pi^{best}, t)$ is the amount of pheromone deposited by the best ant. The pheromone can be deposited by the best-so-far ant (i.e., a special ant that may not necessarily belong to the current constructing colony), in this case, $\pi^{best} = \pi^{bs}$, or by the iteration-best ant, in this case, $\pi^{best} = \pi^{ib}$. These ants allow a transition from early stronger exploration to later stronger exploitation of the search space. To better explain this process, consider

f^{bs} the frequency with which the best-so-far ant deposits pheromone on its trail. Consider f^{bs} to be the number of iterations performed by the algorithm between two best-so-far ant pheromone updates. This frequency is adjusted as the search progress [29] starting with a value that allows the best-so-far ant deposits pheromone at each iteration. After the first 25 iterations, we reduce f^{bs} to 5, 3, 2 and 1 at every 25 additional iterations, retaining the last value until the end of the ACO. This scheme allows that, as the search evolves, the influence of the iteration-best ant on the pheromone trail decreases while the impact of the best-so-far ant grows. This process refreshes at the start of each dynamic change. Observe that the pheromone trail is updated after the end of the local search, so the improvements obtained by the local search can be explored by the proposed hyper-heuristic in the sequence.

3.4. Keeping Solution Diversity

Diversity is a key factor when solving DOPs. Having a diverse set of solutions allows the search to quickly adapt to the new environments escaping from the outdated ones [44]. Given that the best ant is the unique one allowed to deposit pheromone on the trail, the search can be stuck in the solutions found in the first iterations. Consequently, we provide a way to once in a while reinitialize the pheromone trails to the current τ_{max} value to increase exploration. For example, the pheromone reinitialization mechanism is triggered every time stagnation occurs or no improvement in the solution is found for a given number of iterations. Stagnation is detected using λ-branching [45], which calculates the statistics regarding the distribution of the current pheromone trails.

We also impose lower and upper limits ($\tau_{min}\tau_{max}/(2n)$, where n is the number of nodes and $\tau_{max} = 1/(\rho \cdot \phi(\pi^{bs}, t))$) for the pheromone trails, keeping $p_{ij}^{k} > 0$ and giving a chance to all arcs to be selected. The τ_{max} value is updated every time a new best-so-far ant is found.

3.5. Responding to Dynamic Changes

One of the great advantages of ACO algorithms is that they can be directly applied to DOPs. This occurs because they can use the pheromone trails to keep the knowledge obtained from previous environments [36,46]. For example, normally the environment changes are not huge; therefore, the pheromone trails can retain sufficient knowledge to accelerate the optimization process of the new environment. Meanwhile, ACO must be supple to accept the knowledge provided for the former pheromone trails as well to discard that information to deal with the new environment. These actions take place through the evaporation process that quickly eliminates unpromising areas of the new environment. Thereby, the ants can continue exploring the search space for the new optimum. This is also important when the LSOs do their job. However, if the environments become completely different, i.e., the magnitude of changes is very large, re-initializing the pheromone trails is sometimes is better than transferring knowledge [36,46,47].

3.6. Integration between Algorithms in the Dynamic Test Environment

The hyper-heuristic proposed can be summarized in a running environment that allows for the intelligent integration of ACO with both LK and US algorithms. The synchronization between algorithms happens through a shared distance matrix. While running the tests, each T iteration of a perturbation was applied to the distance matrix, as described in Section 2.2. The algorithm adopted as a local search is selected in the beginning of each hyper-heuristic throughout a weighted roulette wheel. If the ACO improves the solution or does not produce an improvement solution after five or more iterations, apply local search. Additionally, if the algorithm does not produce an improvement after 0.4 T iterations (where T is the dynamic change period), apply both LSO in sequence. The first algorithm to be applied on the sequential local search is selected through the weighted roulette. The general framework for the dynamic test running environment is in Algorithm 2. Observe that this idea can be applied in a wide variety of DOPs, mainly routing problems; one recent

example for dynamic vehicle routing problem can be found in [48], where local search operators smartly repair the solution obtained by ACO.

Algorithm 2 Hyper-heuristic HULK Dynamic Test Running Environment.

Initialize parameters: T (the period of dynamic change), M (the maximum number of iterations for the test), $A_s(i)$ (the magnitude of change for the iteration i) and $\gamma, \delta \leftarrow 0.50$ (where γ is the percentage of choice of US, and δ is the percentage of choice of LK)
$i \leftarrow 0$ (the current number of iterations for the test)
$n \leftarrow 0$ (the current number of iterations without improvement for the test)
$t \leftarrow 0$ (the current period of dynamic change)
$DistanceMatrix \leftarrow$ ReadProblem()
InitializeEnvironment
InitializePheromoneTrails(τ_0)
Select A using a weighted roulette wheel (biased by γ for US and δ for LK) to be the first algorithm to interact with ACO, A' will be the LSO who was not chosen
while $(i < M)$ **do**
  if $(mod(i/T) = 0)$ **then**
    $DistanceMatrix \leftarrow$ ApplyDynamicChanges($A_s(i)$)
    $\gamma, \delta \leftarrow 0.50$
    $t \leftarrow t + 1$
  end if
  ConstructSolutions
  $\pi^{ib} \leftarrow$ FindIterationBest
  if $(mod(i/T) = 0$ and $t = 1)$ **then**
    $\pi^{bs} \leftarrow \pi^{ib}$
  end if
  if $(n >= 0.4T)$ **then**
    $\pi^{ib} \leftarrow$ ApplyLSO($\pi^{ib} A, DistanceMatrix$)
    $\pi^{ib} \leftarrow$ ApplyLSO($\pi^{ib} A', DistanceMatrix$)
  else
    if $(n >= 5$ or $(\phi(\pi^{ib}, t) < \phi(\pi^{bs}, t)))$ **then**
      $\pi^{ib} \leftarrow$ ApplyLSO($\pi^{ib}, A, DistanceMatrix$)
    end if
  end if
  PheromoneUpdate
  if $((\phi(\pi^{ib}, t) < \phi(\pi^{bs}, t)))$ **then**
    $\pi^{bs} \leftarrow \pi^{ib}$
    $n = 0$
  end if
  Select A using a weighted roulette wheel to be the algorithm to interact with ACO in the next iteration, A' will be the LSO that was not chosen
  $i \leftarrow i + 1$
  $n \leftarrow n + 1$
end while
OUTPUT: π^{bs} %best TSP solution

4. Computational Tests

4.1. Experimental Setup

During the literature review, we noticed that the computational experiments for DTSPs remain in a small number of TSP instances from TSPLib, mainly from the family `KROA` (more specifically `KROA100`, `KROA150` and `KROA200`. The `KROA` family has been proposed by [49] and came from examples of truck-dispatching problems. We compare the results in the literature with the ones obtained by HULK, considering the magnitude of change $m = 0.10$. To be fair to the literature, we first perform the tests without arc blocking with the same set of parameters useed here. We also compute the results from the family `KROA` considering arc blocking. We investigate the proposed hyper-heuristic, which cleverly combines the

flexibility of ACO to provide solutions for symmetric and asymmetric dynamic changes with the power of improvement of two good local search operators. Comparisons of the performance of the following strategies were investigated for DTSPs:

- $\mathcal{MM}$AS + US: for each best-so-far ant found by $\mathcal{MM}$AS, we apply the US local search operator pursuing improvements (detailed in [8]).
- $\mathcal{MM}$AS + LK: for each best-so-far ant found by $\mathcal{MM}$AS, we apply the LK local search operator pursuing improvements (detailed in [8]).
- HULK: proposed hyper-heuristic that combines ACO with the LSOs LK and US, biased by a self-adjusting weighted roulette wheel (detailed in Algorithm 2).

The initial parameters for tested methods use the following values: $\alpha = 1$, $\beta = 5$, $\gamma = 0.5$, $\delta = 0.5$, $\rho = 0.8$ and $\omega = 50$ (number of ants). In addition to the instances of the KROA family, DTSPs are generated from six originally symmetric static benchmark instances obtained from TSPLIB (Available from http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/, accessed on 9 August 2021): `d198`, `pcb442`, `u574`, `pcb1173`, `rat783` and `lin318`, using the dynamic generator described in Section 2.2. The first four benchmark instances originate from the operation of drilling holes in printed circuit boards [50], the fifth benchmark instance arises from rattled grid [51] and the sixth benchmark instance from the travel cost between cities [34].

The frequency of change f was set to change every 100 algorithmic evaluations, and the amplitude of change m was set to 0.05, 0.1, 0.2 and 0.4, denoting small to medium changing environments; we also allowed blocking up to 1% of the arcs selected for dynamic change trying to keep at least one Hamiltonian cycle. On the whole, there were 72 instances to be tested as follows: a series of four DTSP test cases were constructed from each stationary instance, for symmetric and asymmetric changes, to systematically investigate the performance of the algorithms (all asymmetric problem instances had an extension of `.atsp` at the end of the problem label). For the asymmetric instances only asymmetric dynamic changes were allowed. Thirty independent runs for each algorithm were performed on a DTSP, kept at the same set of random seeds. For each run, we allowed 100 environmental changes, storing the best-so-far solution for each of them. All seeds used in the random generators were kept for all environmental changes, and the random numbers were uniformly distributed. To be fair, all methods performed the same number of evaluations. The proportional evaluations required when applying LSOs were added to the total evaluations of the algorithms.

To evaluate the methods overall performance, the *offline performance* [52] is used as defined in Equation (15).

$$\bar{P}_{offline} = \frac{1}{E} \sum_{t=1}^{E} \phi(\pi^{bs}, t), \tag{15}$$

where E is the total number of evaluations and π^{bs} is the best-so-far solution quality after each dynamic change.

4.2. Experimental Results and Discussion

We first present Table 1, containing the results of 24 instances from the KROA family, without arc blocking. The experiments show that HULK outperforms the other methods in all symmetric and asymmetric instances and $\mathcal{MM}$AS + US have better performance than $\mathcal{MM}$AS + LK. From this first set of instances, we can already observe HULK efficiency.

In the literature, we have not found results for asymmetric instances from the KROA family. Therefore, we choose four algorithms to compare HULK with a magnitude of change $m = 0.1$. They use immigrant schemes (that arise as the most promising mechanisms due to their structural simplicity even superior in computational performance) [35], RIACO (random ACO), EIACO (elitism-based immigrants ACO) and MIACO (memory-based immigrants ACO) are detailed in [36]. The last one is the best result among the four strategies presented in [27] of the procedure named ALNS (adaptive-large-neighborhood Search-based immigrant schemes). Obeserve that ALNS outperforms the other three

immigrant schemes, but HULK is better than the best ALNS for at least 6.8% (KROA100.tsp) and at most 14.6% (KROA200.tsp). Since LSO has an asymptotic behavior considering its initial solution, i.e., it will have at least the same value as its initial solution, the comparisons presented in Table 2 show that HULK is much better than any other immigrant scheme approach. Thus, the remaining tests will compare HULK to the best procedures presented in [8] in instances with higher dimensions, barely explored previously in the literature, and we will show another variant of DTSPs considering arc blocking when generating the dynamic changes, as one extension of the results in [8]. The problems considering arc blocking seem to be more difficult to solve—see Tables 1 and 2 (KROA100 family), where HULK has outperformed the compared methods in all cases. However, when we include arc blocking, the superiority of HULK is still evident, but we observe in Tables 3 and 4 that for KROA100.tsp and KROA100.atsp there are some cases in which HULK loses to other compared methods but always by less than 1.0%.

Table 1. Experimental results regarding the average offline performance of the presented methods on symmetric DTSPs (upper half) and asymmetric DTSPs (lower half) for instances from KROA family without arc blocking. The number between brackets means the distance in percentage from the current value to the best value, in boldface.

Problem Instance	m	$\mathcal{MM}$AS + LK	$\mathcal{MM}$AS + US	HULK
KROA100.tsp	0.05	20,500.7 (1.2)	20,275.6 (0.1)	**20,262.9**
	0.1	20,101.1 (0.3)	20,079.3 (0.2)	**20,045.5**
	0.2	20,439.7 (2.7)	19,985.4 (0.4)	**19,899.1**
	0.4	20,323.7 (2.4)	20,013.0 (0.8)	**19,852.8**
KROA150.tsp	0.05	27,929.4 (2.4)	25,350.3 (0.1)	**25,318.9**
	0.1	25,564.0 (2.1)	25,079.8 (0.2)	**25,028.1**
	0.2	25,605.1 (1.8)	25,212.7 (0.2)	**25,158.6**
	0.4	25,538.7 (3.0)	24,868.3 (0.3)	**24,803.8**
KROA200.tsp	0.05	27,942.6 (2.4)	27,334.7 (0.1)	**27,299.6**
	0.1	27,945.7 (1.7)	27,521.8 (0.2)	**27,465.4**
	0.2	28,343.7 (2.9)	27,586.8 (0.2)	**27,531.9**
	0.4	28,407.6 (3.4)	27,545.5 (0.2)	**27,482.5**
KROA100.atsp	0.05	20,340.2 (2.3)	19,937.4 (0.3)	**19,876.4**
	0.1	20,124.9 (1.9)	19,864.8 (0.6)	**19,755.3**
	0.2	20,109.1 (0.9)	20,014.6 (0.6)	**20,028.3**
	0.4	20,175.0 (3.0)	19,768.2 (0.9)	**19,593.8**
KROA150.atsp	0.05	25,471.1 (1.3)	25,241.8 (0.3)	**25,133.7**
	0.1	25,382.5 (1.1)	25,219.6 (0.5)	**25,098.6**
	0.2	25,277.0 (1.4)	25,078.4 (0.6)	**24,933.4**
	0.4	25,320.4 (2.0)	25,036.3 (0.9)	**24,824.0**
KROA200.atsp	0.05	28,210.2 (1.7)	27,821.4 (0.3)	**27,725.9**
	0.1	28,273.7 (2.1)	27,825.8 (0.4)	**27,703.2**
	0.2	28,169.9 (1.8)	27,868.3 (0.7)	**27,676.0**
	0.4	28,498.2 (3.6)	27,742.0 (0.9)	**27,498.0**

Table 2. Comparison of HULK and four immigrant schemes from literature. The number between brackets means the distance in percentage from the current value to the best value, in boldface.

Instance	HULK	RIACO	EIACO	MIACO	ALNS
KROA100.tsp	**20,045.5**	23,635.8 (17.9)	23,417.2 (16.8)	23,398.7 (16.7)	21,407.5 (6.8)
KROA150.tsp	**25,028.1**	30,343.8 (21.2)	29,892.6 (19.4)	29,893.0 (19.4)	28,203.7 (12.7)
KROA200.tsp	**27,465.4**	34,203.1 (24.5)	33,496.6 (22.0)	33,576.9 (22.3)	31,468.3 (14.6)

Tables 3 and 4 present the offline performance results of all methods compared in this paper considering all DTSPs and the different magnitudes of dynamic changes, for sym-

metric and asymmetric problems, respectively. Table 5 presents the statistical analysis of the results, performing pairwise *Wilcoxon* rank-sum statistical tests with a significance level of 0.05 (see Julia library: HypothesisTests.jl, at https://juliastats.org/HypothesisTests.jl/stable/nonparametric/, accessed on 22 October 2021). To better visualize the results, we apply the following symbols: "+" (if the first algorithm is better than the second one)"++" (if the first algorithm is significantly better than the second one), "−" (if the second algorithm is better than the first one), "− −" (if the second algorithm is significantly better than the first one) and "∼" (if there is no significant difference between them). It is significantly better when the statistics show difference between the algorithms and the percentage difference between the average offline performance is greater than 2.0%, and when they have no significant difference when the statistics show it or the percentage difference between the average offline performance is less or equal to 0.1%.

Table 3. Experimental results regarding the average offline performance of the presented methods on symmetric DTSPs. The number between brackets means the distance in percentage from the current value to the best value, in boldface.

Problem Instance	m	$\mathcal{MM}$AS + LK	$\mathcal{MM}$AS + US	HULK
KROA100.tsp	0.05	20,383.9 (0.7)	20,254.8 (0.1)	**20,235.5**
	0.1	20,457.7 (0.9)	20,334.4 (0.3)	**20,280.3**
	0.2	20,482.0 (2.0)	20,178.2 (0.5)	**20,073.3**
	0.4	20,373.5 (1.8)	20,188.4 (0.8)	**20,022.9**
KROA150.tsp	0.05	25,477.0 (0.8)	25,347.2 (0.3)	**25,274.2**
	0.1	25,635.1 (2.4)	25,189.4 (0.6)	**25,043.1**
	0.2	25,758.1 (2.5)	25,419.6 (1.2)	**25,128.6**
	0.4	25,800.4 (3.3)	25,275.1 (1.2)	**24,969.3**
KROA200.tsp	0.05	28,118.8 (2.2)	27,524.4 (0.0)	**27,521.0**
	0.1	28,250.7 (2.2)	**27,645.6**	27,649.7 (0.0)
	0.2	28,814.1 (4.4)	**27,595.1**	27,842.3 (0.9)
	0.4	29,038.8 (4.8)	28,067.7 (1.3)	**27,705.5**
d198.tsp	0.05	**13,280.9**	14,157.0 (6.6)	13,403.0 (0.9)
	0.1	**13,078.4**	13,837.3 (5.8)	13,393.0 (2.4)
	0.2	**13,147.6**	14,014.3 (6.6)	13,513.9 (2.8)
	0.4	**12,943.2**	13,887.8 (7.3)	38,842.3 (3.1)
lin318.tsp	0.05	39,520.4 (0.4)	40,624.7 (3.2)	**39,355.2**
	0.1	39,371.9 (0.3)	40,615.7 (3.5)	**39,252.9**
	0.2	39,057.7 (0.1)	40,312.8 (3.3)	**39,032.5**
	0.4	38,795.8 (0.0)	40,091.8 (3.3)	**38,793.8**
pcb442.tsp	0.05	**48,811.3**	49,709.1 (1.8)	48,944.7 (0.3)
	0.1	**48,776.4**	49,683.1 (1.9)	49,044.5 (0.5)
	0.2	**48,692.5**	49,770.8 (2.2)	49,359.8 (1.4)
	0.4	**48,501.4**	49,615.1 (2.3)	49,416.6 (1.9)
u574.tsp	0.05	37,456.7 (8.5)	35,920.9 (4.1)	**34,521.9**
	0.1	37,260.4 (8.1)	35,698.2 (3.6)	**34,473.1**
	0.2	36,997.4 (5.5)	35,564.3 (1.5)	**35,055.9**
	0.4	36,964.7 (5.9)	35,451.3 (1.6)	**34,907.4**
rat783.tsp	0.05	8173.1 (0.5)	8364.1 (2.9)	**8129.6**
	0.1	8064.4 (0.6)	8267.5 (3.1)	**8019.9**
	0.2	8034.6 (0.1)	8239.4 (2.7)	**8024.4**
	0.4	7995.4 (0.2)	8202.9 (2.8)	**7980.0**
pcb1173.tsp	0.05	**54,478.8**	55,930.7 (2.7)	54,639.2 (0.3)
	0.1	**54,321.4**	55,904.8 (2.9)	54,476.0 (0.3)
	0.2	54,419.8 (0.2)	55,991.1 (3.1)	**54,287.6**
	0.4	59,943.1 (10.8)	55,814.4 (3.2)	**54,105.2**

Table 4. Experimental results regarding the average offline performance of the presented methods asymmetric DTSPs. The number between brackets means the distance in percentage from the current value to the best value, in boldface.

Problem Instance	m	$\mathcal{MM}$AS + LK	$\mathcal{MM}$AS + US	HULK
KROA100.atsp	0.05	20,279.1 (1.5)	20,058.4 (0.4)	**19,981.4**
	0.1	20,051.2 (0.4)	20,120.6 (0.3)	**19,975.4**
	0.2	20,050.9 (0.9)	20,038.6 (0.8)	**19,875.4**
	0.4	20,090.3 (1.5)	19,978.9 (0.9)	**19,802.0**
KROA150.atsp	0.05	25,447.8 (1.3)	25,236.0 (0.5)	**25,109.0**
	0.1	25,277.0 (0.8)	25,291.1 (0.8)	**25,083.7**
	0.2	25,451.3 (2.6)	25,062.8 (1.0)	**24,808.1**
	0.4	25,268.0 (2.5)	24,907.7 (1.0)	**24,651.8**
KROA200.atsp	0.05	**28,110.2**	28,183.5 (0.3)	28,131.3 (0.1)
	0.1	28,170.0 (0.9)	27,988.0 (0.3)	**27,910.7**
	0.2	28,326.1 (2.6)	**27,595.1**	27,626.3 (0.0)
	0.4	28,433.7 (3.6)	27,796.9 (1.3)	**27,453.0**
d198.atsp	0.05	14,582.0 (5.5)	14,999.4 (8.6)	**13,815.4**
	0.1	14,489.4 (5.3)	14,831.4 (7.8)	**13,761.6**
	0.2	14,236.4 (2.9)	14,842.6 (7.3)	**13,834.8**
	0.4	13,789.3 (6.3)	14,578.9 (12.4)	**12,967.4**
lin318.atsp	0.05	41,768.9 (2.6)	41,924.9 (3.0)	**40,700.7**
	0.1	41,615.0 (3.0)	42,168.5 (4.4)	**40,394.4**
	0.2	41,587.3 (3.7)	42,457.0 (5.8)	**40,111.2**
	0.4	41,314.8 (3.5)	42,675.8 (6.9)	**39,928.6**
pcb442.atsp	0.05	51,549.2 (3.2)	50,997.1 (2.1)	**49,970.9**
	0.1	51,763.4 (2.9)	51,543.3 (2.5)	**50,305.6**
	0.2	51,694.0 (3.0)	51,844.5 (3.3)	**50,175.4**
	0.4	51,471.9 (3.3)	52,153.3 (4.6)	**49,846.4**
u574.atsp	0.05	37,363.4 (2.0)	38,305.9 (4.6)	**36,620.7**
	0.1	36,893.0 (0.8)	38,369.8 (4.0)	**36,587.3**
	0.2	36,775.8 (2.9)	38,385.6 (4.4)	**35,727.4**
	0.4	36,437.0 (2.8)	38,383.5 (5.3)	**35,442.5**
rat783.atsp	0.05	8678.9 (1.1)	8722.9 (1.6)	**8586.5**
	0.1	8649.5 (2.6)	8800.3 (4.4)	**8431.1**
	0.2	8567.1 (2.0)	8790.2 (4.7)	**8395.5**
	0.4	8491.7 (2.5)	9103.6 (9.8)	**8288.3**
pcb1173.atsp	0.05	60,343.1 (2.4)	61,194.5 (3.9)	**58,920.1**
	0.1	60,197.8 (2.7)	61,655.1 (5.1)	**58,636.8**
	0.2	60,027.8 (2.6)	62,008.5 (6.0)	**58,515.8**
	0.4	59,639.9 (2.3)	62,706.6 (7.6)	**58,270.7**

Table 5. Statistical results regarding the average offline performance of the presented methods on symmetric DTSPs (upper half) and asymmetric DTSPs (lower half), where $\mathcal{MM}$AS + LK and $\mathcal{MM}$AS + US are replaced by ALK and AUS, respectively.

Problem Instance	m	Symmetric			Asymmetric		
		HULK vs. ALK	**HULK vs.** AUS	ALK **vs.** AUS	**HULK vs.** ALK	**HULK vs.** AUS	ALK **vs.** AUS
KROA100	0.05	+	∼	−	+	+	−
	0.1	+	+	−	+	+	∼
	0.2	+	+	−	+	+	∼
	0.4	+	+	−	+	+	−
KROA150	0.05	+	+	−	+	+	−
	0.1	++	+	−	+	+	∼
	0.2	++	+	−	++	+	−
	0.4	++	+	−	++	+	−
KROA200	0.05	++	∼	− −	∼	+	+
	0.1	++	∼	− −	+	+	−
	0.2	++	∼	− −	++	∼	− −
	0.4	++	+	− −	++	+	− −
d198	0.05	−	++	++	++	++	++
	0.1	− −	++	++	++	++	++
	0.2	− −	++	++	++	++	++
	0.4	− −	++	++	++	++	++
lin318	0.05	+	++	++	++	++	+
	0.1	+	++	++	++	++	+
	0.2	∼	++	++	++	++	++
	0.4	∼	++	++	++	++	++
pcb442	0.05	−	+	+	++	++	−
	0.1	−	+	+	++	++	−
	0.2	−	+	++	++	++	+
	0.4	−	+	++	++	++	+
u574	0.05	++	++	− −	+	++	++
	0.1	++	++	− −	+	++	++
	0.2	++	+	− −	++	++	++
	0.4	++	+	− −	++	++	++
rat783	0.05	+	++	++	+	+	+
	0.1	+	++	++	++	++	+
	0.2	∼	++	++	+	++	++
	0.4	+	++	++	++	++	++
pcb1173	0.05	−	++	++	++	++	+
	0.1	−	++	++	++	++	++
	0.2	+	++	++	++	++	++
	0.4	++	++	− −	++	++	++

To visualize and understand the behavior of the compared methods, we plotted for the last twenty environmental changes the dynamic average offline performance of the presented hyper-heuristic (HULK), $\mathcal{MM}$AS + US and $\mathcal{MM}$AS + LK, and see Figure 5 for symmetric, and Figure 6 for asymmetric dynamic changes. The problems plotted are rat783 and pcb1173, and we can observe that for the symmetrical change case all methods are competitive with an advantage to HULK and $\mathcal{MM}$AS + LK, but when we turn to the asymmetrical change case we can see the consistently better performance of HULK proving that the combination of two powerful LSOs can achieve much better solutions. We can also trace the following observations from the computational results.

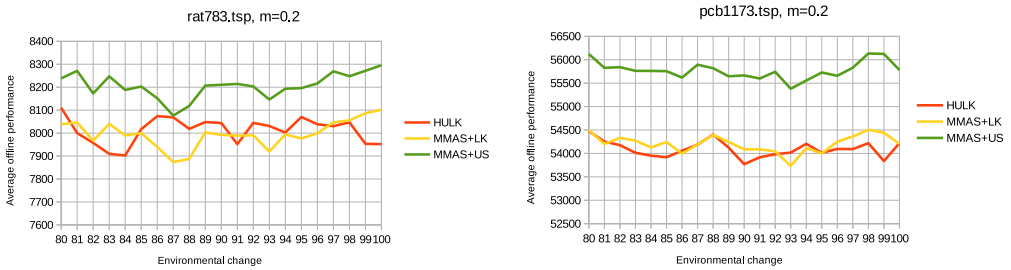

Figure 5. Dynamic average offline performance of the presented methods for symmetric DTSPs with $m = 0.2$.

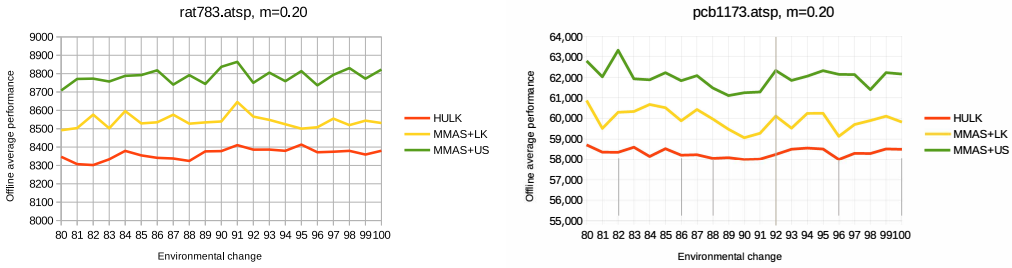

Figure 6. Dynamic average offline performance of the presented methods for asymmetric DTSPs with $m = 0.2$.

Considering the symmetric environmental changes (Table 3), HULK performs better in 66.7% of test cases and $\mathcal{MM}$AS + LK performs better in the rest, but there are two where $\mathcal{MM}$AS + US wins. In terms of numbers, HULK wins in 24 out of 32 instances; in seven the percentage difference is less than 1.0%, and for the other five the percentage difference remains between 1.0% and 3.1%. There are situations where $\mathcal{MM}$AS + US outperforms $\mathcal{MM}$AS + LK, e.g., `u574.tsp` and the entire KROA family, but there are only two cases where $\mathcal{MM}$AS + US outperform HULK. We can also observe Table 5, which shows that HULK always has a similar or superior quality compared with $\mathcal{MM}$AS + US and in some cases loses to $\mathcal{MM}$AS + LK. This alternation performance was expected since both LSOs were originally developed to deal with symmetric TSP, taking advantage of the properties of instances that use Euclidean distances and respect triangle inequality.

However, when we observe the asymmetric environmental changes (Table 4) we note a total HULK superiority outperforming $\mathcal{MM}$AS + LK and $\mathcal{MM}$AS + US in 94.4% of test cases. Observing Table 5, we can see that there are only two cases where HULK has worse performance, i.e., $\mathcal{MM}$AS + LK (`KROA200.atsp`, $m = 0.05$) and $\mathcal{MM}$AS + US (`KROA200.atsp`, $m = 0.2$), where the statistics show similar performance between them. This behavior shows how beneficial the clever combination of LSOs is, which allows for the hyper-heuristic to take advantage of the best aspects of each LSO and proves that $\mathcal{MM}$AS can retain crucial information when a dynamic change occurs. Due to this fact, $\mathcal{MM}$AS is able to provide good initial solutions for the local search operators, and HULK can use the best features of both local search operators to work in a very robust way in both symmetric and asymmetric dynamic changes environments, as expected. The complete results are presented in Tables 3 and 4 and the statistical comparisons in Table 5.

Differently from what was presented in [8], we included arc blocking on the dynamic changes and produced environments much closer to reality, creating a new and more difficult problem to solve. We chose two of the best LSOs (LK and US) for TSP and adapted them to work with both symmetric and asymmetric environmental changes. So, we become able to take advantage of the synergy created between them through hyper-heuristic based on ACO. We can also noticed from the results that the HULK performance is independent of the frequency and the magnitude of environmental changes. The results showed that

$\mathcal{MM}$AS + US and $\mathcal{MM}$AS + LK have similar behavior in favor of one, or sometimes another (for symmetric and asymmetric dynamic changes). This probably happens because they are based on arc exchange operations taking advantage of TSP properties. $\mathcal{MM}$AS + LK outperforms HULK in a few symetric cases (d198.tsp, pcb442.tsp and pcp1173.tsp), while $\mathcal{MM}$AS + US shows a performance similar to HULK only in a few cases (all of them in KROA family).

HULK shows complete superiority in asymmetric dynamic changes, and it is better in symmetric dynamic changes. In general, it achieves the best results in more than 80.0% of the cases and improves its performance when the problem dimension grows. The way that HULK combines the LSO significantly improves the method performance with no considerable additional computational effort. HULK shows how to improve the potentiality of LSO by responsively interacting between both and the choice of the parameters γ and δ shows itself to be effective with two LSO. The application of both LSOs in sequence sometimes improves the overall performance of HULK. For future research, we can choose more LSO and add more intelligence to HULK, directing the search to unexplored regions in the solution space.

5. Conclusions

This paper presented the hyper-heuristic HULK based on ACO and LSOs. We have used one of the best ACO variants named $\mathcal{MM}$AS and integrated it with two powerful LSOs (LK and US). This integration enabled the adaptation capabilities of $\mathcal{MM}$AS in dynamic environmental changes and the solution improvements of LSO that led to better exploitation and exploration in the search space. HULK has shown better performance than other compared methods in more than 80% of tested instances (66.7% considering symmetric changes and 94.4% for asymmetric changes). The idea of using an adaptive weighted roulette wheel to chose among the LSO available proved to be an important issue since when there was some change in the LSO during the same dynamic environment, the ACO retained the information due to the previous one and this synergy could direct the solution search to unexplored areas in the search space. We also applied both LSOs in sequence when some stagnation was detected. This behavior can be better seen when we analyze each environment and see how HULK improves the method intelligence. The method of adjusting the weights during the execution of HULK always seeks to favor the best LCO at the current moment, using as a guide the average of up to three latest solutions, but never leaving a single LSO no chance of being chosen. The TSP was the base problem used to generate the dynamic test instances; to make the problem more realistic, we included the possibility of blocking arcs, in both senses if the dynamic change was symmetric and if the change was asymmetric in just one sense.

At the beginning of the experiments, we compared HULK with other ACO with immigrant schemes found in the literature. HULK outperformed them by a minimal of 6.8%. So, we turned our attention to instances of higher dimensions, and to the best of our knowledge, ACO-based methods for DOPs have never been mentioned before in the literature. We also extended the dynamic changes, including arc blocking and asymmetry, making the instances more difficult to solve. Section 4 analyzes in detail the behavior of all methods exploring the results from Tables 1–5, showing the superiority of HULK mainly in asymmetric environments.

Finally, we can conclude that HULK has had better performance than $\mathcal{MM}$AS + LK $\mathcal{MM}$AS + LK in solving dynamic symmetric changes and even better performance in dynamic asymmetric changes. In HULK, both LSOs can work together, improving the search for solutions capacity. In future research, we can more widely explore the parameters to choose the most suitable LSO, and the possibility exists to include more LSOs. The idea of HULK is easily adapted to cope with other DOPs such as dynamic routing problems that interact very well with ACO and have a wide variety of LSOs in the literature.

Author Contributions: The authors contributed equally to this work. All authors have read and agreed to the published version of the manuscript.

Funding: The work of Felipe Martins Müller was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brasil (CAPES)—Finance Code 2380-14-5. Iaê Santos Bonilha is a Doctor Student partially supported by CAPES.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The instances are available at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ (accessed on 22 October 2021) and the computational codes are available by request.

Acknowledgments: The authors thanks the anonymous reviewers for their valuable contributions.

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

References

1. Burke, E.K.; Hyde, M.; Kendall, G.; Ochoa, G.; Özcan, E.; Woodward, J.R. A classification of hyper-heuristic approaches. In *Handbookof Metaheuristics*; Gendreau, M., Potvin, J.-Y., Eds.; Springer: New York, NY, USA, 2010; pp. 449–468.
2. Macias-Escobar, T.; Dorronsoro, B.; Cruz-Reyes, L.; Rangel-Valdez, N.; Gómez-Santillán, C. A Survey of hyper-heuristics for dynamic optimization problems. In *Intuitionistic and Type-2 Fuzzy Logic Enhancements in Neural and Optimization Algorithms: Theory and Applications*; Castillo, O., Melin, P., Kacprzyk, J., Eds.; Studies in Computational Intelligence; Springer: Cham, Switzerland, 2020; Volume 862, pp. 463–477._33 [CrossRef]
3. Dorigo, M.; Stützle, T. *Ant Colony Optimization*; MIT Press: Cambridge, MA, USA, 2004; 304p.
4. Applegate, D.L.; Bixby, R.E.; Chvatal, V.; Cook, W.J. *The Traveling Salesman Problem: A Computational Study*; Princeton University Press: Princeton, NJ, USA, 2007; 608p.
5. Jin, Y.; Branke, J. Evolutionary optimization in uncertain environments—A survey. *IEEE Trans. Evol. Comput.* **2005**, *9*, 303–317. [CrossRef]
6. Emelogu, A.; Chowdhury, S.; Marufuzzaman, M.; Bian, L.; Eksioglu, B. An enhanced sample average approximation method for stochastic optimization. *Int. J. Prod. Econ.* **2016**, *182*, 230–252. [CrossRef]
7. Hart, E.; Ross, P. An immune system approach to scheduling in changing environments. In Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation, Orlando, FL, USA, 13–17 July 1999.
8. Mavrovouniotis, M.; Bonilha, I.S.; Müller, F.M.; Ellinas, G.; Polycarpou, M. Effective ACO-Based Memetic Algorithms for Symmetric and Asymmetric Dynamic Changes. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 10–13 June 2019; pp. 2567–2574. [CrossRef]
9. Mavrovouniotis, M.; Müller, F.M.; Yang, S. An ant colony optimization based memetic algorithm for the dynamic travelling salesman problem. In Proceedings of the Conference on Genetic and Evolutionary Computation Conference (GECCO2015), Madrid, Spain, 11–15 July 2015; pp. 49–56.
10. Mavrovouniotis, M.; Müller, F.M.; Yang, S. Ant colony optimization with local search for the dynamic travelling salesman problems. *IEEE Trans. Cybern.* **2017**, *47*, 1743–1756. [CrossRef]
11. Mavrovouniotis, M.; Yang, S. Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors. *Appl. Soft Comput.* **2013**, *13*, 4023–4037. [CrossRef]
12. Mavrovouniotis, M.; Yang, S. A memetic ant colony optimization algorithm for the dynamic travelling salesman problem. *Soft Comput.* **2011**, *15*, 1405–1425. [CrossRef]
13. Sama, M.; Pellegrini, P.; D'Ariano, A.; Rodriguez, J.; Pacciarelli, D. Ant colony optimization for the real-time train routing selection problem. *Trans. Res. Part B Methodol.* **2016**, *85*, 89–108. [CrossRef]
14. Cheng, H.; Yang, S. Genetic algorithms with elitism-based immigrants for dynamic load balanced clustering problem in mobile ad hoc networks. In Proceedings of 2011 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE), Paris, France, 11–15 April 2011.
15. Cheng, H.; Yang, S. Genetic algorithms with immigrants schemes for dynamic multicast problems in mobile ad hoc networks. *Eng. Appl. Artif. Intell.* **2010**, *23*, 806–819. [CrossRef]
16. Cheng, H.; Yang, S. Multi-population genetic algorithms with immigrants scheme for dynamic shortest path routing problems in mobile ad hoc networks. In Proceedings of the European Conference on the Applications of Evolutionary Computation, Istanbul, Turkey, 7–9 April 2010.
17. Du, W.; Li, B. Multi-strategy ensemble particle swarm optimization for dynamic optimization. *Inf. Sci.* **2010**, *178*, 3096–3109. [CrossRef]
18. Eddaly, M.; Jarboui, B.; Siarry, P. Combinatorial particle swarm optimization for solving blocking flowshop scheduling problem. *J. Comput. Des. Eng.* **2016**, *3*, 295–311. [CrossRef]
19. Liu, J. Rank-based ant colony optimization applied to dynamic traveling salesman problems. *Eng. Optim.* **2005**, *37*, 831–847. [CrossRef]

20. Guntsch, M.; Middendorf, M. Pheromone modification strategies for ant algorithms applied to dynamic TSP. In Proceedings of the Workshops on Applications of Evolutionary Computation, Como, Italy, 18–20 April 2001.
21. Eyckelhof, C.J.; Snoek, M. Ant systems for a dynamic TSP. In Proceedings of the International Workshop on Ant Algorithms, Brussels, Belgium, 12–14 September 2002.
22. Mavrovouniotis, M.; Yang, S.; Yao, X. Multi-colony ant algorithms for the dynamic travelling salesman problem. In Proceedings of the IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments, Orlando, FL, USA, 9–12 December 2014.
23. Guntsch, M.; Middendorf, M. A population based approach for ACO. In Proceedings of the Workshops on Applications of Evolutionary Computation, Kinsale, Ireland, 3–4 April 2002.
24. Angus, D. Niching for ant colony optimisation. *Biol.-Inspired Optim. Methods* **2009**, *210*, 165–188.
25. Angus, D. Niching for population-based ant colony optimization. In Proceedings of the Second IEEE international Conference on e-Science and Grid Computing, Amsterdam, The Netherlands, 4–6 December 2006.
26. Mavrovouniotis, M.; Yang, S. Memory-based immigrants for ant colony optimization in changing environments. In Proceedings of the European Conference on the Applications of Evolutionary Computation, Torino, Italy, 27–29 April 2011.
27. Chowdhury, S.; Marufuzzaman, M.; Tunc, H.; Bian, L.; Bullington, W. A modified ant colony optimization algorithm to solve a dynamic traveling salesman problem: A case study with drones for wildlife surveillance. *J. Comput. Des. Eng.* **2019**, *6*, 368–386. [CrossRef]
28. Dorigo, M.; Maniezzo, V.; Colorni, A. Ant system: Optimization by a colony of cooperating agents. *IEEE Trans. Syst. Man Cybern. Part B Cybern.* **1996**, *26*, 29–41. [CrossRef] [PubMed]
29. Stützle, T.; Hoos, H. $\mathcal{MAX}$–$\mathcal{MIN}$ ant system. *Future Gener. Comput. Syst.* **2000**, *16*, 889–914. [CrossRef]
30. Ulder, N.; Aarts, E.; Bandelt, H.-J.; van Laarhoven, P.; Pesch, E. Genetic local search algorithms for the traveling salesman problem. In *Parallel Problem Solving from Nature*; Schwefel, H.-P., Männer, R., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1991; Volume 496, pp. 109–116.
31. Stodola, P.; Michenka, K.; Nohel, J.; Rybanský, M. Hybrid Algorithm Based on Ant Colony Optimization and Simulated Annealing Applied to the Dynamic Traveling Salesman Problem. *Entropy* **2020**, *22*, 884. [CrossRef]
32. Gendreau, M.; Hertz, A.; Laporte, G. New insertion and postoptimization procedures for the traveling salesman problem. *Oper. Res.* **1992**, *40*, 1086–1094. [CrossRef]
33. Lin, S.; Kernighan, B. An effective heuristic algorithm for the traveling salesman problem. *Oper. Res.* **1973**, *21*, 498–516. [CrossRef]
34. Helsgaun, K. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *Eur. J. Oper. Res.* **2000**, *126*, 106–130. [CrossRef]
35. Mavrovouniotis, M.; Yang, S. Ant colony optimization with immigrants schemes in dynamic environments. In Proceedings of the International Conference on Parallel Problem Solving from Nature, Krakov, Poland, 11–15 September 2010.
36. Mavrovouniotis, M.; Yang, S. Adapting the pheromone evaporation rate in dynamic routing problems. In *Applications of Evolutionary Computation*; Esparcia-Alcázar, A., Ed.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2013; Volume 7835, pp. 606–615.
37. Tinós, R.; Whitley, D.; Howe, A. Use of explicit memory in the dynamic traveling salesman problem. In Proceedings of the Conference on Genetic and Evolutionary Computation, Vancouver, BC, Canada, 12–16 July 2014; pp. 999–1006.
38. Dantzig, G.; Fulkerson, R.; Johnson, S. Solution of a large-scale traveling-salesman problem. *J. Oper. Res. Soc. Am.* **1954**, *2*, 393–410. [CrossRef]
39. Roberti, R.; Toth, P. Models and algorithms for the Asymmetric Traveling Salesman Problem: An experimental comparison. *EURO J. Transp. Logist.* **2012**, *1*, 113–133. [CrossRef]
40. Mele, U.J.; Gambardella, L.M.; Montemanni, R. A New Constructive Heuristic Driven by Machine Learning for the Traveling Salesman Problem. *Algorithms* **2021**, *14*, 267. [CrossRef]
41. Stützle, T.; Hoos, H. $\mathcal{MAX}$–$\mathcal{MIN}$ ant system and local search for the traveling salesman problem. In Proceedings of the IEEE International Conference on Evolutionary Computation, Indianapolis, IN USA, 13–16 April 1997; pp. 309–314.
42. Jonker, R.; Volgenant, T. Transforming asymmetric into symmetric traveling salesman problems. *Oper. Res. Lett.* **1983**, *2*, 161–163. [CrossRef]
43. França, P.M.; Gendreau, M.; Laporte, G.; Müller, F.M. A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *Int. J. Prod. Econ.* **1996**, *43*, 79–89. [CrossRef]
44. Mavrovouniotis, M.; Li, C.; Yang, S. A survey of swarm intelligence for dynamic optimization: Algorithms and applications. *Swarm Evol. Comput.* **2017**, *33*, 1–17. [CrossRef]
45. Gambardella, L.M.; Dorigo, M. Ant-Q: A reinforcement learning approach to the traveling salesman problem. In Proceedings of the International Conference on Machine Learning, Tahoe City, CA, USA, 9–12 July 1995; pp. 252–260.
46. Angus, D.; Hendtlass, T. Ant colony optimisation applied to a dynamically changing problem. In *Developments in Applied Artificial Intelligence*; Hendtlass, T., Ali, M., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2002; Volume 2358, pp. 618–627.
47. Guntsch, M.; Middendorf, M. Applying population based ACO to dynamic optimization problems. In *Ant Algorithms*; Dorigo, M., Di Caro, G., Sampels, M., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2002; Volume 2463, pp. 111–122.

48. Bonilha, I.S.; Mavrovouniotis, M.; Müller, F.M.; Ellinas, G.; Polycarpou, M. Ant colony optimization with heuristic repair for the dynamic vehicle routing problem. In Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, ACT, Australia, 1–4 December 2020; pp. 313–320.
49. Krolak, P.; Felts, W.; Nelson, J. A Man-Machine Approach Toward Solving the Generalized Truck-Dispatching Problem. *Transp. Sci.* **1972**, *6*, 149–170. [CrossRef]
50. Grötschel, M.; Jünger, M.; Reinelt, G. Optimal control of plotting and drilling machines: A case study. *ZOR-Methods Models Oper. Res.* **1991**, *35*, 61–84. [CrossRef]
51. Grötschel, M.; Pulleyblank, W.R. Clique Tree Inequalities and the Symmetric Travelling Salesman Problem. *Math. Oper. Res.* **1986**, *11*, 537–569. [CrossRef]
52. Branke, J.; Schmeck, H. Designing evolutionary algorithms for dynamic optimization problems. In *Advances in Evolutionary Computing*; Ghosh, A., Tsutsui, S., Eds.; Natural Computing Series; Springer: Berlin/Heidelberg, Germany, 2003; pp. 239–262.